SESSION

RESOURCE MANAGEMENT, RESOURCE ALLOCATION, SCHEDULING, AND DATA MANAGEMENT

Chair(s)

TBA

Comparison of Energy-Constrained Resource Allocation Heuristics under Different Task Management Environments

Bhavesh Khemka*, Ryan Friese*, Sudeep Pasricha*†, Anthony A. Maciejewski*, Howard Jay Siegel*†,
Gregory A. Koenig‡, Sarah Powers‡, Marcia Hilton§, Rajendra Rambharos§, Mike Wright§, and Steve Poole§
*Department of Electrical and Computer Engineering,
†Department of Computer Science,
Colorado State University,
Fort Collins, CO 80523, USA
‡Oak Ridge National Laboratory,
Oak Ridge, TN 37831, USA
§Department of Defense,
Washington, DC 20001, USA

Email: {Bhavesh.Khemka, Ryan.Friese, Sudeep, AAM, HJ}@colostate.edu, {Koenig, PowersSS}@ornl.gov,
mmskizig@verizon.net, Jendra.Rambharos@gmail.com, Michael.Wright4@comcast.net, SWPoole@gmail.com

***Abstract*—There is a growing need for energy-efficiency in high performance computing, especially with systems approaching exascale levels. The Extreme Scale Systems Center at Oak Ridge National Laboratory faces a need for resource management techniques that maximize the performance of the system while satisfying an energy budget. The performance of the system is measured as the total "utility" earned from completing tasks. Utility is represented as a time-varying importance of a task. We perform an in-depth examination into the energy-constrained utility maximization problem by comparing the performance of resource management techniques in two different task management environments: queued and polled. In one environment, tasks are queued for execution on the different machines and certain tasks are not allowed to be re-scheduled. In the other environment, machines are polled at regular intervals and each idle machine is only assigned one task. Multiple First Come First Served heuristics are designed and compared against other heuristics. We design a new adaptive energy filter that can be used with any of the heuristics to bring energy awareness to them. This filtering technique can be readily deployed in any environment without the need of any off-line parameter tuning experiments. The filtering operation allows the heuristics to better regulate their energy expenditure in the energy constrained environment. The polled task management environment and our novel filtering technique give significant performance improvements for the heuristics while meeting the energy budget requirement.**

***Index Terms*—resource allocation; adaptive energy filtering; heterogeneous computing; energy-aware resource management heuristics; system utility**

Corresponding author: Bhavesh Khemka

I. Introduction

The need to solve applications of higher complexity with greater accuracy combined with the need for faster execution from high-performance computing (*HPC*) systems is resulting in higher energy consumption and costs to operate these systems. A recent National Resources Defense Council report showed that data centers in the U.S. consumed an estimated 91 billion kWh in 2013 (that is double the amount of electricity consumed by all of the households in New York City) and are on track to reach 140 billion kWh by 2020 [1]. Some data centers are now unable to increase their computing performance due to physical limitations on the availability of energy. For example, in 2010, Morgan Stanley, a global financial services firm based in New York, was physically unable to draw the energy needed to run a data center in Manhattan [2]. Many HPC systems are now being forced to execute with constraints on the amount of energy they can consume. The issue of increased energy consumption is estimated to significantly worsen as we approach exascale systems. As a result, there is a growing concern regarding energy needed to operate these systems (e.g., [3], [4]) and it is becoming increasingly important for system administrators to adopt energy-efficient workload execution policies.

This research builds on prior work that developed energy-aware resource management techniques with the goal of maximizing the performance of a workload executing on an energy-constrained heterogeneous HPC system [5]. In this new work, we analyze the performance of the resource management techniques in *task management environments* that differ in their policies of which tasks and machines can be considered by the scheduling techniques. We study and contrast a queued and a polled environment. Also, we enhance an energy filter technique (introduced in [5]) by removing the need to determine its parameters empirically and making it adaptive by

using current system information. Such a filtering technique can be deployed directly into any environment without the need to be tuned off-line. The goal of the filtering technique is to ignore allocation choices that use more energy than an estimated fair-share. This improves the distribution of the budgeted energy across the constrained time period.

We model a compute facility and workload of interest to the Extreme Scale Systems Center (*ESSC*) at Oak Ridge National Laboratory (*ORNL*). The ESSC is a joint venture between the United States Department of Defense (*DoD*) and Department of Energy (*DOE*) to perform research and deliver tools, software, and technologies that can be integrated, deployed, and used in both DoD and DOE environments. This system incorporates heterogeneous compute resources that utilize a mix of different machines to execute workloads with diverse computational requirements. In such an environment, each task typically has different execution time and energy consumption characteristics when executed on different machines. We model our machines to have different ACPI performance states (P-states) in which tasks can execute [6].

Each task in the system has a monotonically-decreasing utility function associated with it that represents the task's *utility* (or value) as a function of the task's completion time. The system performance is measured in terms of *total utility earned*, which is the sum of utility earned by all completed tasks [7]. The goal for resource management techniques in this environment is to maximize the amount of utility earned during a period of time that has a constraint on the amount of energy that can be consumed. To keep our simulations tractable, we consider the time period of a day, but one could use any length of time (e.g., six hours, one month, one year). We compare and analyze the performance of our heuristics with different First Come First Served (*FCFS*) heuristics in different task management environments.

In summary, we make the following contributions: (a) the design of a novel adaptive energy filtering mechanism that can be readily deployed into any environment, (b) a comparative analysis of the advantages and disadvantages of a polled task management environment that can be used in HPC environments, and (c) a comparison of multiple FCFS heuristics that are typically used in real schedulers with smarter heuristics that can improve system performance.

The remainder of this paper is organized as follows. The next section discusses our system model and the problem we address. Section III describes the task management environments and our resource management techniques. Our simulation setup is detailed in Section IV. Section V discusses and analyzes our experimental results. We provide an overview of related work in Section VI. We finish with our conclusions and plans for future work in Section VII.

II. Problem Description

A. System Model

In this study, the system model is similar to the model described in [5]. We model a dynamic system where tasks arrive throughout the day and a resource manager maps the tasks to machines for execution. We consider an *oversubscribed* environment, i.e., the incoming workload exceeds the capacity of the computing system. The workload and computing system we model are based on the interests of the ESSC. Each task in the system has an associated utility function (introduced in [7]). The utility function of a task is a monotonically-decreasing function that represents the "value" of completing that task at different times. Figure 1 shows an example utility function for a task and highlights the utility that will be earned when the task is completed at two different times. The utility function of a task is assumed to be set by the user in collaboration with the system owner. The utility functions are described by three parameters: priority, urgency, and utility class [7]. *Priority* controls the overall importance of the task by setting the task's maximum (i.e., starting) utility value. *Urgency* sets the overall rate of decay for the utility function. The *utility class* allows the shape of the utility function to be modified by partitioning it into intervals and specifying shape modifiers for each of the intervals.

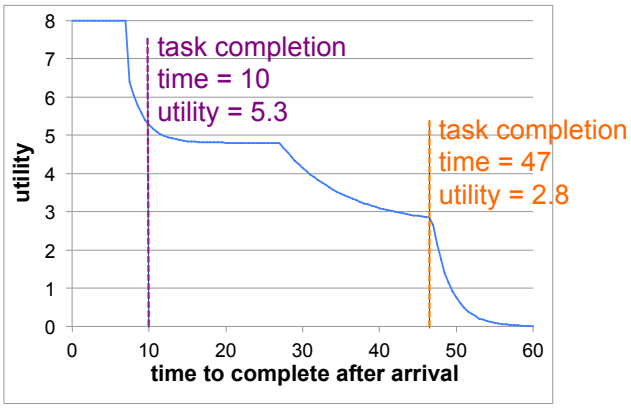


Fig. 1. A sample utility function showing the utility earned at two different completion times.

Our computing system environment consists of heterogeneous machines, where each machine belongs to a specific *machine type* (rather than a single large monolithic system, such as Titan [8]). Machines belonging to different machine types may differ in their microarchitectures, memory modules, and other system components. We model the machines to contain CPUs with dynamic voltage and frequency scaling (*DVFS*) enabled to utilize different ACPI performance states (*P-states*) that offer a trade-off between execution time and power consumption. We group tasks with similar execution time and power characteristics into *task types*. Tasks belonging to different task types may differ in characteristics such as computational intensity, memory intensity, I/O intensity, and memory access pattern. The application (task) developer specifies which task type the application falls into. The type of a task is not related to the utility function of the task. Because the system is heterogeneous, machine type A may be faster (or more energy-efficient) than machine type B for certain task types but slower (or less energy-efficient) for others. We model

general-purpose machine types and *special-purpose* machine types in our heterogeneous system [9]. The special-purpose machine types execute certain special-purpose task types much faster than the general-purpose machine types, although they may be incapable of executing the other task types.

We assume that for a task of type i on a machine of type j running in P-state k, we are given the *E*stimated *T*ime to *C*ompute ($ETC(i,j,k)$) and the *A*verage *P*ower *C*onsumption ($APC(i,j,k)$). It is common in the resource management literature to assume the availability of this information based on historical data or experiments [10]–[16]. The APC incorporates both the static power (not affected by the P-state of the task) and the dynamic power (different for different P-states). We can compute the *E*stimated *E*nergy *C*onsumption ($EEC(i,j,k)$) by taking the product of execution time and average power consumption, i.e., $EEC(i,j,k) = ETC(i,j,k) \times APC(i,j,k)$. ETC and APC values for the ESSC environment are not available to researchers. Therefore, for the simulation study conducted in this paper, we synthetically create ETC and APC matrices based on recommendations provided by ESSC and based on general trends of the workloads.

Tasks are assumed to be independent (they do not require inter-task communication) and can execute concurrently (each on a single machine, possibly with parallel threads). This is typical of many environments, such as [17]. We do not allow the preemption of tasks, i.e., once a task starts execution, it must execute until completion.

B. Problem Statement

With tasks dynamically arriving, the scheduler does not know the arrival time, type, or utility function of the next task. *The goal of the scheduler is to maximize the total utility that can be earned from completing tasks during a given period of time while satisfying an energy constraint (E) for that time period.* We use the duration of a day to keep the simulation time tractable. Instead of one day we could base our constraint on any interval of time (e.g., two hours, six months, a year). For ESSC, constraints on power (energy per time) are not a concern. As the system modeled is oversubscribed, the machines are never turned off, and therefore, the fraction of static versus dynamic power is not relevant.

III. Resource Management

A. Overview

Heuristics are commonly used to solve task to machine scheduling problems that have been shown to be NP-hard [18]. A *mapping event* occurs any time a scheduling decision has to be made. We use *batch-mode heuristics* that trigger mapping events after fixed time durations (one minute in our environment) after the previous mapping event completes [7], [19].

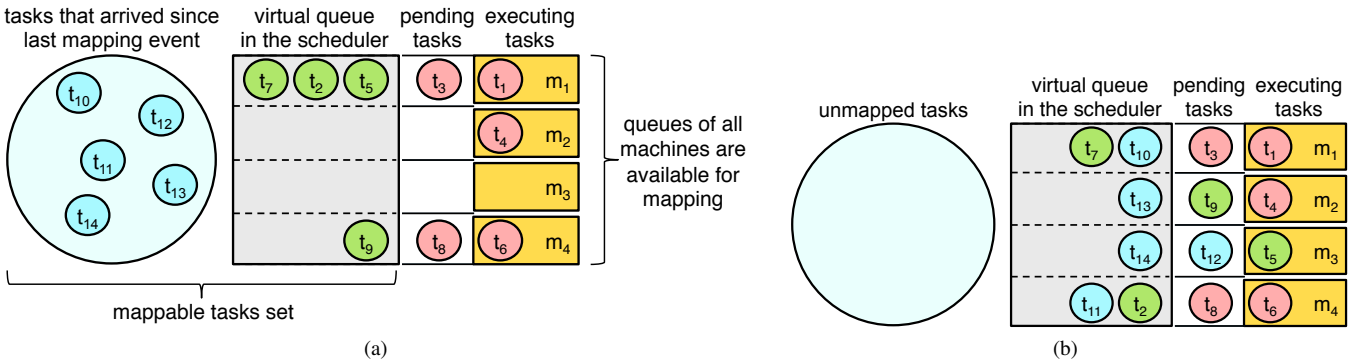
During a mapping event, three decision processes are executed. The first operation drops tasks that have low potential utility at the current time to allow the system to better tolerate high oversubcription scenarios. The second operation is an energy filtering technique. We adapt the technique from our work in [5] to control the energy expenditure by preventing tasks from using more than their "fair-share" of energy. We enhance the energy filtering technique to use more system information and enable it to automatically adjust its level of energy filtering without the need for any parameter tuning. The final operation during a mapping event does the actual mapping of tasks to machines in certain P-states using some heuristic approach.

We study two different task management environments: queued and polled. In the *queued* environment, each of the machines has a queue of tasks that it will execute in the queue order. The task that is next-in-line for execution on a machine is referred to as the *pending task*. All other tasks that are queued for the machines are said to be in the *virtual queues* of the scheduler. Figure 2a shows the state of a small example system prior to a mapping event with four machines and executing tasks, tasks in the pending slots, the virtual queues of the scheduler, and the tasks that have arrived since the last mapping event. At a mapping event in the queued environment, the batch-mode heuristics make scheduling decisions for a set of tasks comprising those that have arrived since the last mapping event and the ones that are currently in the virtual queues. This set of tasks is called the *mappable tasks set*. The batch-mode heuristics are not allowed to remap the pending tasks so that the machines do not idle if the currently executing tasks complete while the heuristic is executing. As we do not allow preemption, the currently executing tasks cannot be interrupted and therefore are not available for mapping. We refer to the pending and the currently executing tasks as the *unmovable tasks*. As there are queues for the machines, the heuristics consider all machines as *available* choices when performing mapping decisions. Figure 2b shows an example state of the machines immediately after the mapping event is performed.

In the *polled* environment, individual machines do not have queues. Rather, at each mapping event, the machines are polled to check if they are currently idle or if they are executing a task. Only the machines that are idle are considered to be "available" for scheduling during the mapping event. The "mappable tasks set" in this environment comprises tasks that were either unmapped in the previous mapping event or newly arrived tasks since the last mapping event. In the polled environment, the only "unmovable tasks" are the tasks that are currently executing. Figure 3a shows an example state of a four-machine system in a polled environment prior to a mapping event, and Figure 3b shows a possible state of the system immediately after the mapping event.

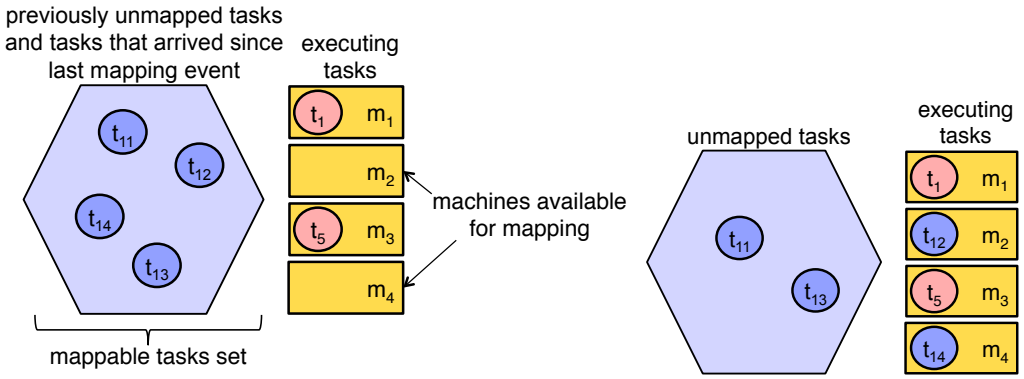
B. Dropping Low Utility Earning Tasks

We use a technique to drop tasks with low potential utility at the current time (introduced in our previous work [7]). *Dropping* a task means that it will never be mapped to a machine. Due to the oversubscribed environment, if a resource allocation heuristic tried to have all tasks execute, most of the task completion times would be so long that the utility of most tasks would decay significantly and be very small. This



(a) (b)

Fig. 2. Example state of a four-machine system in the "queued" environment (a) before and (b) immediately after a mapping event.



(a) (b)

Fig. 3. Example state of a four-machine system in the "polled" environment (a) before and (b) immediately after a mapping event.

would negatively impact users as well as the overall system performance. Given that the performance measure is the total utility achieved by summing the utilities of the completed tasks, dropping tasks leads to higher system performance, as well as more users that are satisfied.

The dropping operation determines the maximum possible utility that each mappable task could earn on any available machine assuming it can start as soon as possible, i.e., immediately after the unmovable tasks. If this utility is less than a *dropping threshold* (determined empirically), we drop this task from the set of mappable tasks. If the utility earned is not less than the threshold, the task remains in the mappable tasks set and is considered in the subsequent allocation decisions of the mapping event.

As our environment is oversubscribed, the number of tasks in the mappable tasks set increases quickly. With more tasks in the mappable set, the heuristics may take longer to perform their mapping decisions. This can delay the trigger of subsequent mapping events and result in poor performance because any newly arrived high utility tasks may not get serviced in a timely manner. Therefore, by dropping tasks with low potential utility, we reduce the size of the mappable tasks set and enable the heuristics to complete their execution quicker and as a result trigger subsequent mapping events sooner. This allows the heuristics to promptly service any newly arriving high utility-earning tasks.

C. Energy Filtering

The goal of our energy filter technique is to remove potential allocation choices (task/machine/P-state combinations) from a heuristic's consideration if the allocation choice consumes more energy than an estimated fair-share energy budget (e_{bud}). The value of the e_{bud} needs to adapt based on the energy remaining in the day and the time remaining in the day. Therefore, the value of the e_{bud} is recomputed at the start of every mapping event.

We denote e_{cons} as the total energy that has been consumed by the system in the current day, and e_{unmov} as the energy that is guaranteed to be consumed, i.e., by unmovable tasks. The total energy that can be scheduled by heuristics (without violating the day's energy constraint) is denoted by e_{rem}. It is computed as $e_{rem} = E - (e_{cons} + e_{unmov})$.

To estimate e_{bud}, the filter also needs to compute the time remaining in the day within which the above energy can be consumed. The *availability time* of a machine is set to either the completion time of the last unmovable task on the machine or the current time, whichever is later. We compute the total time remaining for computations (τ_{rem}) by summing across machines the difference between the end time of the day and the availability time of the machine.

The average of the execution time values and energy values of all task types, machine types, and P-states is represented as $\bar{\tau}$ and $\bar{e}$, respectively. The energy filtering technique needs

to estimate the total number of tasks that it can execute until the end of the day on average. Based on the time remaining, the estimated number of tasks that can complete on average was calculated as $\tau_{rem}/\bar{\tau}$ [5]. Similarly, based on the energy remaining, we now estimate the number of tasks that can complete on average as $e_{rem}/\bar{e}$, and use the minimum of these two ratios as the number of tasks that the system can complete on average (n):

$$n = min\left(\frac{\tau_{rem}}{\bar{\tau}}, \frac{e_{rem}}{\bar{e}}\right). \quad (1)$$

To control the strictness of the filtering technique, we use a multiplier (λ). e_{bud} is computed using:

$$e_{bud} = \lambda \times \frac{e_{rem}}{n}. \quad (2)$$

When n is determined by the energy remaining in the system in Equation 1 ($e_{rem}/\bar{e}$), the second term of the product in Equation 2 appropriately reduces to using the average energy consumption of a task ($\bar{e}$) to determine e_{bud}.

Instead of using a fixed value for λ that needs to be empirically determined by running multiple parameter tuning experiments [5], we enable it to adapt based on the current rate of energy consumption and the target energy consumption rate. We denote the total compute time available in the system at the start of the day as T and its value is calculated by simply multiplying the number of machines by the total time in a day. The adaptive value for λ (which is recomputed at the start of every mapping event) is calculated as:

$$\lambda = \frac{E/T}{\left(\frac{e_{cons} + e_{unmov}}{T - \tau_{rem}}\right)}. \quad (3)$$

The goal of this adaptive parameter is to distribute the energy usage throughout the day. The numerator in Equation 3 is the target energy consumption rate and the denominator is the current average rate of energy consumption. If the current average rate is lower than the target rate, then that leads to a larger value for λ, which allows more energy to be consumed by allocation choices. If the current average rate is higher than the target, the resulting smaller value of λ will only let low energy-consuming allocation choices pass through the filter. In this way, this adaptive technique automatically adjusts the level of filtering and can be deployed readily into any environment without the need for extensive off-line parameter tuning experiments.

D. Heuristics

We use the dropping operation with all the heuristics. We analyze the performance of the heuristics with and without the energy filtering technique. The heuristics are given the tasks and the task/machine/P-state choices that passed the dropping operation and the energy filtering technique (if used) to finally make assignments of the mappable tasks to the available machines. The *ready time* of a machine is the time by which it completes execution of the last task that is queued on it. In the polled environment, the ready time of the available machines is simply the current time. All of the heuristics progress iteratively, and in each iteration they make assignment for a mappable task to an available machine. The heuristics stop executing if either the set of mappable tasks is empty, or if there are no more available machines. In this work, we compare and analyze the performance of twelve heuristics. These heuristics assign tasks during mapping events while there still remains energy in the day. All heuristics ensure that the allocation they plan to make is a *valid assignment*, i.e., not assigning a task that cannot run on a particular special-purpose machine.

As most real-world schedulers assign tasks in an order based on their arrival time, we design and study the performance of a First Come First Served (*FCFS*) heuristic. The FCFS heuristic maintains a list of mappable tasks in an ascending order of arrival time. It then iteratively works through the list, each time making an assignment for the first task in the list. For the task being considered in each iteration, the heuristic assigns it to the available machine that has the earliest ready time that is both a valid assignment, and that has passed the energy filter in the fastest P-state (i.e., P-state 0). In the polled environment, the heuristic picks a machine that is immediately available. The assigned task is removed from the sorted list and the next task is considered. We call this the ***FCFS P-state 0*** heuristic. We implemented another version of this heuristic that examines if the slower P-states pass through the energy filter in case the fastest P-state does not. We call this the ***FCFS All P-states*** heuristic.

The system examined in this study is oversubscribed, and the utility of tasks may start to decay once they arrive. Therefore, we consider an alternative to the FCFS heuristic that gives higher preference to the latest arrived task. This is the Last Come First Served (*LCFS*) heuristic. The LCFS heuristic is similar to the FCFS heuristic except that it maintains a list of mappable tasks in a descending order of their arrival times. We call the version that only permits the fastest P-state as the ***LCFS P-state 0*** heuristic and the version that allows any P-state to be chosen as the ***LCFS All P-states*** heuristic.

To account for the different importance levels of the tasks, we design prioritized versions of the FCFS and the LCFS heuristics. We envision that a version of these heuristics are implemented in real-world schedulers where the submitted jobs have different priority levels. The ***Prioritized-FCFS P-state 0*** heuristic first groups the mappable tasks based on their priority level (i.e., value of their initial maximum utility). It maintains a list of the tasks within each group in an ascending order of arrival time. The heuristic then considers the highest priority level group that contains any tasks. Considering those tasks in order, it makes assignment for each task to the earliest available machine that is a valid assignment and that has passed through the energy filter in P-state 0. After considering all the tasks in this group, it then considers the next highest priority level group that contains any tasks and continually repeats this process. We designed another version of this heuristic that allows the other P-states to be considered as well. We call this the ***Prioritized-FCFS All P-states*** heuristic.

The prioritized versions of the LCFS heuristics are called the ***Prioritized-LCFS P-state 0*** and ***Prioritized-LCFS All P-states*** heuristics. These heuristics are similar to the Prioritized-FCFS heuristics with the difference that they maintain a list of tasks within each group in a descending order of their arrival times.

The ***Max Utility*** heuristic (designed based on the Min-Min technique [19]–[21]) gives preferences to mapping choices that earn the highest utility. The heuristic computes the utility that will be earned by each of the mappable tasks on each of the available machines in the different P-states. Each mappable task independently finds the machine/P-state choice that is valid, passes the energy filter, and that maximizes the utility earned. Among the different task/machine/P-state choices found, an assignment is made for the choice that earns the highest utility. The assigned task is removed from the set of mappable tasks and the process is repeated. This heuristic examines the utility function to find the utility that will be earned at the task completion time as opposed to the Prioritized heuristics that only look at the starting utility value of the task.

Unlike the Max Utility heuristic that solely maximizes for utility, the Max Utility-per-time (***Max UPT***) heuristic picks the task/machine/P-state choice that maximizes the ratio: "utility earned / execution time." In an oversubscribed environment, it is important to consider how much utility is earned per execution time used.

The Max Utility-per-Energy (***Max UPE***) heuristic focuses on reducing energy along with maximizing utility earned. It picks the allocation choice that maximizes the ratio: "utility earned / energy consumed by allocation." In an energy-constrained environment, this heuristic helps to rank allocation choices by considering both the worth of a task and its energy consumption.

For comparison purposes, we implement a ***Random*** heuristic that randomly assigns a mappable task to an available machine in a random P-state (among the allocation choices that passed through the energy filter and are valid).

IV. Simulation Setup

A. Overview

We use simulations to study our problem because we want to test and analyze the performance of the heuristics in a variety of environmental conditions. We simulate the arrival and mapping of tasks over a duration of 26 hours, with the first two hours used to bring the system up to steady-state operation. We collect our results (e.g., total utility earned, energy consumed) only from the start of the third hour to the end of the 26$^{\text{th}}$ hour (total of 24 hours) to avoid the scenario where the machines start with empty queues. All the simulation experiments were run on the ISTeC Cray System at Colorado State University [17]. Each of the trials represents a new workload of tasks (with different utility functions, task types, and arrival times), and a different computing environment by using new values for the entries in the ETC and APC matrices (but without changing the number of machines). All of the parameters used in our simulations are set to closely match the expectations for future environments of interest to the ESSC.

B. Workload Generation

A utility function for each task in a workload is given, and each task has a maximum utility value (depending on its priority level) that starts at one of 8, 4, 2, or 1. These values are based on the plans of the ESSC, but for other environments, different number of values and different values of maximum utility may be used. A method for generating utility functions can be found in [7]. Each task belongs to one among four urgency levels and 20 utility classes.

In our simulation environment, approximately 32,000 tasks arrive during the duration of a day, and each belongs to one of 100 task types. Out of the 100 task types, 83 are general-purpose and 17 are special-purpose. Each task type has approximately the same number of tasks in it. We generate the arrival patterns to closely match patterns of interest to ESSC [7]. The general-purpose tasks arrive in a sinusoidal pattern and special-purpose tasks follow a bursty arrival pattern.

C. Execution Time and Power Modeling

The compute system that we model has 13 machine types (four special-purpose) consisting of a total of 100 machines. The four special-purpose machine types have 2, 2, 3, and 3 machines in them. The remaining 90 machines are general-purpose and are split into the remaining nine machine types as follows: 5, 5, 5, 10, 10, 10, 10, 15, and 20. The machines of a special-purpose machine type run a subset of special-purpose task types approximately ten times faster on average than the general-purpose machines can run them (as discussed below). The special-purpose machines do not have the ability to run tasks of other task types. In our environment, three to five special-purpose task types are special for each special-purpose machine type.

We assume that heuristics can make use of three P-states in all machines: the highest power P-state (P-state 0), lowest power P-state, and an intermediate P-state. We use techniques from the Coefficient of Variation (*COV*) method [22] to generate the entries of the ETC and APC matrices in the highest power P-state. The mean value of execution time on the general-purpose and the special-purpose machine types is set to ten minutes and one minute, respectively. The mean dynamic power was set to 133 watts. To generate the dynamic power values for the intermediate P-state and the lowest power P-state, we scale the dynamic power to 75% and 50%, respectively, of the highest power P-state. The execution times for these P-states are also generated by scaling the execution time at the highest power P-state by sampling a gamma distribution with a mean value that is approximately $1/\sqrt{(\%\ scaled\ in\ power)}$. For example, the lowest power P-state's execution time will be scaled by a value sampled from a gamma distribution that has a mean approximately equal to $1/\sqrt{0.5}$. The execution time of any task is guaranteed to be the shortest in the highest power P-state, but the most energy-efficient P-state can vary across tasks. Such a model

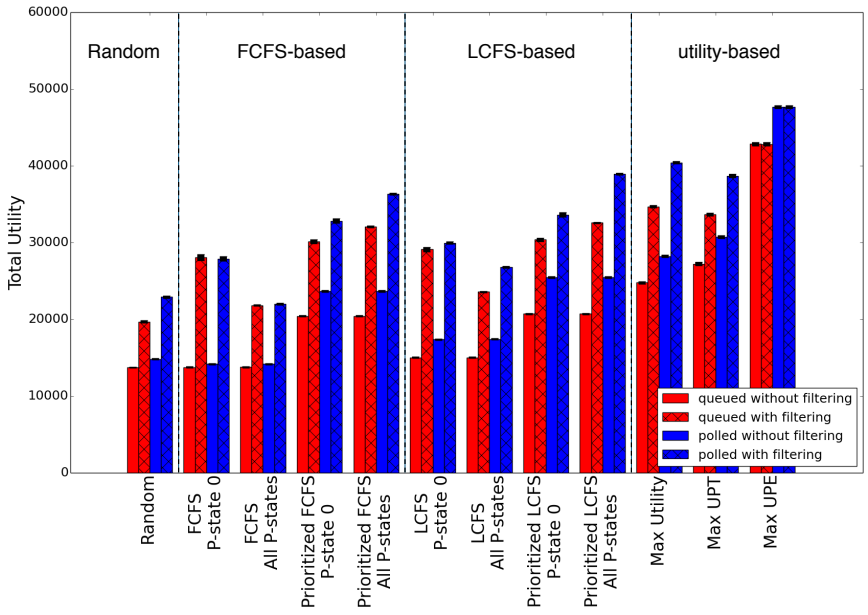


Fig. 4. Total utility earned by the heuristics in the queued and the polled task management environments. For each of those cases, the hatched bars show the performance when the adaptive energy filter technique was used. For almost all the heuristics, the polled environment and the energy filtering technique help to significantly improve performance. The results are averaged over 48 simulation trials and 95% confidence intervals are computed.

approximates reality where the impact on execution time and energy consumption by switching P-states depends on, among other factors, the CPU-intensity/memory-intensity of the task and static power of the system.

D. Obtaining an Energy Constraint

To obtain an energy constraint for use in our simulation studies, we applied a heuristic from our previous work that did not have an energy constraint. In particular, we used Max UPT because it maximized utility in [7], where energy was not a constraint. We used the same workload and environment setup as mentioned in Sections IV-B and IV-C and recorded how much energy was consumed in the day when using Max UPT to make scheduling decisions without any constraint on energy. We determined this energy value for the 48 simulation trials and computed its average. To make our problem challenging, we set the energy constraint for our environment to be 70% of this average value. As a result, for our simulations, we used an energy constraint value of 1.11 GJ per day.

V. Results

All results shown in this section display the average over 48 simulation trials with 95% confidence interval bars. The execution time of the heuristics and other mapping operations is on the order of 10^{-4} seconds per mapping event with the maximum time being ≈ 3 milliseconds. Therefore, in all of our cases, the mapping events were triggered approximately every 60 seconds. All of the heuristics used a dropping threshold of 0.5 units of utility to tolerate the oversubscription, as it gave the best performance without removing tasks of any one priority level completely. The dropping operation helped reduce the execution time of heuristics in the oversubscribed environment. When selecting a dropping threshold, one must consider the level of oversubscription of the environment in addition to the utility values of tasks.

Figure 4 shows the utility earned by the various heuristics in both queued and polled environments as well as with and without the energy filtering technique. The dashed vertical lines separate heuristic types into the following: Random, FCFS-based heuristics, LCFS-based heuristics, and the heuristics that use utility function information. We observe that, in general, the performance of the heuristics on the right is better than those on the left. The Prioritized FCFS heuristic performs better than the FCFS heuristic because it focuses on executing the high priority tasks first, and as a result, earns more utility per task. This is useful in an oversubscribed system as the heuristic is required to pick the better tasks to run. We see a similar trend when comparing Prioritized LCFS with LCFS. The LCFS-based heuristics usually outperform their FCFS counterparts because the LCFS-based heuristics focus on serving more recently arrived tasks. These recently arrived tasks tend to have utility values that have decayed less than tasks that arrived in the system earlier. The best performance is obtained by the heuristics that use utility function information to determine how much utility a task will earn. This is important because even though a task may have the highest priority it may also decay very fast. By the time such a task completes it may earn less utility than a task that may have a lower priority but does not decay much or does so slowly. The Max UPE heuristic significantly outperforms the other heuristics because it proactively reduces energy consumption and uses the saved energy to execute more

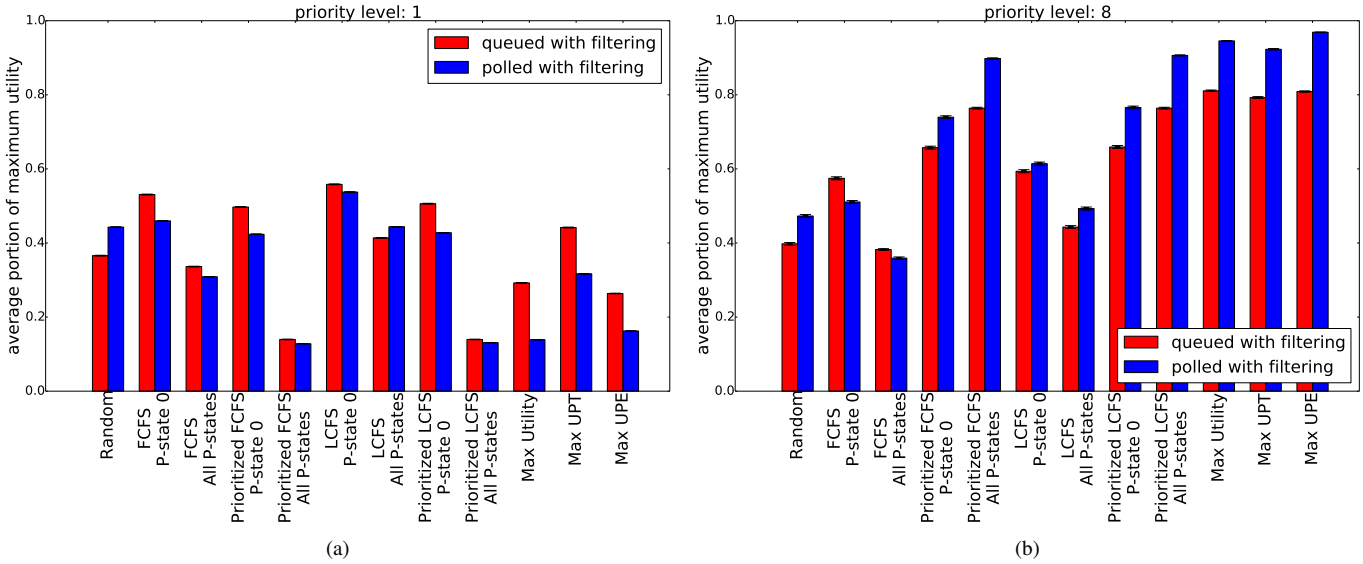


Fig. 5. Portion of maximum utility earned by the different heuristics when using the filtering technique for tasks that belong to (a) priority level 1, and (b) priority level 8.

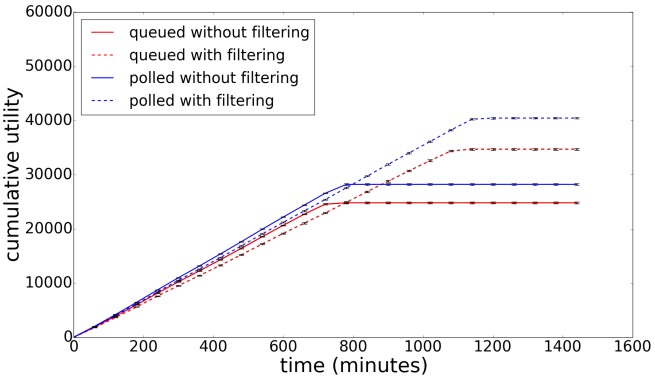


Fig. 6. Cumulative utility earned by the Max Utility heuristic as time progresses highlighting the benefit of the filtering technique to save energy and use it to earn utility in the later parts of the day. The results are averaged over 48 simulation trials and 95% confidence intervals are computed.

tasks and earn more utility.

We observe that the polled environment significantly improves the performance for most of the heuristics, compared to the queued environment. This happens because the polled environment does not lock down a task into the pending slot, as happens in the queued environment, and therefore, is better able to more quickly serve any high utility earning tasks that arrive compared to the queued environment. It is worth noting that the polled environment has more idle time than the queued environment. This is because, in the polled environment, whenever a machine finishes execution of a task, it has to wait (idle) until the trigger of the next mapping event for another task to be assigned to it. Therefore, traditional metrics for performance such as utilization would incorrectly identify the polled environment to be performing worse, but it is those small amounts of idling distributed throughout the day that improves the ability of the system to quickly service any high utility tasks that may arrive. Figures 5a and 5b show on average the portion of maximum utility that was earned from tasks that belonged to priority level 1 and priority level 8, respectively. We see that for tasks that belong to the higher priority level, the polled environment does a better job of earning higher utility as compared to the queued environment. The figures also highlight the effectiveness of the priority-based and utility function-aware heuristics in attempting to earn a larger portion of the maximum utility from tasks that have a higher priority level versus those that have a priority level of 1. The trend with a priority level of 4 is very similar to 8 and the trend with a priority level of 2 is in-between the trends in the priority level 8 and 1 charts. The results shown in these figures are for the cases using the filtering technique but the no filtering cases also show similar overall trends.

It is worth noting that the time between subsequent mapping events versus the average task execution time is an important factor in determining which type of task management environment would be the best. For example, if the heuristic execution times were to be very long, leading to longer times between mapping events, then it is likely that the queued environment will perform better than the polled environment because the polled environment will starve the machines whereas in the queued environment the machines can at least allocate tasks based on the queues.

Figure 4 also shows the significant benefit provided by our novel adaptive energy filtering technique for almost all of the heuristics in either of the task management environments. The filtering technique allows the heuristics to save energy for the later part of the day that can then be used to execute and earn utility from any high utility tasks that arrive at those

times. A trace chart showing the utility earned by the Max Utility heuristic as time progresses (Figure 6) highlights the benefit of using the filtering technique. In our ESSC inspired environment, high-utility tasks arrive throughout the day and therefore it is typically beneficial to have some energy left to spend towards the end of the day. Similar trends are observed for the other heuristics as well and our filtering technique helps all of the heuristics in a similar manner by adapting to their energy consumption rate. When not using the filtering technique, all of the heuristics except Max UPE hit the energy constraint before the end of the day, and therefore, the heuristics (except Max UPE) benefit from using the energy filter. As Max UPE already considers energy consumption, it does not benefit from the energy filter. Figure 6 shows how the polled environment always has a slightly higher slope than the queued environment. This is because the polled environment is consistently better able to serve the high priority tasks that arrive than the queued environment is able to serve.

For the FCFS-based and the LCFS-based heuristics, whether only P-state 0 (fastest P-state) is permitted or if all the other P-states are allowed only matters in the filtering cases, because in the no filtering case, the heuristic always makes assignments in P-state 0. When using filtering with the FCFS and LCFS heuristics, Figure 4 shows that considering only P-state 0 has a larger improvement in performance compared to the performance improvement when considering all P-states. This happens because of two reasons. The first reason is that by considering other P-states in all of the machines, even though more energy is saved, the execution time also increases leading to only a limited increase in total utility. It is worth noting that the first choice that satisfies the filter is chosen as opposed to the best energy choice. The second reason is that by only considering P-state 0, the FCFS and LCFS heuristics get the benefit of searching for another machine that satisfies the energy filter (but still only uses the fastest P-state). This search for a low energy machine also leads to minimizing the execution time in our environment (as energy = power $\times$ time). This results in more utility being earned from task completions. These benefits are not substantial when using the Prioritized FCFS and Prioritized LCFS heuristics, as they first consider the highest priority tasks that are not necessarily the earliest arriving or latest arriving, and the benefit from considering only P-state 0 is less than the benefit of the energy savings provided by considering other P-states, allowing them to earn more utility during the later part of the day.

VI. Related Work

In [7], the concept of utility functions to describe a task's time-varying importance is introduced. Energy is not considered at all in that paper. In this work, we are concerned with maximizing utility while obeying an energy constraint.

Energy-aware scheduling has been extensively studied. In [23], the authors design techniques to schedule a bag-of-tasks to a heterogeneous computing system with the goal of minimizing energy consumption under a throughput constraint. In [24], the authors formulate a bi-objective resource allocation problem to analyze the trade-offs between makespan and energy consumption. Our work differs from these as we maximize utility earned under an energy constraint.

In [25], a set of dynamically arriving tasks with individual deadlines are allocated to machines within a cluster environment with the goal of conserving energy. Specifically, the authors try to optimize the energy consumption while meeting the constraint of completing all tasks by their deadlines. Our environment tries to maximize the total utility earned while operating under an energy constraint. As a result, we design an adaptive energy filtering technique. Additionally, [25] models an undersubscribed system, while our work focuses on highly oversubscribed environments.

The research in [26] attempts to maximize a mathematical model of Quality of Service under an energy constraint by using DVFS to take up slack time in an undersubscribed system, which is very different from our oversubscribed environment.

A dynamic resource allocation problem in a heterogeneous energy-constrained environment is studied in [27]. Tasks in this system contain individual deadlines, and the goal is to complete as many tasks by their individual deadlines as possible within an energy constraint. This is a different problem from our work as we are trying to maximize the utility earned (based on each task's completion time) and not the number of tasks that meet their hard deadlines. The concept of an energy filter is used in [27], and we build on that for a more complex filter that automatically adjusts its level of filtering.

VII. Conclusion

In this paper, we study the problem of maximizing utility in an oversubscribed heterogeneous computing environment while satisfying an energy constraint. We examine and compare the performance of multiple heuristics in different task management environments. Our results indicate that the polled environment provides significant benefit over the queued environment in a system like ours because it has the ability to quickly service newly-arrived tasks with high utility. The queued environment with its pending slot and virtual queues may be more useful in an environment where the time between mapping events is longer than the average task execution time. We design, implement, and analyze multiple versions of the First Come First Served heuristic that are commonly used in many real-world schedulers. We compare the performance of these heuristics with Last Come First Served heuristics and other smart heuristics and demonstrate the strength of these smart heuristics. We also design a novel energy filtering technique that can be used with any of the heuristics and can be readily deployed in any environment without the need for any off-line parameter tuning. The adaptive energy filtering technique improves the performance of almost all the heuristics by allowing the heuristics to distribute their consumption of the budgeted energy and earn more utility.

Possible directions for future research include: (1) making the resource allocation techniques robust to uncertainties such as stochastic task execution times, machine failures, etc., (2)

experimenting with other types of task management environments, and (3) considering parallel and dependent tasks scheduling.

ACKNOWLEDGMENTS

This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, supported by the Extreme Scale Systems Center at ORNL, which is supported by the Department of Defense. Additional support was provided by a National Science Foundation Graduate Research Fellowship, and by NSF Grants CCF-1302693 and CCF-1252500. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This research also used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386. The authors thank Mark Oxley and Daniel Dauwe for their valuable comments.

REFERENCES

[1] America's data centers consuming and wasting growing amounts of energy. [Online]. Available: http://www.nrdc.org/energy/data-center-efficiency-assessment.asp

[2] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, vol. 53, no. 3, pp. 50–58, Mar. 2010.

[3] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers," in *Power Aware Computing*, ser. Series in Computer Science, R. Graybill and R. Melhem, Eds. Springer US, 2002.

[4] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, and S. Poole, "Energy-efficient application-aware online provisioning for virtualized clouds and data centers," in *International Green Computing Conference*, Aug 2010, pp. 31–45.

[5] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, "Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system," *Sustainable Computing: Informatics and Systems*, vol. 5, pp. 14–30, Mar. 2015.

[6] Advanced configuration and power interface specification. [Online]. Available: http://www.acpi.info/spec.htm

[7] B. Khemka, R. Friese, L. D. Briceño, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Transactions on Computers*, accepted 2014, to appear.

[8] "Introducing Titan," Jun 2014. [Online]. Available: https://www.olcf.ornl.gov/titan/

[9] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole, "An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environments," in *22nd Heterogeneity in Computing Workshop (HCW 2013), in the proceedings of the IPDPS 2013 Workshops & PhD Forum (IPDPSW)*, May 2013, pp. 19–30.

[10] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," in *10th Heterogeneous Computing Workshop (HCW 2001), in the proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001, pp. 875–882.

[11] M. K. Dhodhi, I. Ahmad, and A. Yatama, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1338–1361, Sep. 2002.

[12] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, vol. 26, no. 6, pp. 78–86, June 1993.

[13] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, July 1998.

[14] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, vol. 26, no. 6, pp. 18–27, June 1993.

[15] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," in *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86–97.

[16] D. Xu, K. Nahrstedt, and D. Wichadakul, "QoS and contention-aware multi-resource reservation," *Cluster Computing*, vol. 4, no. 2, pp. 95–107, Apr. 2001.

[17] Colorado State University ISTeC Cray High Performance Computing Systems. [Online]. Available: http://istec.colostate.edu/activities/cray

[18] M. R. Gary and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[19] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–121, Nov. 1999.

[20] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, June 2001.

[21] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

[22] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and Sa. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering, Special Issue, Invited*, vol. 3, no. 3, pp. 195–207, Nov. 2000.

[23] J.-F. Pineau, Y. Robert, and F. Vivien, "Energy-aware scheduling of bag-of-tasks applications on masterworker platforms," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 145–157, Feb. 2011.

[24] R. Friese, T. Brinks, C. Oliver, A. A. Maciejewski, H. J. Siegel, and S. Pasricha, "A machine-by-machine analysis of a bi-objective resource allocation problems," in *The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2013)*, July 2013, pp. 3–9.

[25] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *IEEE/ACM International Symposium of Cluster Computing and the Grid (CCGrid 2007)*, 2007, pp. 541–548.

[26] H. Yu, B. Veeravalli, and Y. Ha, "Dynamic scheduling of imprecise-computation tasks in maximizing QoS under energy constraints for embedded systems," in *Asia and South Pacific Design Automation Conference (ASPDAC 2008)*, Mar. 2008, pp. 452–455.

[27] B. D. Young, J. Apodaca, L. D. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou, "Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environments," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 326–347, Feb. 2013.

GetLB++: Improving Transaction Load Balancing on the Electronic Funds Transfer Landscape

Felipe Rabuske[1], Cristiano André da Costa[1], Rodrigo da Rosa Righi[1], Gustavo Rostirolla[1], Antonio Alberti[2], Anselm Busse[3] and Hans-Ulrich Heiss[3]

[1]Applied Computing Graduate Program - Univ. Vale do Rio dos Sinos - São Leopoldo, RS, Brazil

[2]Instituto Nacional de Telecomunicações - INATEL - Santa Rita do Sapucaí, MG, Brazil

[3]Technical University Berlin - TU Berlin - Sekretariat EN 6 - Einsteinufer 17, D-10587 Berlin

Abstract— *The continuous growth of electronic payment methods in business transactions is a reality that boosts e-commerce, being present more and more in our daily lives. Considering this situation, we proposed in a previous work a model denoted GetLB, which contains a scheduler that provides good results when compared to the traditional dispatching approach – the Round-Robin. Although offering a good distribution of transactions to processing (PMs), GetLB's scheduling routine is time consuming since each input is always analyzed against each target PM. Thus, in this paper we are proposing GetLB++ – a GetLB improvement that covers scheduling computation efficiency by bursting a transaction to a specific PM in accordance with a updated in-memory decreasing-sorted list of PM capacities. The results using an Amazon EC2 cluster instance showed a higher scheduling speed on GetLB++ in comparison with the standard GetLB, presenting gains about 20% on the makespan time. Besides scalability on EFT systems, GetLB++'s contributions are not limited to the context of transactional systems, but can also be extended for load balancing in e-commerce systems, cloud computing, and parallel programming.*

Keywords: Eletronic Funds Transfer, Scheduling, Algorithm

1. Introduction

Electronic payment methods, such as debit and credit cards, are being adopted by the society as the mainstream payment method for business transactions [1]. The benefits offered by EFT (Electronic Funds Transfer) range from a higher commodity for the buyers to a greater security for commercial institutions [2]. Usually, the dispatcher that receives transactions in the EFT company uses the Round-Robin (RR) algorithm to distribute them to processing machines, or PMs [3], [4]. RR algorithm consists of dispatching the new tasks in a circular fashion amongst the PMs, guaranteeing that the tasks are distributed uniformly between the processing units [5]. RR can be seen as a very fast strategy, with complexity $O(1)$, presenting an optimal load balancing for homogeneous systems [6]: when both consumers (in our case, electronic transactions) and resources (in our case, PMs) have the same configuration, performance is kept unchanged over time. Nevertheless, this scenario is not common in EFT systems, because of each kind of transaction has different computational needs [2].

Regarding the aforementioned scope, we developed a model named GetLB, which proposes a framework for scheduling transactions on an EFT company [7]. Periodically, GetLB's scheduling algorithm takes into account several characteristics of the transactions, such as the number of CPU instructions and memory consumption, as well as PMs data, to distribute the tasks. According to [7] GetLB obtained good results when distributing the workload amongst a dozen of nodes on homogeneous and heterogeneous clusters. The processing time of the GetLB's scheduling algorithm was about seven times greater than the RR routine. Additionally, this time tends to be bigger as the number of PM nodes of a cluster increases as well, facing a scalability problem. This happens for two reasons: (i) PMs periodically update their data to the dispatcher; (ii) at each transaction input, GetLB recalculates the workload that would be added on each PM of the cluster before choosing the one that will receive a new transaction.

Clearly, there is a gap on GetLB that can be explored in terms of expanding the model's scalability. In this way, we have developed GetLB++ — an enhanced and more flexible version of the standard GetLB, now focusing on improving scheduling routine (*i.e.*, the calculus involved in the scheduling procedure) but maintaining or yet improving the quality of the transactions-PMs assignment. GetLB++ can be used for processing different types of tasks, not being restricted to EFT scenarios. We developed a prototype that covers both GetLB and GetLB++ algorithms, besides an implementation of the RR. The prototype was evaluated with different rates for transactions arrival, distinct Amazon EC2 cluster instances and various input workloads. The results were encouraging, where GetLB++ obtained a better scheduling time and quality, besides presenting a larger scalability when compared to GetLB and RR.

The remainder of this article will first introduce the fundamental concepts in Section 2, presenting the main ideas of GetLB. Section 3 discusses about the related studies, giving the open issues in the EFT area. Section 4 describes the GetLB++ model in details, while Section 5 covers its

prototype. Section 6 brings the evaluation methodology to analyze GetLB, GetLB++, and RR. Section 7, in turn, discusses the results obtained from the experiments. Finally, the conclusion is written in Section 8, addressing scientific contributions and future work.

2. Background

This section presents the functioning of GetLB [7], offering the basis to understand the advances in the newer version. Electronic funds transfer transactions have different processing needs: CPU time, database access, network, and access to external systems. Thinking about these peculiarities, GetLB was developed to deliver a better load balance for transactions to PMs, so enabling benefits both to the users and provider company administrators. Figure 1 depicts the GetLB's architecture. GetLB was structured with the following design decisions in mind: (i) the scheduling heuristic algorithm runs in the dispatcher module and must work with up to date information regarding the PMs; (ii) the heuristic scheduling must combine relevant data in order to compose the notion of load; (iii) PMs must be capable to notify the switch; (iv) the framework must deal with heterogeneous resources at both communication and computing levels.

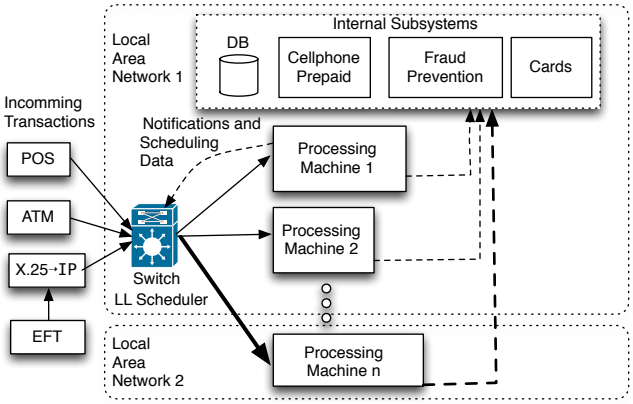


Fig. 1: GetLB architecture, emphasizing network decoupling and "PMs→switch" cooperative interaction besides the traditional one for transaction dispatching in opposite direction.

Regarding the scheduling activities, the dispatcher has an array that contains information about all PMs. Processing machines periodically report updates to the dispatcher, impacting on updating its array afterwards. The dispatcher performs all scheduling calculus with in-memory data, where the updating period informs how recent is PMs data regarding CPU, memory, and network. Thus, we previously developed a scheduling heuristic called LL (Load Level), which considers transactions and PMs are heterogeneous and, PMs as a part of a dynamic environment. LL can be viewed as a decision function $LL(i,j)$ where i means a specific type of transaction, while j denotes a candidate target PM for receiving transaction i. For each new transaction i, the switch will calculate n equations $LL(i,j)$, where n means the number of processing machines. Therefore, the lowest result will inform the target that will receive a transaction. $LL(i,j)$ can be obtained by computing Equation 1.

$$LL(i,j) = Recv(i,j) + Proc(i,j)\,, \tag{1}$$

$$Recv(i,j) = bytes(i) \times transfer(j)\,, \tag{2}$$

$$Proc(i,j) = transaction(i,j) + \sum_{z=0}^{m-1} transaction(z,j)\,, \tag{3}$$

$$\begin{aligned} transaction(i,j) &= \frac{instructions(i)}{clock(j) \times [1 - load(j)]} \\ &+ \frac{RAM(i) \times serviceRAM(j)}{freeRAM(j)} \\ &+ \frac{HD(i) \times serviceHD(j)}{freeHD(j)} \\ &+ sub(i,j)\,, \end{aligned} \tag{4}$$

$$sub(i,j) = \sum_{y=0}^{x-1} [2\, sub_a(y,j) + sub_c(y)] \times sub_r(i,y)\,. \tag{5}$$

Equation 1 is given by adding the estimated reception time ($Recv(i,j)$) and the estimated processing time ($Proc(i,j)$) of transaction i by machine j. In Equation 2, $bytes(i)$ means the size of the transaction i and $transfer(j)$ refers to the time necessary to transfer a single byte to the PM j. The Equation 3 calculates the total time that machine j needs to process transaction i, being divided in two sub-elements: (i) a prediction of computation time for transaction i on PM j; (ii) a prediction of all m transactions that have already been mapped to PM j previously and remain on its input queue. Static data (theoretical values for CPU, memory and access time to sub-systems) and the machine's dynamic data (considering CPU load, communication time and I/O requirements) are taken into consideration to calculate the estimated processing time, represented here in Equation 4.

Equation 5 captures the time spent by the sub-systems accessed by the machine j in order to process the transaction i. Each type of transaction i must access x subsystems. Thus, $sub_a(y,j)$ considers the time spent by PM j for accessing the particular subsystem y through network interaction. This time is multiplied for 2 in order to consider a round-trip evaluation. The field $sub_c(y)$ refers to the service time of the subsystem y and $sub_r(i,y)$ represent the number of times that subsystem y is called for the complete computation of i. The main drawback of the GetLB's algorithm is explained as follows: considering empty PMs, a single task i is mapped to PM0 so we have a $LL(i,0)$ equal to x. For the next transactions that test j as destination, the previous mapped transaction i can impact much larger than x since the $load(j)$ in Equation 4 was updated. This feature has a strong impact on limited machines, since they will be set as overloaded faster.

3. Related Work

The most studied topic in electronic transactions systems considers the security of information [8], [9], [10], [11].

However, security is not the only important topic in the context of EFT systems, but we can contemplate the load balancing (and also scheduling and resource management) problem too. This addresses how fast a provider computes a set of transactions, impacting directly in the user experience. In this way, Sousa et al. [12], [3] present a stochastic model for performance evaluation and resource planning. The tests compared measures of disk and processor utilization on a real system against the values obtained through the utilization of the proposed evaluation model. Desnoyers et al. [13] developed a system called Modellus, which allows modeling the usage of data centers around the Internet, automatically. The service rates are typically variable, limiting queuing theory application to this problem.

Mcheick et al. [14] explain that distributed systems can suffer from degradation problems in terms of performance and scalability. The authors point out that static algorithms work fine when there are no variations in the workload; therefore, they are not indicated for EFT scenarios where workload is not know in advance. Righi et al. [7] proposed the GetLB model aiming at filling the aforementioned gap, presenting both a framework and a scheduler capable to handle heterogeneous workload in dynamic environments. Although the heuristic used by GetLB (named Load Level - LL) took around six times more processing time than Round-Robin. In this perspective, we envision an opportunity to address EFT scalability by improving our previous work with the GetLB++ proposal, which is described in the next section.

4. GetLB++: An Improved Model for EFT Transactions Processing

This section presents GetLB++, which provides an evolution of GetLB to improve quantity and speed of the EFT transactions scheduler. GetLB++ consists of two parts: (i) a task processing framework for distributed systems; (ii) a transaction scheduling algorithm. Our idea is to offer a scalable system when combining both aforementioned parts. In terms of architectural elements, as GetLB does, GetLB++ works with input transactions from EFT terminals, a dispatcher or switch that schedules them to end processing units, named as processing machines or PMs. GetLB++ was developed with the following design decisions in mind: (a) the communication from the dispatcher to PMs must be asynchronous to void network latency; (b) PMs must be able to notify the dispatcher of any event that has impact on scheduling decisions; (c) the framework must allow the usage of different load balancing algorithms for scheduling purposes; (d) the framework must be able to process other types of tasks, so not restricting it to electronic funds transfer scenarios.

The dispatcher uses only in-memory data to schedule a transaction to a PM. Therefore, the processing machines are in charge of both monitoring their hardware and updating this information to the dispatcher in accordance to two modes: periodical and critical. In the periodical mode, PMs have a parameter named *verification period* that is used to both check their own hardware status and to send this information to the dispatcher afterwards. In this way, the dispatcher operates locally with data that are update in accordance with the aforesaid parameter. In the critical mode, a verification period is also used, but here only to check the hardware status in the PM. The switch is only updated with the hardware information of a certain processing machine when there is a critical change in the PM context. Thus, considering two consecutive measurements, there is another parameter named *critical change* indicating the percentage to consider as critical a sudden modification in the hardware status, which may represent impact on scheduling procedures.

4.1 Framework Modeling

GetLB++ follows the same idea of architectural elements from the GetLB model, with transactions, EFT terminals, a dispatcher and PMs (see details in Figure 1). In this work we are generalizing the use of transactions by employing the term Task, since GetLB++ was modeled for being not restrictive to the EFT scenarios. The task interface specifies methods that return information about the workload, estimated size, and a list of external systems that are accessed during the task processing. In addition, an object of this type must also implements a method *process()* which actually performs the processing of the task. Through this encapsulation, the terminals can send different types of tasks to be processed by the GetLB++, since the components of the framework do not need to know the implementation of the task. Therefore, this allows the framework to be extended beyond the EFT scenarios.

The GetLB++ model defines that the switch should have the capacity to operate using different scheduling algorithms, where an implementation of the Scheduler interface accomplishes this objective. This interface specifies the *getNextPU()* method, which returns the machine that will receive a task. The scheduling algorithms that implement this interface have access to information related to the processing machines through the list of ProcessingUnits inside the dispatcher, including their task queues. Therefore, when dispatching a task, the switch calls the *getNextPU* method, passing as parameter the list of Processing Units and the task that will be processed, and the scheduler returns the most suitable Processing Unit to accommodate the task.

4.2 Scheduling Algorithm

Although GetLB++ accepts various scheduling algorithms, the framework presents a default scheduler that was completely redesigned in order to meet the following goals when compared to the original GetLB: (i) have a higher

scheduling velocity; (ii) be more scalable; (iii) maintain or improve the load balancing quality by using the same system metrics as GetLB. As depicted in Figure 2, GetLB always computes n times the $LL(i, j)$ function, where n refers to the number of PMs, so verifying the impact and the conclusion time of task i on PM j. Considering that the mean number of already mapped transactions on each PM is equal to m, GetLB schedules a transaction with $O(n.m)$ complexity. See Equations 1 up to 6 for details. On the other hand, as mentioned in Section 1, Round-Robin is not suitable for heterogeneous systems but offers a $O(1)$ complexity.

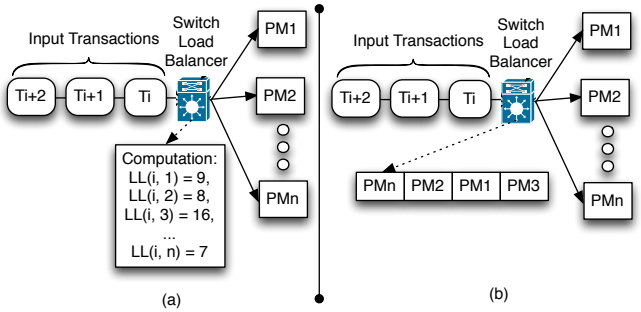


Fig. 2: (a) GetLB scheduling approach, in which each transaction is computed against each PM (Processing Machine); (b) GetLB++ list-scheduling approach.

The GetLB++'s scheduling algorithm goal is to explore scalability on scheduling calculus. Instead of on-the-fly computing the LL indexes and taking again the values for each PM at each incoming task, GetLB++ maintains a descending-ordered resource list which informs the PM with the higher processing capacity at a given moment (See Figure 2 (b)). This allows the system to determine the machine that should process the new task with almost no additional calculations, since the PM on the top of the list will always receive it. The aforesaid list is created when initializing the environment by using the *InitializingPUList* method of the scheduler.

The workload of a task is calculated by Equation 6, where i means the input task and j refers to the top machine in the aforementioned list. The terms *Recv(i,j)* and *transaction(i,j)* were further explained in Section II. The main difference from Equations 1 and 6 is that the last does not take into consideration the already mapped transactions to PM j but only the impact of task i on it. This modeling happens because GetLB++ does not try to test several PMs, but only the impact of a single transaction in a particular PM j. After calculating the workload of this task using *LL'(i,j)*, this value is added to the total load of machine j, that is repositioned in the Processing Unit list afterwards.

$$LL'(i, j) = Recv(i, j) + transaction(i, j) \quad (6)$$

To ensure that the total workload value on a machine reflects its most current state, the workload added by a new task using Equation 6 is stored in the task structure. When this task is completed, the processing machine sends an ending confirmation to the dispatcher. This last subtracts the load value related to the task from the Preprocessing using the *RemoveTask()* method and relocate again the PU in the list of machines.

Finally, we highlight that the GetLB++ scheduler denoted LL' calculates the Load Level of a task against a single PM, no matter the number of PMs and tasks residing in the cluster, which makes the algorithm highly scalable. Unlike this, the standard GetLB algorithm recalculates the Load Level for all machines in the cluster, always considering the target task and all previous mapped tasks on each machine.

5. GetLB++ Prototype

We developed a GetLB++ prototype using Java programming language and RMI middleware for communication substrate. SIGAR API[1] was used for real-time hardware monitoring on PMs. The prototype is divided into three components: (i) Task Launcher; (ii) Scheduler Machine; (iii) Processing Machine. Each component can be mapped to a different machine. The Task Launcher is in charge of reading an input file containing the tasks to be processed by the system, sending them to the Scheduler Machine after that. Each line of the input file corresponds to a task, which presents a type and a time, in milliseconds, informing how long the system should wait before process the next line.

The Scheduler Machine is an implementation of the dispatcher, being responsible for scheduling the beforehand received tasks. This component reads an XML file informing the scheduler type (today we are supporting GetLB++, GetLB and RR), the verification period, the critical change percentage and the updating mode.

6. Evaluation Methodology

This section describes the environment used for the tests, starting by presenting the considered tasks in Table 1. There are three possible types of tasks, A, B and C, referring to balance, prepaid telephony, and purchasing transactions, respectively. Transactions data were collected from a real EFT company provider called GetNet. We are evaluating the tasks against three scheduling algorithms: GetLB++, GetLB, and RR. In addition, four different input files were used by the Task Launcher element, each one containing five thousand tasks to be processed. These files were generated with the intention of testing the framework and scheduling behaviors under the combination of different situations: (i) homogeneous and heterogeneous tasks; (ii) pure sequential without delay between tasks (when sending a task to the dispatcher) and waiting times between them. The heterogeneous tasks were generated randomly, while the waiting time is a quadratic random function ranging from 0 to 100 milliseconds. This quadratic function is pertinent to emulate

[1] https://support.hyperic.com/display/SIGAR/

the real functioning of an EFT company, that receives more transactions not in the start/end of the day, but close to noon or 18:00 hs.

Table 1: Characteristics of the tasks used in the tests

Type	Properties		External Systems			
	Class	Size (n)	Card Sys.	Cryptography	Fraud Prev.	Rech. Sys.
A	Balance	500	Yes	Yes	Yes	No
B	Prepaid	1000	Yes	Yes	No	Yes
C	Telephony Purchasing	2000	Yes	Yes	Yes	No

In addition we are also varying the verification period, percentage of critical change and update mode. These parameters are only valid in the context of GetLB and GetLB++. Here, a percentage of 100% indicates that the last measure must be at least two times greater when compared to the previous one to trigger a PM-switch communication. The configurations used to test the hardware where: (1) Critical with 15% of change and verification period of 500 ms; (2) Periodical with 15% of change and period of 10 ms; (3) Critical with 100% of change and period of 100 ms and (4) Periodical with 1% of change and period of 1 ms

6.1 Infrastructure Testbed

All tests were executed in the infrastructure of the Amazon Elastic Computer Cloud[2], where clusters consist of machines running Windows Server 2012 R2. We are working with two clusters, composing homogeneous and heterogeneous infrastructures. Both were formed by ten machines, where the homogeneous setting includes only machines named t2.micro (as labeled by Amazon) and a communication latency of 40 milliseconds. t2.micro is composed of a single-core CPU with 2.5 GHz, 1 GiB of memory and 20 GiB of storage (SDD). The dispatcher is also a t2.micro machine, being used to run the Task Launcher too. The heterogeneous cluster, in turn, has different hardware settings varying from t2.micro to c3.large machines.

7. Results

This section presents the obtained results, starting with performance and scalability tests, observing the time on scheduling procedures. After that, we developed two sections to accommodate performance and quality of mapping results over homogeneous and heterogeneous clusters.

7.1 Scheduling Performance and Preliminary Scalability Tests

We have prepared basic scalability test involving the creation of a homogeneous cluster. We are varying the number of PMs from 2 to 12 to test the overhead on the RR, GetLB, and GetLB++ scheduling algorithms. Figure 3 depicts then this context in different curves. As expected, Round Robin does not suffer major impacts as the number of machine increases. GetLB got the worst scheduling besides presenting a low scalability when enlarging the resource infrastructure. Clearly, GetLB++ outperforms GetLB in the scheduling performance, but we must take care with the current observations in the following way: the performance ratio between GetLB and GetLB++ is slightly lower when working with 12 machines. Technically, 6.02 and 5.59 are the performance rations for these two algorithms when analyzing 2 and 12 PMs, respectively.

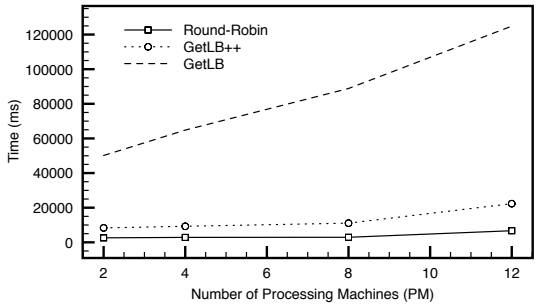


Fig. 3: Scheduling time when varying the number of PMs

7.2 Evaluation in Homogeneous Topology

This section presents the results when using a homogeneous cluster, but varying the type of the input workload. The graphics in this section show the load distribution among the PMs in the cluster. The values for these graphs were obtained by calculating the load level for each different type of tasks multiplied by the number of tasks of a particular type that were transferred to each machine.

7.2.1 Homogeneous Tasks

Figure 4 illustrates the load distribution of tasks among the PMs belonging to the homogeneous cluster. For this graph, it was used the test data of the first configuration of the update parameters, since there are no noticeable differences between this and the other settings. We observed a similar behavior on the three algorithms. Particularly, RR presents a completely uniform distribution of tasks to PMs because of considering, in this context, the duet resource and task as a homogeneous system.

7.2.2 Heterogeneous Tasks

Figure 5 shows a graph containing the load distribution of tasks in the homogeneous cluster for the three analyzed algorithms. Once again, the data used for the graph corresponds to the first configuration of the update parameters. It is noticeable that, while the Round Robin makes a homogeneous distribution, without considering that each type of task has special processing needs; the other two algorithms perform a fairer load distribution between the processing machines. Figure 5 shows that both GetLB and GetLB++

[2]http://aws.amazon.com/ec2

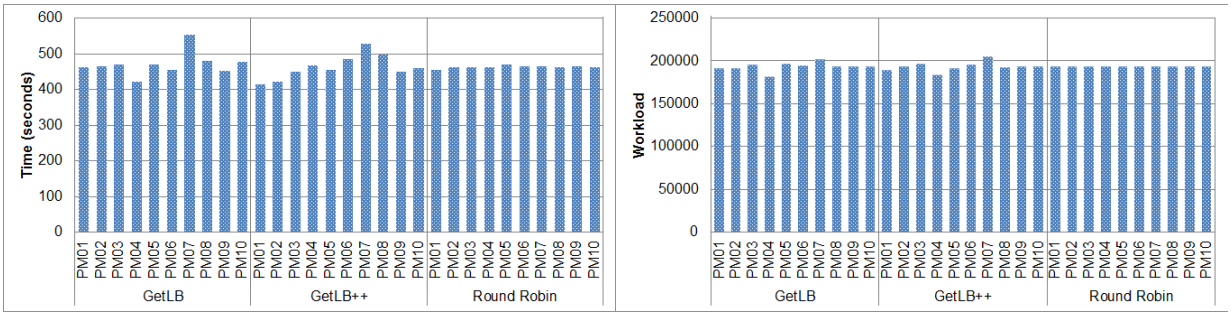


Fig. 4: Observing time and load distribution at each PM when running a homogeneous workload on the homogeneous cluster

Table 2: Processing times of a heterogeneous cluster with homogeneous workload. The column configuration follows the description in the Section 6.

Config.	Average Processing Time(ms) RR	GetLB	GetLB++
01	1640,73	1224.63	989.49
02	1640,73	1211.47	1005.40
03	1640,73	1229.62	971.20
04	1640,73	1356.21	1203.81

present a good load balancing since the homogeneous nodes (named in the graph as PMs) practically advance together in a horizontal line.

7.3 Evaluation in Heterogeneous Topology

This section presents the results obtained in the tests executed in the cluster composed by heterogeneous nodes. Here the results are presented in tables in order to show all four configurations values. Gains of 26% and 38% were obtained by GetLB and GetLB++ against the RR execution time over the configuration number two.

7.3.1 Homogeneous Tasks

Table 2 presents the results when handling homogeneous tasks. The standard deviations of these results were 32 ms for the average processing time. This expressive gain in performance was originated by the quality of the mapping provided by GetLB and GetLB++. We can observe that the higher the nodes capacity, the higher the load assigned to it. In addition, the use of heterogeneous resource turn more evident the differences between GetLB and GetLB++. Since GetLB++ does not try to evaluate a task against all PMs for scheduling purposes, but considers a decreasing sorted-list of previous mapped load to each PM, GetLB++ can provide both a better scheduling time and quality.

7.3.2 Heterogeneous Tasks

The scenario that puts together heterogeneous tasks and resources is responsible for the best results in favor of GetLB++. The obtained values are presented in Table 3, where the standard deviation of the average processing time is 41 ms. Gains up to 42% were obtained when comparing

Table 3: Processing times of a heterogeneous cluster with heterogeneous workload. The column configuration follows the description in the Section 6.

Config.	Average Processing Time(ms) RR	GetLB	GetLB++
01	3125.15	1870.12	1715.90
02	3125.15	1999.90	1790.62
03	3125.15	1989.59	1752.46
04	3125.15	1968.58	1706.07

GetLB++ and RR, and 11% in favor of GetLB++ when comparing its execution against the standard GetLB. The effectiveness of the RR scheduling calculus is totally ignored by the mapping provided by this algorithm. Analyzing the mapping of the load among the PMs we observed that the sequential method of RR leaves underloaded the more powerful resources.

7.4 Analysis and Discussion

As expected, RR was unbeatable when the term homogeneous is applicable for both resources and tasks. However, whenever a system component has a heterogeneous behavior (resources or task), Round Robin presented the worst processing time between the three analyzed algorithms. Homogeneous resources are responsible for a similar behavior between GetLB and GetLB++, presenting slightly better indexes for the last one. However, GetLB++ outperforms GetLB with heterogeneous resources both when considering uniform and non-uniform input workloads. The use of heterogeneous resources presented better results for GetLB++, which offers not only a faster scheduling calculus when compared to GetLB, but also a better mapping transactions-PMs for the following reasons:

8. Conclusion

According to the World Payments Report 2014[3], 70% of customers worldwide are expected to use mobile commerce in 2015 and more than 90% will likely be using online banking. So, this reality implies on performance challenges to EFT company providers, where the speed of an EFT

[3]https://www.worldpaymentsreport.com/

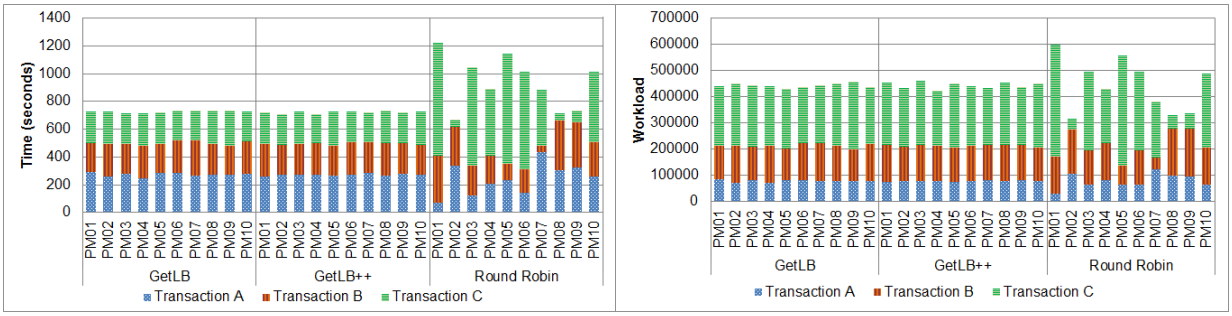


Fig. 5: Observing time and load distribution at each PM when running a heterogeneous workload on the homogeneous cluster

transaction can help on the client loyalty, while brings benefits to administrators who can handle more transactions per second, as well. In this context, this article addressed EFT performance through the GetLB++ proposal — an extension of GetLB especially focused on the scheduling procedure. GetLB++'s scientific contribution resides in the scheduling approach: contrary to GetLB, the number of PMs and the number of tasks mapped to each PM beforehand do not affect the scheduling performance. GetLB++ maintains a load-ranked decreasing-sorted list of resources and only takes the top position when arriving a new task. At each either hardware update at PMs perspective, conclusion of a task or dispatch of a task, this list is updated to reveal the most suitable quality scheduling.

According to the conducted tests, GetLB++ is in average 6.5 times faster than GetLB when performing the scheduling of a transaction, providing also in average a reduction in the total processing time by 11.78%. Furthermore, GetLB++ offers a more flexible framework, which allows the use of multiple scheduling algorithms, different PM-switch interaction approaches, and the processing of different kinds of tasks, not particularly EFT transactions. In this way, the contributions of GetLB++ are not limited to the context of transactional systems, but can also be extended for load balancing in e-commerce systems, cloud computing, and parallel programming. Heterogeneity at both resource and transaction levels were explored in the current paper to evaluate performance and scheduling quality. So, future work includes tests with resource dynamics and the use of notifications. In addition, we also intend to create a multi-cluster transactional environment considering different network latencies and bandwidths.

References

[1] D. R. Millen, C. Pinhanez, J. Kaye, S. C. S. Bianchi, and J. Vines, "Collaboration and social Computing in emerging financial services," in *Proceedings of the 18th ACM Conf. Companion on Computer Supported Cooperative Work & Social Comp.*, ser. CSCW'15 Companion. New York, NY, USA: ACM, 2015, pp. 309–312. [Online]. Available: http://doi.acm.org/10.1145/2685553.2685562

[2] B. Singh, R. Singh, and P. Singh Tanwar, "Electronic payment systems for online smart cards transaction system," *Int. Journal of Technology Research and Management*, vol. 1, no. 1, March 2014.

[3] C. Araujo, E. Sousa, P. Maciel, F. Chicout, and E. Andrade, "Performance modeling for evaluation and planning of electronic funds transfer systems with bursty arrival traffic," in *Intensive Applications and Services, 2009. INTENSIVE '09. First Int. Conf. on*. Valencia, Spain: IEEE, April 2009, pp. 65–70.

[4] I. Sbeity and M. Dbouk, "Software performance engineering using uml2san: Deadlock prediction of funds transfer," in *Computer Engineering Systems (ICCES), 2014 9th Int. Conf. on*, Dec 2014, pp. 318–323.

[5] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10922-014-9307-7

[6] J. Lewis, "The docline electronic funds transfer system (efts)," *Journal of Interlibrary Loan, Document Delivery & Electronic Reserve*, vol. 17, no. 3, pp. 75–83, 2007.

[7] R. Righi, C. A. da Costa, L. Gonzaga, Jr., K. Farias, A. L. Andrade, and L. Graebin, "Redesigning transaction load balancing on electronic funds transfer scenarios," in *Proceedings of the 29th Annual ACM Symposium on Applied Comp.*, ser. SAC '14. New York, NY, USA: ACM, 2014, pp. 775–777. [Online]. Available: http://doi.acm.org/10.1145/2554850.2555121

[8] R. Sastre, S. Bascon, and F. Herrero, "New electronic funds transfer services over ip," in *IEEE Electrotechnical Conf.*, 2006, pp. 733 –736.

[9] C. Vishik, A. Rajan, C. Ramming, D. Grawrock, and J. Walker, "Defining trust evidence: research directions," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '11. New York, NY, USA: ACM, 2011, pp. 66:1–66:1.

[10] E. Derman, Y. Gecici, and A. Salah, "Short term face recognition for automatic teller machine (atm) users," in *Electronics, Computer and Comp. (ICECCO), 2013 Int. Conf. on*, Nov 2013, pp. 111–114.

[11] R. Priya, V. Tamilselvi, and G. Rameshkumar, "A novel algorithm for secure internet banking with finger print recognition," in *Embedded Systems (ICES), 2014 Int. Conf. on*, July 2014, pp. 104–109.

[12] E. Sousa, P. Maciel, and C. Araujo, "Performability evaluation of eft systems using expolinomial stochastic models," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE Int. Conf. on*. IEEE, Oct 2009, pp. 3328–3333.

[13] P. Desnoyers, T. Wood, P. Shenoy, R. Singh, S. Patil, and H. Vin, "Modellus: Automated modeling of complex internet data center applications," *ACM Trans. Web*, vol. 6, no. 2, pp. 8:1–8:29, 2012.

[14] H. Mcheick, Z. Mohammed, and A. Lakiss, "Evaluation of load balance algorithms," in *Software Engineering Research, Management and Applications (SERA), 2011 9th Int. Conf. on*. Baltimore, MD, USA: IEEE, Aug 2011, pp. 104–109.

Scheduling Methods for OpenVX Programs on Heterogeneous Multi-core Systems

Tzu-Hsiang Lin, Cheng-Yen Lin, and Jenq-Kuen Lee
Department of Computer Science, National Tsing-Hua University, Taiwan
Email: thlin@pllab.cs.nthu.edu.tw, kennylin@pllab.cs.nthu.edu.tw, jklee@cs.nthu.edu.tw

Abstract— *Heterogeneous multi-core architectures are playing an important role in improving the overall performance of computer systems. To program such systems, OpenVX[1] promises to provide a standard programming framework for computer vision processing. OpenVX is with a graph-based execution model to describe the computation behavior and data flow relationship. Each computation node in the graph can be dispatched to a different target, such as multicore CPUs with C, OpenMP runtime, OpenCL on GPUs, or a dedicated hardware. Therefore, how to efficiently schedule all the computation nodes to those different targets opens up the optimization opportunities.*

In this paper, we propose a method to schedule OpenVX task graph by considering both memory locality and system throughput. The proposed two phase scheduling method first perform coarsen schemes to cluster nodes together, and then in the second phase a scheduling method is employed to schedule nodes into different targets. Preliminary experiments show that our scheme works well in scheduling OpenVX programs on heterogeneous environments.

Keywords: OpenVX, scheduling, coarsen, heterogeneous systems, computer vision

1. Introduction

In recent years, heterogeneous multi-core architectures are playing an important role in improving the overall performance of computer systems. Many chips are designed with different types of cores(such as GPU, DSP, hardware accelerator, etc.) integration. It challenges the programmer on programming, tasks scheduling and resource management. To ease the challenge of programming such systems, OpenVX[1] promises to provide a standard programming framework for computer vision processing. OpenVX is an open, royalty-free standard proposed by Khronos Group. It is an emerging programming framework for computer vision processing. As the OpenVX specification only defines the functional requirements, implementers can accelerate their implementation by applying a variety of optimization techniques for different optimization objectives. For example, different scheduling policies can be devised to meet the performance requirements or power constraints for mobile devices. Also, one can try to reduce the data transfer overhead by considering memory layout and utilizing the local memory. OpenVX is also with a graph-based execution model to describe the computation behavior and data flow relationship. Each computation node in the graph can be dispatched to a different target, such as multicore CPUs with C, OpenMP[2] runtime, OpenCL[3] on GPUs, or a dedicated hardware. Therefore, how to efficiently schedule all the computation nodes to those different targets opens up the optimization opportunities.

In this work, we address the scheduling issues of OpenVX task graph by considering both memory locality and system throughput. Our proposed two phase scheduling method first perform coarsen schemes to cluster nodes together, and then in the second phase, a scheduling method is employed to schedule nodes into different targets. In our work, we reference the work in [4] for static task-scheduling algorithms classification and the notion of graph attribute definitions such as upward and downward ranking, comparison metrics and the concept of HEFT, and CPOP algorithm. In addition, the task clustering problem is an NP-hard problem and is shown in [5]. Task time estimation model and load blancing machanism are introduced in [6]. The program features with profiling can be seen in [7]. Moreover, many different optimization strategies can be found in the work [8]. Preliminary experiments show that our scheme works well in scheduling OpenVX programs on heterogeneous environments.

The remainder of this paper is organized as follows. Section 2 overviews the OpenVX programming. Section 3 presents our scheduling methods. Next, Section 4 presents the preliminary experimental results and Section 5 concludes this paper.

2. Background with OpenVX Programming

For the sake of completeness, we introduce OpenVX programming in this section. OpenVX is an open standard programming framework for accelerating portable computer vision applications on different computation targets. The optimized implementations of OpenVX framework are provided by hardware vendors, while software developers will find a unified abstraction interface to utilize accelerated implementation either in software or hardware. With the APIs, one hopes both functional and performance portabilities across diverse devices and platforms can be enable for

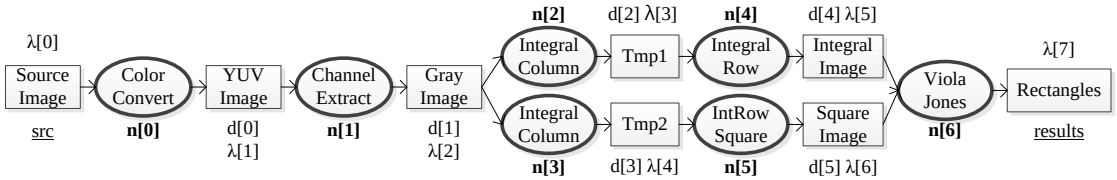


Fig. 1: An OpenVX graph of Haar face detection application

OpenVX applications.

In OpenVX framework, it uses the graph execution model to describe the computation behavior and data flow relationship by composing the computation nodes and data objects. A computation node is an instance of a computer vision kernel that combines with reference parameters, target affinity for execution, and associated graph. In other words, a computer vision function can be implemented as a kernel with parameter signatures, parent graph reference, and target options if the node has corresponding implementation on that targets. A target means an executor that can execute the OpenVX kernels, such as a multicore CPU with C or OpenMP runtime, general purpose computing GPUs with OpenCL or a dedicated hardware. It decouples the client code from a specific vendor's configurations or technologies and leaves the flexibility for vendor's implementations. Currently, the official sample implementation supports only C model, OpenMP, and OpenCL targets. All data objects in OpenVX framework are opaqueness. To avoid facing hardware-specific memory structures, the knowledge of memory location and data layout is controlled by the OpenVX framework implementation. The edges between computation nodes and data objects indicate the data dependency and data objects between nodes in a graph can be defined as virtual data objects to reveal optimization opportunities. In summary, the graph execution model is a directed acyclic graph (DAG) that determines the computation process of an OpenVX application.

The programming flow for an OpenVX application is summarized in Figure 2. There are six major phases in the programming flow. First, we need to create a context for the OpenVX framework to manage the reference counts on all objects. This is a necessary parameter for any manipulation with this framework. Using the context, we can construct the graph in the second phase. In this phase, we define needed data objects and connect them as a graph by distributing the computation nodes and the input and output reference of data objects for nodes. Third, the graph needs to be verified once as the structure of graph has not been verified or has been modified. Fourth, it starts the computation process by issuing the graph to OpenVX framework. We can execute this graph multiple times if needed. Fifth, it destructs the graph by calling the API to release nodes, data objects, and graph resources. Finally, it releases the OpenVX context and exits the program.

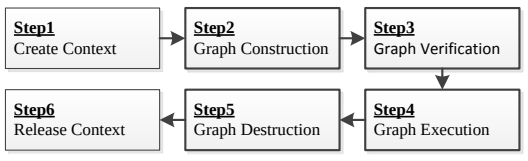


Fig. 2: OpenVX code flow

Figure 1 illustrates an OpenVX task graph of a Haar face detection application. The corresponding code segment is shown in Figure 3. We use oval shape with broad stroke to represent computation nodes while the rectangle shapes represents data objects. To focus on the processing flow of the OpenVX graph, we use function call to represent the routines about image reading (*ReadInputImage*), result drawing (*DrawResult*), and resource releasing(*ReleaseResource*).

Moreover, we also use the function call (*PublishCustomNodes*) to omit the codes manipulate user-defined nodes API and for the process about publishing our custom nodes (*vxIntegralColumn*, *vxIntegralRow*, *vxIntRowSquare* and *vxViolaJones*) to OpenVX framework. The input image(src) will be first converted into gray image(d[1]) by

```
vx_context c = vxCreateContext();
vx_graph g = vxCreateGraph(c);
vx_uint32 w=640, h=480, resSize=100;
PublishCustomNodes(&c);
vx_image src =
    vxCreateImage(c,w,h,VX_DF_IMAGE_RGB);
vx_array results =
    vxCreateArray(c,VX_TYPE_COORDINATES2D,resSize);
vx_image d = {
    vxCreateVirtualImage(g, w, h, VX_DF_IMAGE_IYUV);
    vxCreateVirtualImage(g, w, h, VX_DF_IMAGE_U8);
    vxCreateVirtualImage(g, w, h, VX_DF_IMAGE_U8);
    vxCreateVirtualImage(g, w, h, VX_DF_IMAGE_U8);
    vxCreateVirtualImage(g, w, h, VX_DF_IMAGE_U8);
    vxCreateVirtualImage(g, w, h, VX_DF_IMAGE_U8);
};
vx_node n[] = {
    vxColorConvertNode(g,src,d[0]);
    vxChannelExtractNode(g,d[0],VX_CHANNEL_Y,d[1]);
    vxIntegralColumn(g,d[1],d[2]);
    vxIntegralColumn(g,d[2],d[3]);
    vxIntegralRow(g,d[2],d[4]);
    vxIntRowSquare(g,d[3],d[5]);
    vxViolaJones(g,d[4],d[5],results);
};
if (vxVerifyGraph(g)==VX_SUCCESS) {
    ReadInputImage(w,h,&src);
    vxProcessGraph(g);
    DrawResult(src,results);
}
ReleaseResource(&c,&g,d,n,&src,&results);
```

Fig. 3: Code snippet of OpenVX Haar face detection

color-convert(**n[0]**) and channel-extract nodes(**n[1]**). Then this gray image is fed into two independent nodes to do the integral computation (node **n[2]** to node **n[5]**). After that, two integral images (d[4], d[5]) are merged in nodes(**n[6]**) doing cascade features computation using Viola Jones algorithm. Finally, the output rectangles can be obtained in an array (results). We can observe from the above process that some nodes on the paths (n[2],n[4] and path n[3],n[5]) can be parallel processing. In addition, virtual data objects(d[0] to [5]) will not be accessed by host, it can be optimized into a local memory residing on compute devices. For example, in OpenCL, we can make these memory objects as buffers on GPU devices, and read/write operations only occur when it is with host access or one needs to transfer data to different targets.

3. Coarsen-Scheduling algorithm

The Coarsen-Scheduling algorithm aims to improve the targets utilization and the memory locality in OpenVX framework. Computation nodes in the graph are dispatched to a specific target after processing by our two phase algorithm. The input of this algorithm are an OpenVX graph and a sequence of OpenVX target denoted as $P = \{p_0, p_1, \ldots, p_n\}$. An OpenVX graph is not only a weighted DAG as we mentioned above, but also a bipartite graph that can be denoted as $G = (N, D, E)$. By using this bipartite representation, we divide the graph into two node sets and one edge set. One of the node sets is the computation nodes set $n_0 \ldots n_i$, and the other is the data objects set $d_0 \ldots d_j$. Each edge in the edges set E represents the read or write relationship between a computation node in the computation nodes set N and a data object in the data objects set D. In this algorithm, we process an OpenVX graph in two phases: the node coarsen phase and the node scheduling phase. We now detail the process of each of the two phases.

3.1 Node coarsen algorithm

The node coarsen algorithm groups nodes on the OpenVX graph into clusters. With the input graph and target information, the output of this algorithm is a cluster set, C. Nodes in the same cluster will be forced to dispatch to the same target in the scheduling phase, so we only select the nodes which can reduce data transfer time or largely shorten the computation time into the same cluster. The remaining nodes not clustered will be scheduled for the target in the scheduling phase. Because the structure of an OpenVX graph is fixed after the verification stage, we just need to coarse once for a new graph structure. This algorithm will execute in verification phase(*vxVerifyGraph(graph)*). Before introducing the details of this algorithm, we first give the auxiliary definitions for the clustering steps. The computation time $\boldsymbol{\mu(n, p_i)}$ is the time a node $n \in N$ spent on the OpenVX target p_i doing computation. The data transfer time $\boldsymbol{\lambda(d, u, v)}$ is the time to transfer data $d \in D$ between computation nodes $u \in N$ and $v \in N$, it will be non-zero if u and v are on different targets. To get the $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ value, we need to profile each computation node on each target that OpenVX framework provides at the first time with a fixed data size. Moreover, we assume a linear relationship for data size and computation time, data size and data transfer time. So we will use a quadratic function to evaluate $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ value. The $transfer(d)$ is the maximum time to transfer data $d \in D$ from one writer node $w \in N$ to multiple readers $R \subset N$ defined by

$$transfer(d) = \max_{r \in R}(\lambda(d, w, r)) \tag{1}$$

According to the graph formalism rules defined in the OpenVX specification, every data object $d \in D$ has only a single writer node $n \in N$. So the maximum time to transfer data d must include the output edge from a writer node w connecting to d and one of the edges as input edge connect from d to a reader node. Finally, the improved factor $improve(n)$ is the maximum difference for computation node $n \in N$ running on target p_i and p_j defined as

$$improve(n) = \frac{\max\limits_{p_i \in P} \mu(n, p_i)}{\min\limits_{p_j \in P} \mu(n, p_j)} \tag{2}$$

where p_j is the target that require longest computation time to execute computation node n.

Algorithm 1: NodeCoarsen(G, P)

Input: an OpenVX graph: $G = (N, D, E)$. an OpenVX target sequence: P
Output: nodes for targets C
1 Each computation node $n \in N$ assign to a target $p \in P$ that minimize $\mu(n, p)$;
2 Compute the critical path CP;
3 $clusterCP(CP, C)$;
4 $clusterG(G, C)$;

We now present our node coarsen algorithm in algorithm 1. Initially, we assign each computation node $n \in N$ to a target $p \in P$ that cost minimal computation time in line 1. Then, we compute the critical path in line 2 which will be used in the following cluster procedures. A critical path CP can be computed by traversing the task graph upward while every computation node $n \in N$ choose the lowest computation time target. Starting from the exit node n_{exit}, the upward rank of a node n is recursively defined by

$$rank(n) = \mu(n, p) + \max_{d \in write(n)}(transfer(d) + rank(MTR)) \tag{3}$$

where $write(n)$ is the set of immediate successors of n, that is the set of data objects that are written by computation node n. In addition, MTR is the reader node that cause maximum

$transfer(d)$ time. Due to the data input of source nodes and data output of exit nodes should back to host target, the rank value of exit node $rank(n_{exit})$ is $\mu(n_{exit}, p)$ plus the data transfer time of each output data:

$$rank(n_{exit}) = \mu(n_{exit}, p) + \sum_{out \in output(n_{exit})} \lambda(out, n_{exit}, host\_target) \quad (4)$$

Since rank values are recursively calculated by traversing upward, the largest rank value on the head node plus the data transfer time of each input data is the makespan of the graph. We then divide the following steps of this algorithm into two procedures: $clusterCP$ in line 3 to cluster nodes on critical path first and $clusterG$ in line 4 to cluster the other nodes not on the critical path into clusters.

Procedure clusterCP(CP, C)

Input: Critical path: CP, nodes for targets C
Output: nodes for targets C

1 **while** *there are unvisited data object on* CP **do**
2 Choose an unvisited data object $d \in CP$ with maximum $\lambda(d, w, r)$, where computation nodes $w \in CP$ and $r \in CP$;
3 **if** *w.target=r.target* **then**
4 **if** *isValidWindowSize(*$w.target$*, w)* ***and*** *isValidWindowSize(*$w.target$*, r)* **then**
5 $C(w.target) \leftarrow C(w.target) \cup w \cup r$;
6 **end**
7 **else**
8 $gain \leftarrow \lambda(d, w, r)$;
9 $changeW \leftarrow \mu(w, r.target) - \mu(w, w.target)$;
10 $changeR \leftarrow \mu(r, w.target) - \mu(r, r.target)$;
11 **if** *w.visited=false* **then** $changeW \leftarrow \infty$;
12 **if** *r.visited=false* **then** $changeR \leftarrow \infty$;
13 $cost \leftarrow min(changeW, changeR)$;
14 **if** *gain>cost* **then**
15 **if** *cost=changeW* **then** $w.target \leftarrow r.target$;
16 **else** $r.target \leftarrow w.target$;
17 **if** *isValidWindowSize(*$w.target$*, w)* ***and*** *isValidWindowSize(*$r.target$*, r)* **then**
18 $C(w.target) \leftarrow C(w.target) \cup w \cup r$;
19 Re-Compute the critical path CP;
20 **end**
21 **end**
22 **end**
23 Mark d, w, r as visited;
24 **end**

In the $clusterCP$ procedure, we aim to shorten the time spent on the critical path, so the makespan of the graph can be shorten. This is a loop process until all data objects on the critical path are visited, excluding input data objects of source nodes and output data objects of exit nodes. We first choose a data object d on the critical path causing longest transfer time at line 2. Then we know the writer node w and reader node r connect to this data object d because there is only a writer and a reader can match the connection with data object d and on the critical path conditions. At line 3 to line 6, we try to cluster reader r and writer w into same cluster because the μ value is minimal and transfer time on same target is minimal too. At line 8 to line 20, reader r and writer w are on different targets. We try to change reader or writer target to be the same target if gain earn more than cost. Critical path needs to be recomputed if target changes. At line 21, we mark the d, r and w as visited so the problem size will decrease. The $isValidWindowsSize$ procedure is a mechanism to limit the size of a cluster. We will discuss the impact of the evaluation method used in this procedure in the experiment section.

Procedure clusterG(G, C)

Input: OpenVX graph: G, nodes for targets C
Output: nodes for targets C

1 **while** *there are unvisited computation nodes in* G **do**
2 Choose an unvisited computation node n with maximum $improve(n)$;
3 $diff\_count \leftarrow 0$;
4 **foreach** *data object* d *adjacency to* n **do**
5 **foreach** *computation node* $adjn$ *adjacency to* d *with different read/write direction as* n **do**
6 **if** $adjn.visited$*=true* ***and*** $adjn.target \neq n.target$ **then** $diff\_count \leftarrow diff\_count + 1$;
7 **end**
8 **end**
9 **if** *diff_count=0* **then**
10 **if** *isValidWindowSize(*$n.target$*, n)* **then** $C(n.target) \leftarrow C(n.target) \cup n$;
11 **else**
12 Choose a data object d connect with visited computation node m which will cause maximum $\lambda(d, n, m)$;
13 $gain \leftarrow \lambda(d, n, m)$;
14 $cost \leftarrow \mu(n, m.target) - \mu(n, n.target)$;
15 **if** *gain>cost* ***and*** *isValidWindowSize(*$m.target$*, n)* **then**
16 $n.target \leftarrow m.target$
$C(n.target) \leftarrow C(n.target) \cup n$;
17 **end**
18 **end**
19 Mark n as visited;
20 **end**

After nodes on critical path are visited, the $clusterG$ procedure process the nodes not on the critical path. The aim

of this procedure is to reserve most-earn computation nodes first. We loop until every nodes on the graph are visited. In each round at line 2, we choose the node causing maximum $improve$ value. In other words, the nodes take precedence in this clustering process that has largest time difference on the fastest target and slowest target. At line 3 to line 8, we check computation nodes connect with the same data nodes that adjacency to n. If there are visited nodes, it is because they are on critical path and are marked as visited in the previous $clusterCP$ procedure. The $diff\_count$ will be non-zero if there exist visited nodes with different target. Therefore, at line 9 to line 11 is a simple case. We just try to make the node n into $n.target$ cluster. While a complex case at line 12 to line 18, we find the most expansive data transfer cost edge and try to evaluate the data transfer gain and computation cost of node n. If this trade-off is worth enough, then try to make the node n select target $m.target$ and add into cluster. Finally, computation node n is mark as visited so the loop will stop.

We use the OpenVX graph shown earlier in Figure 1 as an example to explain how our node coarsen algorithm works. To simplify the analysis process, we assume each computation node has only two targets $target\_a$ and $target\_b$. In addition, we assume $target\_a$ is the OpenVX host target. The data transfer time between $target\_a$ and $target\_b$ is symmetry(e.g. $\lambda(d, target\_a\ node, target\_b\ node) = \lambda(d, target\_b\ node, target\_a\ node)$) and would be zero if on the same target. The computation time μ for each node on $target\_a$ and $target\_b$ is expressed as a tuple(e.g. $(6, 2)$ for node n means $\mu(n, target\_a) = 6$ and $\mu(n, target\_b) = 2$) and λ is a non-zero value for each data object if data transfer of different targets is needed. Let λ value for $\lambda[0] \sim \lambda[7] = \{3, 3, 1, 2, 2, 3, 2, 5\}$ and μ value for $n[0] \sim n[6] = \{(6, 2), (4, 1), (6, 5), (6, 5), (4, 6), (6, 8), (15, 10)\}$. At first each computation node $n[0] \sim n[6]$is initially assigned to a target that minimizes $\mu(n, p)$, so we get the target selection of each node as $\{B, B, B, B, A, A, B\}$. We then compute the critical path, the exit node of this graph is $n[6]$. The rank of $n[6]$ is $rank(n[6]) = \mu(n[6], B) + \lambda(results, n[6], A) = 10+5 = 15$. Then we can recursively traverse upward and get the rank of each node is $\{3+2+3+32 = 40, 1+1+max(29, 30) = 32, 5+2+22 = 29, 5+2+23 = 30, 4+3+15 = 22, 6+2+15 = 23, 10+5 = 15\}$. The critical path is $\{n[0], n[1], n[3], n[5], n[6]\}$ and the makespan of this graph is 40.

In $clusterCP$ procedure, we pass the critical path into it. The first unvisited data object on the critical path causing maximum λ value is $d[0]$, because $\lambda(d[0], n[0], n[1]) = 3$ is maximum. Writer and reader node of $d[0]$ are $n[0]$ and $n[1]$ with same target choice B, so at line 4 to line 6 of $clusterCP$ procedure would try to cluster them into cluster. We assume the window size rule here is adjacency nodes in the same cluster can not more than two nodes. So this windows is valid and we can get $C(B) = \{n[0], n[1]\}$. Second loop time, we get $d[3]$ and $d[5]$ with same λ value. We choice $d[3]$ according to the rank non-increasing order. Writer $n[3]$ and reader $n[5]$ prefer different target, so at line 8 to line 22 will try to change them to be on same target. The gain is $\lambda(d[3], n[3], n[5]) = 2$ and cost is $min(6-5, 8-6) = 1$, gain is greater than cost and windows size for writer and reader in cluster A is valid. We get a new cluster $C(A) = \{n[3], n[5]\}$. For the last remaining nodes $d[5]$ and $d[1]$, because visited nodes appear in writer or reader nodes would bring ∞ cost and for $d[5]$ change reader cost is 5, gain is smaller than cost. Two clusters are obtained on critical path.

In $clusterG$ procedure, we cluster remaining nodes $n[2]$ and $n[4]$ on the graph. The $improve(n[2])$ is 1.2 and $improve(n[4])$ is 1.5, so we choose $n[4]$ as a unvisited computation node first. At line 3 to line 8, the $diff\_count$ will count 1 because $n[6]$ is visited and it is on a different target B versus $n[4]$ on target A. So at line 12 to line 18, we try to change the target running $n[4]$. The gain is $\lambda(d[4], n[4], n[6]) = \lambda[5] = 3$ and the cost is $\mu(n[4], B) - \mu(n[4], A) = 6 - 4 = 2$. Gain greater than cost so we will change $n[4]$ to select target B and get a cluster $C(B) = \{n[4], n[6]\}$. For node $n[2]$, the $diff\_count$ count is zero because n[1] and n[4] are on the same target(B) as $n[2]$ and $n[3]$ are read edge to $d[1]$, too. The algorithm finally gets $C(A) = \{n[3], n[5]\}$ and $C(B) = \{(n[0], n[1]), (n[4], n[6])\}$ in node coarsen algorithm.

3.2 Node scheduling algorithm

The node scheduling algorithm dispatches nodes according to non-increasing order of rank values. This algorithm will execute in graph processing phase(*vxProcessGraph(graph)*). After the node coarsen algorithm clusters some of the computation nodes into clusters, this runtime scheduling algorithm directly dispatch nodes with clusters and makes targets decision for non-clustered nodes by considering the runtime environment. We now present our node scheduling algorithm in Algorithm 2.

Initially, we compute the rank value as algorithm 1 at line 1 to line 3. Then we sort the rank value in non-increasing order and take the first one in each loop cycle. If the node in assign to cluster, we dispatch the node directly at line 6. On the contrary, we try to evaluate the gain and cost on prefer target and most free target. Node can be dispatched to preference target if prefer target is free at line 12. At line 14 to line 23, we try to change node n to run on most free target. A new rank get at line 16 always higher because of the μ cost. But if this cost is worth versus wait time of prefer target, we should change the target of n. Two decision at line 17 to line 18 and at line 20 to line 22 are either change the target of n and recompute rank or keep original target selection.

We use the result of previous node coarsen algorithm as sample input. Because the cluster relationship. Node

Algorithm 2: Scheduling(G, P, C)

Input: OpenVX graph: G, an OpenVX target sequence: P, nodes for targets C
Output: computation node n being dispatched to OpenVX target

```
Compute rank value for all nodes by traversing graph upward, starting from sink nodes;
Sort the nodes in the graph by non-increasing order of rank values;
while there are unscheduled nodes in the graph do
    Select the first node n;
    if node n ∈ C(p) then
        Dispatch all nodes in cluster to target p;
    else
        preferP ← n.target;
        rank_on_P ← n.rank;
        mostFreeP ←the least loading target in P;
        if preferP=mostFreeP then
            Dispatch the node n to target preferP;
        else
            Assign n.target to mostFreeP;
            Re-Compute rank value for all nodes;
            if rank_on_P − n.rank < wait_time then
                Dispatch the node n to target mostFreeP;
                Re-Sort the nodes in the graph by non-increasing order of rank values;
            else
                Assign n.target back to preferP;
                Dispatch the node n to target preferP;
                Re-Compute rank value for all nodes;
            end
        end
    end
end
```

$n[0], n[1], n[4]$ and $n[6]$ directly dispatch to target B. Node $n[3]$ and $n[5]$ directly dispatch to target A. Node $n[2]$ is not in cluster, the original rank of $n[2]$ is $\mu(n[2], B) + \mu(n[4], B)+\mu(n[6], B)+\lambda(results, B, A) = 5+6+10+5 = 26$. If most free target is not B, we try to change node $n[2]$ to use target A. A new rank is 29 because the cost of $\lambda(d[2], n[2], n[4])$ and $\mu(n[2], A) - \mu(n[2], B)$. So if wait time for target B is more than $29 - 26 = 3$ units, this node n would be dispatched to target A.

4. Experimental Results

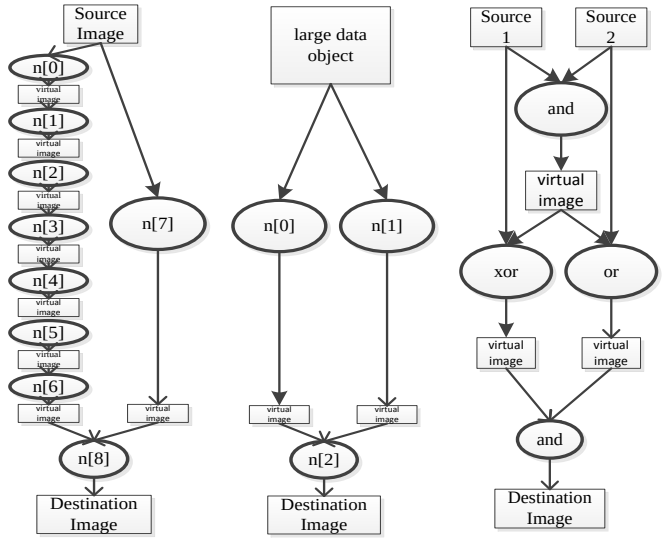
We implement our coarsen-scheduling algorithm within the OpenVX sample implementation released by Khronos Group. Our experiment environment is on AMD A10-7850k APU with 4 core CPUs and a Spectre GPU inside. This experimental OpenVX framework is built on Ubuntu 14.04.2 LTS with OpenCL runtime support and compile by gcc 4.8.2 compiler with -O3 optimization option. In addition, a profile procedure profiles each node at the first time execution of this framework. The profile information includes the average time a node spent on computation and the average data transfer time for each parameter over 100 execution on fixed size data objects(here is 640*480). Every time the framework initialize its context, these profile informations will be loaded. We exclude the nodes do not support both C and OpenCL target in this experiment and list the profiling results in Table 1.

kernel name	target	compute(ms)	transfer(ms)	total(ms)
table_lookup	c-model	2.179		2.179
	pc.opencl	0.085	0.034	0.119
and	c-model	2.957		2.957
	pc.opencl	0.223	0.049	0.272
or	c-model	2.903		2.903
	pc.opencl	0.230	0.049	0.280
xor	c-model	2.980		2.980
	pc.opencl	0.227	0.052	0.279
not	c-model	1.952		1.952
	pc.opencl	0.157	0.032	0.188
histogram	c-model	1.871		1.871
	pc.opencl	0.155	0.033	0.189
box3x3	c-model	11.846		11.846
	pc.opencl	0.198	0.034	0.232
gaussian3x3	c-model	11.855		11.855
	pc.opencl	0.195	0.031	0.226

Table 1: Profiling results of OpenVX kernels

As the profiling results shown in the above table, we can observe that all the nodes on OpenCL target(pc.opencl) consume lower processing time than the nodes compute on CPU target(c-model) (even take account of both computation time and data transfer time). Even the slowest kernel on GPU(the 'or' kernel with 0.280 ms) is 6.68 times faster than the fastest kernel on CPU(the 'histogram' kernel with 1.871 ms). However, there will be more and more accelerating targets and complex applications adopting the OpenVX framework. So the nodes with smaller time difference on different targets will appear in the near future. In the case that different accelerators with similar computation ability, nodes may have less different processing time on different targets for individual execution. For example, a complex computation algorithm may consume similar processing time on GPU and DSP. In this case, our proposed algorithm can schedule these nodes for a good arrangement.

In this experimental framework, the node characteristics make our coarsen-scheduling algorithm have similar results as the target priority design in the original sample implementation. The target priority design dispatch nodes to OpenCL target first, then OpenMP and C model. Nodes have OpenCL target support will choose it and get good performance result because most of the nodes have good result on OpenCL target. But there still exist some cases that will benefit from



(a) Load unbalance (b) Data size (c) Data transfer

Fig. 4: Case analysis for our algorithm

our algorithm. In Figure 4a, a graph with two paths that one is with 7 GPU nodes(node n[0] to n[6]) on it and the other is just a CPU node(node n[7]) on it. A sink node(n[8]) merges the two paths to generate output image. In this case, the load dispatch mechanism in node scheduling algorithm will make the right decision since the wait time for node n[7] to run on GPU is longer than directly compute this node on CPU. The number 7 is because there is about 7 times ratio of execution time between CPU nodes and GPU nodes. Another case in Figure 4b is to exploit the weakness of GPU computing, the data transfer overhead between host and OpenCL computing device. When a node with large data object input will lead a long time to transfer data from host memory to GPU. In this situation, the consideration of communication time in our algorithm will choose CPU for the computation.

Also, we evaluate our method with running programs show in Figure 4c. The size of the clusters are control by the $isValidWindowsSize()$ function in node coarsen algorithm. If we let the windows size to be 4 or more, all the nodes on the graph can be cluster into just one cluster. Since all nodes are in the same cluster, we can benefit from this by memory locality property. The virtual images can directly reside on the OpenCL device because our algorithm consider the graph characteristic. The time results compare for OpenVX framework with our algorithm and sample implementation without our algorithm are show in the Table 2. The transfer column has three sub-column for each parameter of a node row. It represent the parameters connected to that node. So in row 0, the p[0] parameter of node 'and' is source 1 image, the p[1] parameter is source 2 image and the p[2] parameter is virtual image connected with 'and', 'xor' and 'or' node. We can observe that the transfer time for p[2] in row 0, p[1] and p[2] in row 1, p[0] and p[3] in row 2 and p[0] and p[1] in row 3 have been

node	memory locality	compute(ms)	transfer(ms)		
			p[0]	p[1]	p[2]
and	YES	1.169	0.538	0.060	0.007
	NO	1.156	0.557	0.061	0.041
xor	YES	0.445	0.038	0.008	0.007
	NO	0.578	0.025	0.017	0.015
or	YES	0.403	0.007	0.026	0.008
	NO	0.428	0.015	0.020	0.012
and	YES	0.173	0.008	0.008	0.017
	NO	0.191	0.023	0.016	0.016

Table 2: Time results with and without memory locality reduced.

5. Conclusions

In this paper, we proposed a two phase scheduling method to address the scheduling issues of OpenVX task graph. The proposed two phase scheduling method considered both memory locality and system throughput to schedule OpenVX Programs on heterogeneous multi-core systems. The first phase of our algorithm performs coarsen schemes to cluster nodes together and the second phase schedule nodes into different targets. The result of our experiment shows that our algorithm works well in scheduling OpenVX programs on heterogeneous environments and analyzed some cases this algorithm will be benefit.

Acknowledgment

This work is conducted under the "The core technologies of smart handheld devices project" of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China and is also in part by the MOST (Ministry of Science and Technology) project.

References

[1] The Khronos Group, "Openvx - portable, power-efficient vision processing," 2015. [Online]. Available: https://www.khronos.org/openvx/

[2] OpenMP, "The openmp api specification for parallel programming," 2015. [Online]. Available: http://openmp.org/wp/

[3] The Khronos Group, "Opencl, the open standard for parallel programming of heterogeneous systems," 2015. [Online]. Available: https://www.khronos.org/opencl/

[4] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, Mar 2002.

[5] J. chiou Liou and M. A. Palis, "An efficient task clustering heuristic for scheduling dags on multiprocessors," in *Multiprocessors Workshop on Resource Management, Symposium of Parallel and Distributed Processing*, 1996, pp. 152–156.

[6] C. Chen, Y. Chang, Y. Chen, C. Yang, and J. K. Lee, "Switching supports for stateful object remoting on network processors," *The Journal of Supercomputing*, vol. 40, no. 3, pp. 281–298, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11227-006-0023-2

[7] Y. Wen, Z. Wang, and M. O' Boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *Proceedings of the 21st Annual IEEE International Conference on High Performance Computing (HiPC'14)*, 2014.

[8] E. Rainey, J. Villarreal, G. Dedeoglu, K. Pulli, T. Lepley, and F. Brill, "Addressing system-level optimization with openvx graphs," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, June 2014, pp. 658–663.

Intelligent Usage Management of Shared Resources in Simultaneous Multi-Threading Processors

Yilin Zhang[1] **and Wei-Ming Lin**[2]
[1]Advanced Micro Devices, Inc. Austin, TX 78735, USA
[2]Department of Electrical and Computer Engineering
The University of Texas at San Antonio, San Antonio, TX 78249, USA

Abstract— *Simultaneous Multi-Threading (SMT) is a technique that allows multiple independent threads to concurrently execute multiple instructions in each cycle. How to efficiently manage the distribution of the shared resources to simultaneously executing threads is critical to the performance of SMT processors. In this paper, we propose a comprehensive allocation algorithm on the shared resources in an SMT system. We show that, physical register file, Issue Queue (IQ) and write buffer are the most important shared resources in a typical SMT system. By limiting the portion of these shared resources that a thread is allowed to occupy at any given time, the overall system throughput is enhanced by a very significant margin. An improvement in IPC of up to 98.39% and 105.35% is observed when the proposed technique is applied to a 4-threaded and an 8-threaded SMT system, respectively. Furthermore, execution fairness among the threads is not sacrificed as demonstrated by an improved Harmonic IPC of up to 60.8% and 40.96% respectively.*

Keywords: Simultaneous Multi-Threading; Superscalar; Shared Resources Management

1. Introduction

Simultaneous Multi-Threading processors improve the execution efficiency of traditional superscalar processors by allowing the issue of instructions from different threads in the same cycle ([1], [2]). In modern out-of-order SMT processors, this execution model implies the sharing of several resources among threads with a goal to achieve a comparable performance to that from using multiple copies of superscalar processors while saving a significant amount of resources. In order to achieve this, the most significant potential drawback in the SMT execution model is inter-thread blocking due to resource sharing has to be carefully addressed.

In an SMT system, the resources shared among threads normally include physical register file, various machine bandwidths (e.g., inter-stage bandwidth, read/write ports for register file and memory, etc.), inter-stage buffers (e.g., Issue Queue (IQ)), functional units, write buffer, etc. In some processors, Instruction Fetch Queue (IFQ) and Re-Order Buffer (ROB) are also shared among threads. SMT systems in general aim at exploiting Thread-Level Parallelism (TLP) to overcome the limitation of Instruction-Level Parallelism (ILP) and thus improve the overall performance. However, an inefficient resource utilization among threads can easily lead to an undesirable performance outcome.

There have been many research results aiming at improving the shared resources allocation in SMT systems. Some of them manage the resource usage among threads by modifying the fetch policy. For example, ICOUNT [3] assigns a higher fetching priority to a thread with fewer instructions in pre-issue stages; STALL and FLUSH [4] adopts a fetch policy to address issues from L2 cache misses; a dynamical fetch policy DCRA presented in [5] is a technique based on memory performance of each thread to exploit parallelism beyond stalled memory operations; Some other research efforts have also examined the allocation of the shared buffers in the pipeline for a more efficient resource utilization. For example, APRA dynamically assigns resources (IFQ, IQ and ROB) to threads according to changes of threads' behavior [6]. Hill-Climbing [7] is a learning-based algorithm that uses performance feedback to partition the shared hardware resources in the pipeline. In [8], a write buffer occupancy capping technique is proposed for an efficient utilization on the shared write buffer. Some other research results have enhanced the overall performance by improving the utilization of IQ ([9], [10], [11], [12]).

Each of the aforementioned techniques in the literature attempts to optimize a local stage of the pipeline. Such an approach may relieve a certain degree of problem in the target stage but its performance-improving potential may be significantly curtailed without considering similar constraints posed by other stages. In this paper, we instead propose a "global" resource utilization optimizing technique considering the three most important shared resources in SMT, including physical register file, IQ and write buffer. This technique aligns all these shared resources according to their respective sizes and proportionally caps the portion of each resource that a thread is allowed to occupy at any time. Compared with existing techniques, the proposed method involves adding very little extra control logic and hardware overhead for scheduling. As our simulation results show, the proposed technique improves IPC (Instruction Per Cycle) by as high as 98.39% and 105.35% in a 4-threaded and an 8-threaded SMT system, respectively. Furthermore, the

execution fairness (measured with the Harmonic IPC) is enhanced by up to 60.8% and 40.96%, respectively. From the resource utilization's perspective, with the proposed technique, the system is able to deliver a better throughput even with a smaller amount of resources.

The rest of this paper continues with descriptions on the mechanism of the shared resources (physical register file, IQ and write buffer) in an SMT system. Section 3 is devoted to the introduction of the simulation environment adopted by this research including the performance metrics used. The proposed technique is described in Section 4, followed by the complete simulation results in Section 5. It is then wrapped up by several concluding remarks in the last section.

2. Resource Sharing in SMT Systems

The three target shared resources that we propose to systematically allocate among threads are (1) physical register file, (2) IQ and (3) write buffer. These shared resources are considerd "blocking" resources since such a resource unit/buffer once occupied cannot be preempted by another instruction (from the same or a different thread) and the duration of such an occupation is not "pre-determined" due to unexpected delays. This section is devoted to the introduction on how these shared "blocking" resources are utilized in SMT systems and the impacts on the overall performance.

2.1 Physical Register File

A physical register file contains more register slots than those defined in the ISA, the so-called "architectural" registers, for register renaming. In order to ensure proper execution, a physical register is assigned to an architectural register of an instruction at rename stage and will not be released (deallocated) until the very next "co-renamed" instruction, the one that writes to the same architectural register, is committed. A rename table is used to record the mapping between an architectural register and its latest assigned physical register. Performance of a system adopting this approach is heavily influenced by the availability of these physical registers in that the lack of physical registers will delay the register renaming process and subsequently prevent new instructions from entering the system.

Due to its deallocation mechanism, physical register file is partially shared among threads in an SMT system in that a portion of physical register file is always devoted to the architectural registers. Figure 1 depicts the organization of the physical register file, where R_t, R_a, and R_r are used to denote the number of all physical registers, per-thread architectural registers, and all physical registers for renaming, respectively. Thus, $\{R_r = R_t - n \times R_a\}$ where n is the number of threads in the system. Note that R_r will be effective amount of rename registers for our proposed allocation technique.

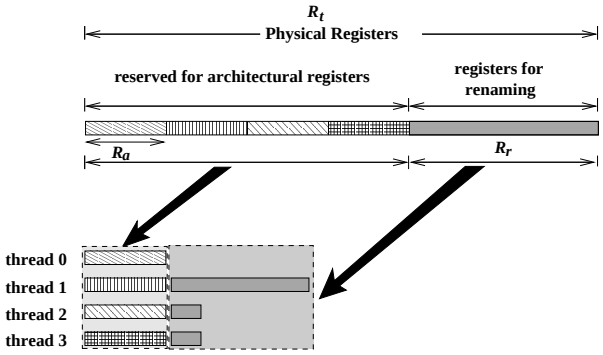


Fig. 1: The Organization of Physical Register File in a 4-threaded SMT System

Compared to a single-threaded system, multiple threads sharing several key resource components would most likely lead to a longer expected latency per instruction, which in turn prolong occupancy duration of a rename register by an instruction. Another reason for requiring a large register file comes from the real-time changing behavior of competing threads and the potentially long occupancy duration of a register from the time it is allocated to when it is released. Without employing a physical register file unreasonably large, one has to manage the allocation effecetively in order to achieve the desirable throughput improvement from an SMT system.

2.2 Issue Queue

Issue Queue (IQ) is also known as a reservation station (either centralized or functional unit-specific) where instructions wait for being issued into the corresponding functional units whenever the issue conditions are met, i.e. operands are ready and the requested functional unit becomes available. In an SMT system, IQ is normally shared among threads and the instructions from these thread-specific ROBs have to "compete" for IQ entries through a dispatching scheduling algorithm. Having its output sent into a tightly shared resource, as well as encountering the unexpected long latencies caused by deep-level cache miss, miss-speculation, etc., the instruction dispatch stage is considered one of the most critical stages that dearly affect the overall system performance. Figure 2 depicts a basic functional block diagram of a 4-threaded SMT system.

2.3 Write Buffer

A write buffer (or referred to as "store buffer" in some articles) is designed to hide long write latency when CPU encounters cache misses ([13], [14]). Although there are many different designs and interpretations, the write buffer is usually referred to the buffer that resides between CPU and cache with the purpose of absorbing long write latencies from cache-miss writes without further slowing down the

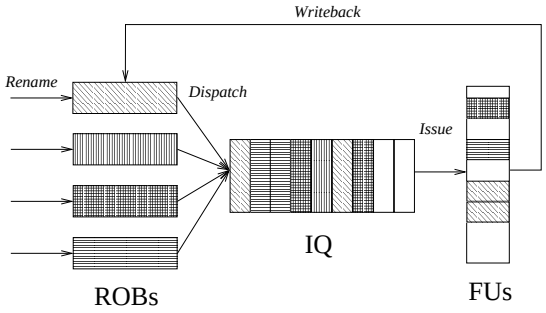


Fig. 2: A Simple 4-threaded SMT Instruction Processing Flow

CPU [14]. All writes initiated by the CPU will be first sent to this buffer in order for the cache/memory controller to handle writes of different latencies due to cache misses along with bus contention at different levels. A typical two-level cache organization with write-back policy at both levels is shown in Figure 3 [8].

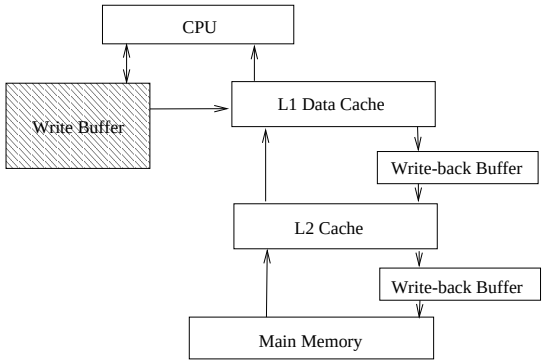


Fig. 3: A Typical Two-Level Cache Organization with a Write Buffer

The data to be written will stay in the write buffer until the write operation finishes. This means some data may occupy their write buffer slots potentially for a long period of time if they encounter deep-level cache misses. Write-merging (also known as write-coalescing) is a technique aggregating writes to the same cache block (line) to reduce miss penalty as well as the buffer size requirement, which is prevalent in most modern processors [14]. In a write-coalescing buffer, a new write will be merged into an existing write in the buffer if they belong to the same block, i.e. they will share the same entry in the buffer until they are eventually written. When no merging exists, a write will not be allowed to commit if the buffer is full.

Note that instruction-committing in a superscalar system is always processed "in-order" to ensure precise exception and correct speculative processing. The availability of the write buffer significantly influences the throughput of the system since if a write instruction cannot commit due to a full write buffer, all subsequent instructions will be blocked as well.

2.4 Summary

An SMT system suffers even more from the potentially long occupancy time on its shared resources compared to a single-threaded system in that long occupancy from one slower thread could easily hog most of the shared resources and subsequently stall all other threads' processing. Due to the cost-efficiency concern, we normally can not afford a very large size of the shared resources, especially when the shared resources are on-chip components. Employing large shared resources may not only be cost inhibitive but also lead to longer access latency and excessive area/power consumption. In addition, such a choice is exactly contradictory to the underlying design philosophy of an SMT in sharing resources that are supposed to be less than that from multiple copies of single-threaded systems. Thus, effectively managing reasonable-sized shared resources in an SMT system is a must to achieve a desirable balance between throughput and cost.

3. Simulation Environment

The simulation environment adopted by our research, including the simulator and the workloads used are described in this section.

3.1 Simulator

We use the M-Sim [15], a multi-threaded micro-architectural simulation environment model, to estimate performance of the proposed scheme. The M-Sim includes accurate models of the pipeline structures such as explicit register renaming, concurrent execution of multiple threads, separate ROB, Load-Store Queue (LSQ) which are necessary for an SMT model. The physical register file, IQ, functional units and write buffer are shared among threads, while branch predictor is exclusive to each thread. The detailed configuration is shown in Table 1.

3.2 Workloads

Simulation runs for multi-threaded workloads in this paper all use the mixed SPEC CPU2006 benchmark suite [16] with mixtures of various levels of ILP for diversified representation of workloads. ILP classification of each benchmark is obtained by initializing it in accordance with the procedure mentioned in Simpoints tool and simulated individually in a simplescalar environment. Three types of ILPs, low ILP (memory bound), medium ILP and high ILP (execution bound), are so identified. As shown in Table 2 for 4-threaded workloads and Table 3 for 8-thread workloads, a number of multi-threaded workloads are used with threads of various mixtures of ILP types.

Parameter	Configuration
Machine Width	8 wide fetch/dispatch/issue/commit
L/S Queue size	48-entry Load/Store queue
ROB & IQ size	128-entry ROB, 32-entry IQ
Function Units & Latency (total/issue)	4 Int Add (1/1) 1 Int Mult (3/1) / Div (20/19) 2 Load/Store (1/1), 4 FP Add (2/1) 1 FP Mult (4/1) / Div (12/12) / Sqrt (24/24)
Physical registers	integer and floating point as specified in the paper
L1 I-cache	64KB, 2-way set-associative 64-byte line
L1 D-cache	64KB, 4-way set-associative 64-byte line write-back, 1 cycle access latency
L2 Cache unified	512KB, 16-way set-associative 64-byte line write-back, 10 cycles access latency
BTB	512 entry, 4-way set-associative
Branch Predictor	bimod: 2K entry
Pipeline Structure	5-stage front-end (fetch-dispatch) scheduling (for register file access: 2 stages, execution, write back, commit)
Memory	32-bit wide, 300 cycles access latency

Table 1: Configuration of the Simulated Processor

Mix	Benchmarks	Classification (ILP)		
		Low	Med	High
Mix 1	libquantum, dealII, gromacs, namd	0	0	4
Mix 2	soplex, leslie3d, povray, milc	0	4	0
Mix 3	hmmer, sjeng, gobmk, gcc	0	4	0
Mix 4	lbm, cactusADM, xalancbmk, bzip2	4	0	0
Mix 5	libquantum, dealII, gobmk, gcc	0	2	2
Mix 6	gromacs, namd, soplex, leslie3d	0	2	2
Mix 7	dealII, gromacs, lbm, cactusADM	2	0	2
Mix 8	libquantum, namd, xalancbmk, bzip2	2	0	2
Mix 9	povray, milc, cactusADM, xalancbmk	2	2	0
Mix 10	hmmer, sjeng, lbm, bzip2	2	2	0

Table 2: 4-threaded Workload for Simulation

3.3 Metrics

For a multi-threaded workload, total combined IPC is a typical indicator used to measure the overall performance, which is defined as the sum of each thread's IPC:

$$\text{Overall\_IPC} = \sum_{i}^{n} \text{IPC}_i \quad (1)$$

where n denotes the number of threads per mix in the system. However, in order to preclude starvation effect among threads, the so-called Harmonic IPC is also adopted, which reflects the degree of execution fairness among the threads, namely,

$$\text{Harmonic\_IPC} = n / \sum_{i}^{n} \frac{1}{\text{IPC}_i} \quad (2)$$

In this paper, these two indicators are used to compare the proposed algorithm to the baseline (default) system. The following metric indicates the improvement percentage

Mix	Benchmarks	Classification (ILP)		
		Low	Med	High
Mix 1	libquantum, dealII, gromacs, namd, soplex, leslie3d, povray, milc	0	4	4
Mix 2	libquantum, dealII, gromacs, namd, lbm, cactusADM, xalancbmk, bzip2	4	0	4
Mix 3	hmmer, sjeng, gobmk, gcc, lbm, cactusADM, xalancbmk, bzip2	4	4	0
Mix 4	libquantum, dealII, gromacs, soplex, leslie3d, povray, lbm, cactusADM	2	3	3
Mix 5	dealII, gromacs, namd, xalancbmk, hmmer, cactusADM, milc, bzip2	3	2	3
Mix 6	gromacs, namd, sjeng, gobmk, gcc, lbm, cactusADM, xalancbmk	3	3	2

Table 3: 8-threaded Workload for Simulation

averaged over the selected mixes, which is applied to both Overall_IPC and Harmonic_IPC, namely,

$$\text{Imp\_}\% = \left(\sum_{j}^{m} \frac{\text{IPC}_j^{\text{new}} - \text{IPC}_j^{\text{baseline}}}{\text{IPC}_j^{\text{baseline}}} \times 100\%\right)/m \quad (3)$$

where m denotes the number of mixes of the workload in our simulation.

4. Proposed Method

The proposed allocation technique is considered modifications on the schedule algorithms at the stages in the pipeline, including register renaming, dispatch and commit. Instead of trying to optimize allocation of one single resource or three individual resources indendepently, we propose to align the allocation of all three resources in a congruent manner so as to maximize the effect of resource allocation benefits. A very simple control mechanism is imposed on assigning the shared resources entries; that is, each thread is not allowed to occupy more than the preset fraction of resources at any time. Once a thread reaches the maximal number of entries of the shared resource, the resource allocation process for that thread is stopped at the corresponding stage.

In order to have a systematic controlling parameter for all of the three shared resources of different sizes, we adopt the notion of "cap fraction (F)" to represent the ratio between the maximal number of entries of the resource that a thread is allowed to occupy and the size of the corresponding resource. Thus, the absolute maximal number of entries allowed to occupy, known as "cap value (C)" can be represented by $C = F \times S$, where S is the size of the corresponding shared resource. All of the simulations are performed with F ranging from $1/16$ to $16/16$ with the unit of increment being $1/16$. The remaining portion of this section discusses the details of the modified scheduling algorithms on each stages.

- **Register Renaming**

As aforementioned, physical register file is a partially shared resource in an SMT system, a portion of which is dedicated to the architectural registers for each thread. The rest of the physical register file are used for register

renaming, and, if left uncontrolled, it can be overwhelmingly occupied by a single thread. The proposed method aims at making proper allocation of the renaming part of physical register file among threads. Since the occupation rate on floating point register file is very low during the execution of all workloads; therefore our allocation algorithm manages only the usage of integer physical register file by capping the allowance for each thread.

One should also notice that since register renaming is an in-order process, an integer instruction that is stopped at this stage due to the proposed capping process will stop any subsequent instructions (integer or floating point) from proceeding and therefore it essentially manages the usage of floating point physical register file as well in an indirect way.

- **Dispatch**

In a typical SMT system, a shared IQ is required to hold all the necessary information for instructions, thus further increasing the number of entries in the shared IQ usually is hampered by the cost factor, and therefore the utilization of these limited resources becomes very critical to the overall system performance. Once an IQ slot is allocated to an instruction, it will be occupied until the instruction eventually becomes ready for execution, even for hundreds of clock cycles. The proposed technique limits the maximal number of entries that a thread is allowed to occupy at any time, so as to prevent all IQ slots being exhausted by the long-latency instructions. The proposed technique is a simple yet efficient solution in that it ensures the more active threads have opportunity to use more resources than the equally allocation amount while avoiding resource shortage for those inactive threads.

- **Commit**

The default commit algorithm is modified by the proposed technique on the commit conditions for a store instruction when it is going to retire from the system. Normally, a store instruction from a thread when reaching the head of its ROB may come across a delay from waiting for one of the following shared resources: (1) commit bandwidth, (2) write port and (3) write buffer. In a typical SMT system with a sufficient commit bandwidth and write ports, the only potential performance-degrading hazard among these is the long delay from waiting for an available write buffer entry, due to its "blocking" nature as apposed to the former two being "non-blocking". To reduce the the effect of this potential hazard, the proposed technique adds another condition on the commit algorithm that unless the number of write buffer entries a thread is currently occupying is less than the preset cap value, the store instruction will not be allowed to commit. This newly imposed condition is added to avoid overwhelming write buffer occupation by those long-latency instructions, thus to reduce the delay in waiting for a write buffer entry.

Any control technique aimed at eliminating the intended overwhelming occupancy problem, no matter how complicated the technique is, will suffer a drawback in compromising the flexibility that the original untampered shared resource offers. The benefit brought along by the technique will then in turn be offset to a degree by the compromise. In our approach, the degree of compromise depends on the cap fraction (F) imposed. First of all, with n denoting the number of threads in the system, obviously performance delivered by a cap value (C) smaller than S/n (i.e. $F < 1/n$) will very unlikely be optimal since there will be at least $S - n \times C$ entries constantly left unused. With a specified cap fraction, the overall performance may still vary depending on behavior of the threads (e.g. instruction's operation latency, cache performance, number of writes to the same architectural register, etc.) and even the sizes of the resources in the system as well. The proposed technique would certainly manifest its benefits more when the behaviors of each thread greatly vary, or when the competition of resources is severe.

The main compromise between the benefits and drawbacks from the techniques lies at the setting of the cap fraction. If the cap fraction is set too high, the intended function of this technique in suppressing single thread's dominating occupancy will not be effective. On the other hand, if the cap fraction is set too low, overall performance may suffer if concurrent usage on the shared resources in multiple threads do not happen often enough. Our simulation to be presented in the next section will be used to study the effect of this compromise and to lend us a good indication of where a good range of cap fraction should be.

5. Simulation Results

Based on the simulation environment and the workloads described in Section 3, the proposed technique is tested compared to the default system. For each simulation run the cap fraction (F) is set to $d/16$ where d is an integer varying from 1 to 16. The resulted cap value, if not an integer, is always rounded down to the next integer; that is, the overall cap arrangement will be a combination of the three respective cap values (C_R, C_Q, C_W), where $C_R = \lfloor F \times R_r \rfloor$, $C_Q = \lfloor F \times Q \rfloor$ and $C_W = \lfloor F \times W \rfloor$.

Figure 4 demonstrates the proposed technique's influence on the performance with different sizes of shared resources on 4-threaded workloads. Each point represents an average performance over the 10 mixes adopted for 4-threaded workloads. Average improvement from this technique can reach up to 98.39%. As the sizes of the shared resources increase, the optimal cap fraction (F) shifts from $4/16$ to $6/16$. This further ascertains our conjecture that optimal setting of F does not fall below $1/n$. It also indicates that a different emphasis should be made on fairness and flexibility when the amount of shared resources varies. Namely, with a smaller amount of resources, it is more critical to stress on fairness to ensure each thread an fair amount of opportunity

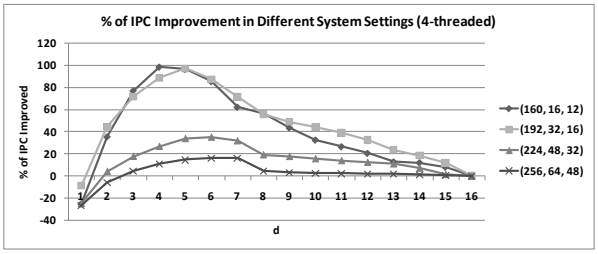


Fig. 4: Average Percentage of IPC Improvement with Different Resource Settings for 4-threaded Workloads

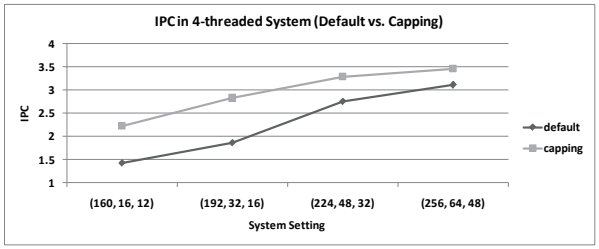


Fig. 5: Comparison of IPC for Resource Saving

to proceed; on the other hand, when more resources are available, the capping should be relaxed more so as to allow more active threads to proceed faster. Another interesting observation from the results is that there is still a significant performance gain even the cap fraction is set to below $1/n$, in which situation a portion of the resources constantly stay unused. For example, in a 4-threaded system, when $d = 3$ there are 25% $(1 - 4 \times \frac{3}{16})$ of resources constantly unused, and yet a 71.74% of improvement is still delivered. Such a phenomenon clearly implies that, when the resource usage is left uncapped as in the default setting, bursting demands from one or very few threads may easily clog up the resources if they run into any kind of long pipeline latency, which explains why even leaving such a huge amount of resources unused and allowing each thread only a bare minimum to utilize still leads to a significant improvement.

This result in general solidifies all our aforementioned claims. First of all, the effectiveness of our technique is more prominent for smaller sizes of resources due to higher competition. Secondly, optimal setting of the cap fraction is not less than the point of $1/n$. Thirdly, there exists an obvious compromise between the ensuing benefit and drawback from setting a different cap fraction value. When the cap fraction is set higher than the optimal point, the higher the cap is the less benefit this technique can produce. On the other hand, once the cap fraction is set lower than the optimal value, the tighter (lower) the cap is, the less flexible the resource allocation becomes and the benefit of sharing becomes less.

Another analysis on this result comparing the proposed capping technique to the default one can be made from the resource-saving point of view. Figure 5 indicates that the proposed capping method when applied to a 4-threaded system equipped with a relatively small amount of resources is capable of delivering a similar, or even better, throughput to that produced by the default system with even larger-sized buffers. For example, the system with $(R, Q, W) = (160, 16, 12)$ with the proposed capping technique can actually deliver a performance of 19.5% better than that of a default system with $(R, Q, W) = (192, 32, 16)$, reflecting a significant amount of resource saving.

An improvement on the overall IPC from a technique will be less valuable if it comes with a sacrifice on the execution fairness among threads. Such a sacrifice is a commonly observed scenario in typical shared-resource systems when the overall performance is improved by devoting more resources to the faster task(s) while significantly trading off the processing fairness among the tasks. Figure 6 shows the percentage of improvement in Harmonic IPC (defined as in Eq. 2) for 4-threaded workloads. From this result,

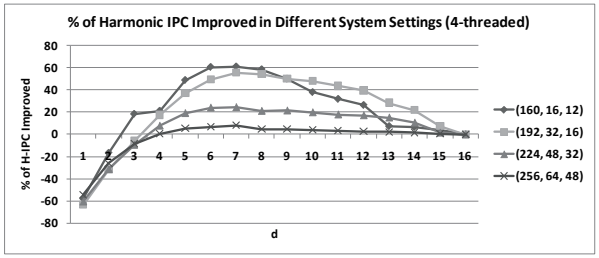


Fig. 6: Harmonic IPC Comparison with Varying System Settings for 4-threaded Workloads

one can see that the improvement on the overall IPC from the proposed method not only leads to no unfair execution among threads but incurs a positive effect with up to 60.8% improvement.

Figure 7 shows that the proposed technique delivers a considerable improvement in a system with more threads as well. As the results show, in an 8-threaded system, a 105.35% and a 40.96% improvement is achieved for IPC and Harmonic IPC, respectively.

We further look into the "aggregate effect" from this proposed global capping when compared to the individual effect from each independent buffer-capping technique applied to its respective pipeline stage. Due to the "uniform" cap fraction applied to all three buffers in our combined capping technique, one would expect the performance improvement from this may not be as effective knowing that the optimal cap fraction for each individual buffer is different when applied separately. Figure 8 displays such a comparison result. When capping is applied only to the register file, a peak improvement of 42.9% happens at $d = 5$, while the optimal d value for individual IQ and write buffer application

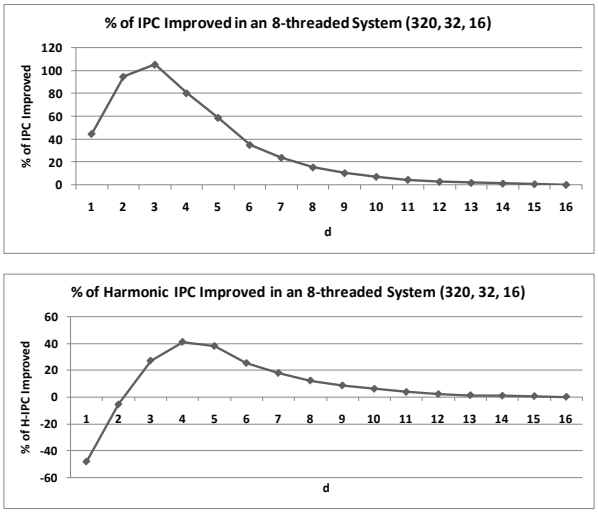


Fig. 7: Performance Improvement in 8-threaded SMT system

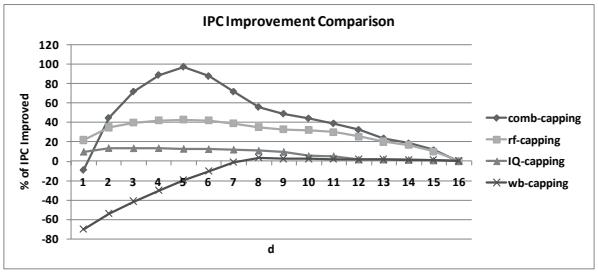


Fig. 8: Performance Comparison: Combined Capping vs. Individual Capping

happens at $d = 4$ and $d = 8$ for an improvement of 13.6% and 3.3%, respectively. When applying the uniform capping to all three buffer, the optimal improvement of 97.2% happens at $d = 5$, an improvement much superior to the add-up of the improvement margins from three individual applications. This clearly indicates that a comprehensive arrangement on the three stages of shared resources is more likely to exploit the full performance potential in an SMT system, rather than applying optimizing techniques to an individual resource.

6. Conclusions

This paper presents an allocation algorithm for optimizing the distribution of shared resources in an SMT system. We clearly show that by setting a cap fraction on the most critical shared resources in an SMT system (that is, physical register file, IQ and write buffer) limiting the maximal resources that a thread is allowed to occupy, the overall performance is enhanced significantly. The simulation results demonstrate that a comprehensive management on multiple shared resources can exploit extra performance potential compared to independently applying to individual ones. The proposed technique is simple yet efficient in allocating shared resources and it requires very marginal hardware overhead and imposes little extra constraints on the clock rate.

References

[1] H. Hirate, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase and T. Nishizawa, "An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads", *In the Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 136-145, May 1992.

[2] D. Tullsen, S. J. Eggers and H. M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", *In the Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 392-403, May 1995.

[3] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M .Levy, J. L. Lo and R. L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous MultiThreading Processor", *In the Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pp. 191-202, May 1996.

[4] D. M. Tullsen and J. A. Brown, "Handling Long-latency Loads in a Simultaneous Multithreading Processor", *In the Proceedings of the 34th International Symposium on Microarchitecture*, pp. 318-327, December 2001.

[5] F. J. Cazorla, A. Ramirez, M. Valero and E. Fernandez, "Dynamically Controlled Resource Allocation in SMT Processors", *In the Proceedings of the 37th International Symposium on Microarchitecture*, pp. 171-182, December 2004.

[6] H. Wang, I. Koren and C. M. Krishna, "An Adaptive Resource Partitioning Algorithm for SMT Processors", *In the Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 230-239, October 2008.

[7] S. Choi and D. Yeung, "Learning-Based SMT Processor Resource Distribution via Hill-Climbing", *In the Proceedings of the 33rd Annual International Symposium on Computer Architecture*, pp. 239-251, June 2006.

[8] Y. Zhang and W.-M. Lin, "Write Buffer Sharing Control in SMT Processors", *2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13)*, July 2013.

[9] T. Nagaraju, C. Douglas, W.-M. Lin and E. John, "Effective Dispatching for Simultaneous Multi-Threading (SMT) Processors by Capping Per-Thread Resource Utilization", *The Computing Science and Technology International Journal*, Vol. 1, No. 2, pp. 5-14, December 2011.

[10] Y. Zhang, C. Douglas and W.-M. Lin, "On Maximizing Resource Utilization for Simultaneous Multi-Threading (SMT) Processors by Instruction Recalling", *2012 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12)*, July 2012.

[11] Y. Zhang and W.-M. Lin, "Capping Speculative Traces to Improve Performance in Simultaneous Multi-Threading CPUs", *The 18th International Conference on Parallel and Distributed Processing Techniques and Applications (IPDPS'13) Workshop on Multithreaded Architectures and Applications*, May 2013.

[12] Y. Zhang, M. Hays, W.-M. Lin and E. John, "Autonomous Control of Issue Queue Utilization for Simultaneous Multi-Threading Processors", *The 22nd High Performance Computing Symposium (HPC 2014)*, April 2014.

[13] P. P. Chu and R. Gottipati, "Write Buffer Design for On-Chip Cache", *In the Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 311-316, October 1994.

[14] J. L. Hennessy and D. A. Patterson, "Computer Architecture:A Quantitative Approach", *Morgan Kaufmann Publishers*, 2007.

[15] J. Sharkey, "M-Sim: A Flexible, Multi-threaded Simulation Environment", *Tech. Report CS-TR-05-DP1*, Department of Computer Science, SUNY Binghamton, 2005.

[16] Standard Performance Evaluation Corporation (SPEC) website, http://www.spec.org/.

Container-based Cluster Management Platform for Distributed Computing

Ju-Won Park and Jaegyoon Hahm
Div. of Supercomputing, KISTI,
245 Daehak-ro, Yuseong-gu, Daejeon 305-806, Korea

Abstract— *Several fields of science have traditionally demanded large-scale workflows support, which requires thousands of CPU cores or more. Since users' demands for software packages and configuration is the difference, an approach to making available in real time a service environment desired by users without significant challenges for administrators is necessary. In this paper, we present a container based cluster management platform and introduce an implementation case to minimize performance decline and to provide a dynamic distributed computing environment desired by users. This paper makes the following contributions. First, a container based virtualization technology is assimilated with resource and job management system to expand its applicability to support large-scale scientific workflows. Second, an implementation case in which docker and HTCondor are interlocked with each other is introduced. Lastly, docker and native performance comparison using two widely known benchmark tools and Monte-Carlo simulation results implemented using various programming languages are presented.*

Keywords: Container-based virtualization, Docker, HTCondor, Distributed computing

1. Introduction

Traditionally, high energy physics, oceanography, meteorology, astronomy, and space science require large-scale workflows demanding CPUs consisting of more than several thousand cores [1], [2]. These scientific workflows come in a variety of forms, ranging from high throughput computing (HTC) combining millions of loosely-coupled tasks to high performance computing (HPC) referring to a tightly-coupled from such as message passing interface (MPI) tasks being processed simultaneously by several thousand cores. In order to handle such large-scale scientific workflows, large-capacity cluster systems such as supercomputers are widely used. Such a large-capacity cluster system usually supports RJM (Resource and job Management) functions, enabling multitude of uses to share resources fairly. However, as the resources are shared by multiple users and organizations, there still exist many challenges, the biggest of which is the difference in users' demands for software packages and configurations. Because of these challenges, in practice an operating system (OS) and software stacks are often once installed and kept unchanged for a very long time [3]. These rigid utilization practices pose many constraints to new technology development initiatives, dampening expectation of performance increase following software version upgrade.

To overcome such issues of rigid utilization practices and deliver more dynamic service environments to users, many studies - aimed at configuring and providing users with clusters using virtualization resources built on Xen or KVM-based virtualization technologies - have been conducted [4], [5], [6]. In fact, many scientists are working on researches, using the VM available from Amazon EC2. Although significant performance improvement has been achieved thanks to the improvement of hypervisor technology and development of various techniques such as passthrough approach, overhead incurred by hypervisor inevitably compromises performance [7]. Because of such constraints, container-based virtualization technologies such as Linux-VServer, OpenVZ, and LXC are frequently utilized recently [7], [8], [9].

This paper presents a container-based cluster management platform and introduces an implementation case to minimize performance decline and to provide a dynamic distributed computing environment desired by users. As container-based virtualization technology compromises performance less than hypervisor-based one, the former can reach near-native performance. This paper is designed to contribute in three regards. First, container-based virtualization technology is assimilated with RJM to expand its applicability to cluster environment in a bid to support large-scale scientific workflows with near-native performance. As conventional container resource utilization approaches were focused on providing user-customized service environment in a single computer, they were not suitable for supporting scientific workflows utilizing resources on a large scale. Second, an implementation case in which docker [10], which is a container-based virtualization technology using LXC, and HTCondor [11] frequently used in HTC applications are interlocked with each other is introduced to present a method of implementing with ease the approach presented herein. Third, docker and native performance comparison using widely known benchmarking tools is presented and Monte-Carlo simulation results implemented using various programming languages in an environment where HTCondor and docker are interlocked with each other are presented.

This paper is organized as follows. The motivation and related work are described in Section 2. Then, detailed descriptions of the proposed approach and implementation are proposed in Section 3. Next, Section 4 shows the performance of cluster system implemented on HTCondor and docker. Finally, we conclude this paper in Section 5.

2. Background

2.1 Motivation

Most large-capacity cluster systems are shared by a multitude of individual and organizational users. As these users require way different service environments (OS, software package, configuration, etc.) in this setting, it is significantly challenging to meet their varying requirements in entirety. In particular, in-house codes developed by scientists on their own require specific OS and library versions.

To overcome these constraints, PLSI[1] provides users with compilers supported by different clusters, mathematical libraries, and installation paths on its website. Then, users need to find and access clusters where they can compile their own codes for execution [12].

This environment poses challenges to both administrators and users:

- *Challenges for administrators*: First of all, a lot of packages required by users should be installed on all computing nodes upon request. Given that most clusters have 500 or more computing nodes and numerous libraries and versions, this administration approach involves very daunting challenge. In addition, if a user-requested kernel version is different from OS already installed, it is difficult to fulfill such request. Because of this issue, service approach is commonly available without significant modification except for some bug fix and security enhancement in most clusters. This rigid service approach cannot support new technology development and compromises performance improvement following compiler version upgrade.
- *Challenges for users*: A user always has to confirm in advance if essential packages are available and, if not available, sends a request to administrator. In particular, if each cluster has different administrator in an environment where multiple clusters are interlocked with each other like in PLSI, a user has to request multiple administrators to install necessary software packages. Therefore, it is difficult for users to be provided with execution environments in real time as they desire. Because of these issues, scientists run their application programs using public cloud services despite performance degradation.

[1]To ensure that supercomputing resources are provided to researchers as efficiently as possible, a project named Partnership & Leadership for the Nationwide Supercomputing Infrastructure (PLSI) is carried out in Korea, aiming to establish a unified system of resources utilization by integrating supercomputing resources across the nation.

An approach to making available in real time a service environment desired by users without significant challenges for administrators is necessary.

2.2 Related Work

There has been good research activities in addressing the performance of virtualized resource in cloud computing environments [13], [14], [15], [16]. Walker [13] conducted the study on HPC in cloud by benchmarking Amazon EC2. Then, He et al. [14] extend to evaluating the technical capability of current public cloud computing platforms, and their suitability for running scientific applications, especially High Performance Computing (HPC) applications. Jackson et al. [15] represents the evaluation comparing conventional HPC platforms to Amazon EC2, using real applications representative of the workload at a typical supercomputing center. To evaluate the performance of real scientific workloads, it uses the NERSC benchmarking framework [17]. Iosup et al. [16] analyze the performance of cloud computing services for scientific computing workloads. Specifically, it focused on the real scientific computing workloads of Many-Task Computing (MTC) users. Despite these many activities, the use of virtualization has been traditionally laid off in most HPC facilities due to inherent performance overhead [3].

Recently, container-based virtualization systems (e.g., Linux VServer, OpenVZ, and Linux Containers) are investigated since it offer a lightweight virtualization layer, which promises a near-native performance [7], [8], [9]. In [7], the performance of three well known open source hypervisors, KVM, OpenVZ, and Xen was evaluated in the context of HPC. Their results showed that OpenVZ had the best performance for I/O throughput among them. Soltesz et al. [8] described a virtualization approach which is a synthesis of resource containers and security containers applied to general-purpose, time-shared operating systems. They conducted a network bandwidth benchmark using iperf and macro benchmarks for CPU and disk I/O intensive. From their results, I/O related benchmarks perform worse on Xen when compared to Linux VServer. Xavier et al. [9] conducted a number of experiments of container-based virtualized for HPC. Their results showed that the container-based virtualization system had better performance than traditional hypervisor-based virtualization. Furthermore, they described that LXC demonstrated to be the most suitable of the container-based system for HPC since the performance degradation can be offset by the easy of management.

Docker is a lightweight and powerful open source container virtualization technology combined with a work flow for building and containerizing applications [10]. It provides a toolset and unified API for managing kernel-level technologies, such as containers, cgroups, namespace and union file systems. Therefore, docker lets us quickly assemble applications from components and eliminates the friction

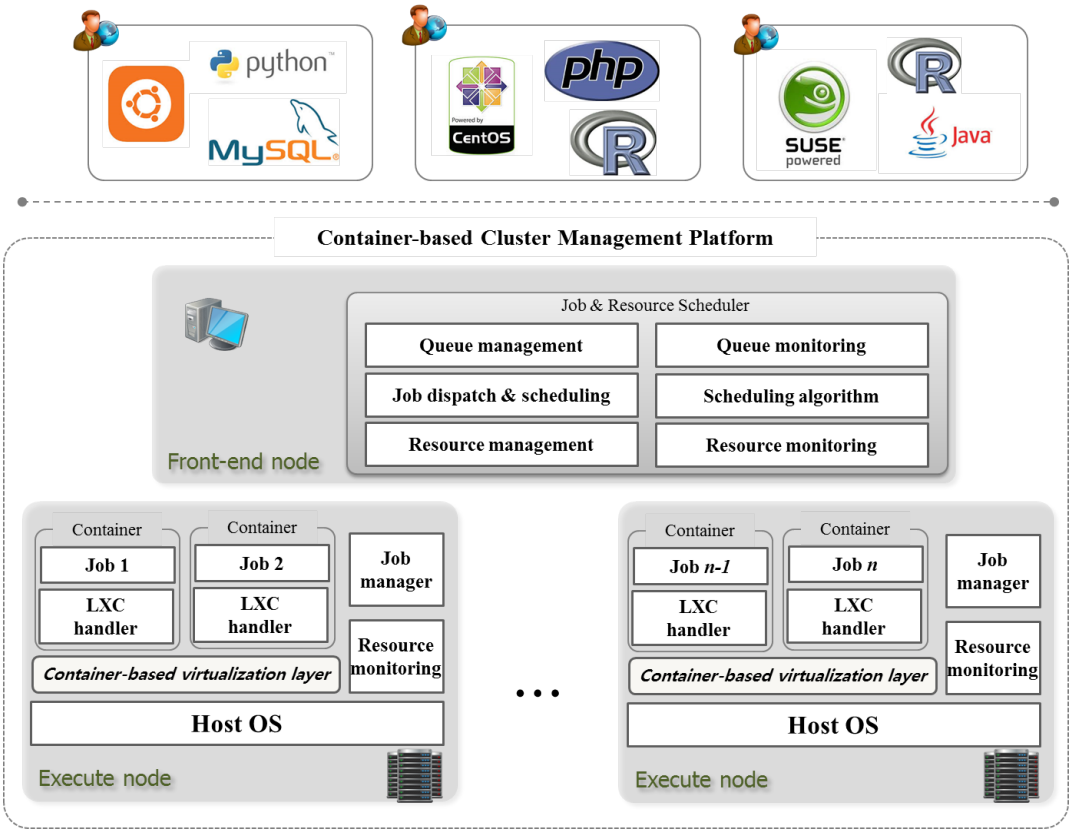


Fig. 1: Container-based cluster management platform architecture.

between development and production environments.

3. Container-based cluster management platform

3.1 Approach

Fig. 1 shows the proposed approach. In general, cluster systems supporting scientific workflows consist of a front-end node allocating computing resources in response to user requests and multiple execute nodes running actual tasks. In our approach, a user can submit tasks to the front-end node and multiple execute nodes regularly measure node resource status and report measured data to the front-end node. If resources are available, the front-end node dispatches tasks from a queue in accordance with FIFO, round-robin, priority-based preemptive or other scheduling algorithm and match-makes them with available resources for resource allocation. Upon completion of resource allocation, files needed for actual task execution are transferred to execute nodes. Execute nodes receive tasks to be executed from the front-end node and run application programs based on container-based virtualization layer. Execution results are transferred back to the front-node that submitted the tasks initially and forwarded to a user.

3.2 Implementation

In our implementation case, HTCondor was used as a job and resource scheduler and docker for container-based virtualization.

3.2.1 HTCondor daemons

In HTConodr pool, each machine can serve a variety of roles. Then, different daemons are running on the machine based on the role [11]. For the sake of simplicity, we focus on the six essential daemons in this paper.

- SCHEDD: This daemon takes responsibility for resource requests to the HTCondor pool. For this, it advertises the status of job queue and claims available resources to serve those requests.
- STARTD: This daemon takes responsibility for resource management of execute node. It advertises certain attributes about the execute node and is responsible for enforcing the policy that the resource owner configures.
- COLLECTOR: It collects all the information about the status of an HTCondor pool. All other demons periodically send ClassAd[2] updates to COLLECTOR.

[2]ClassAd is a scheme-free resource allocation language to represent arbitrary services and constraints on their allocation [18]

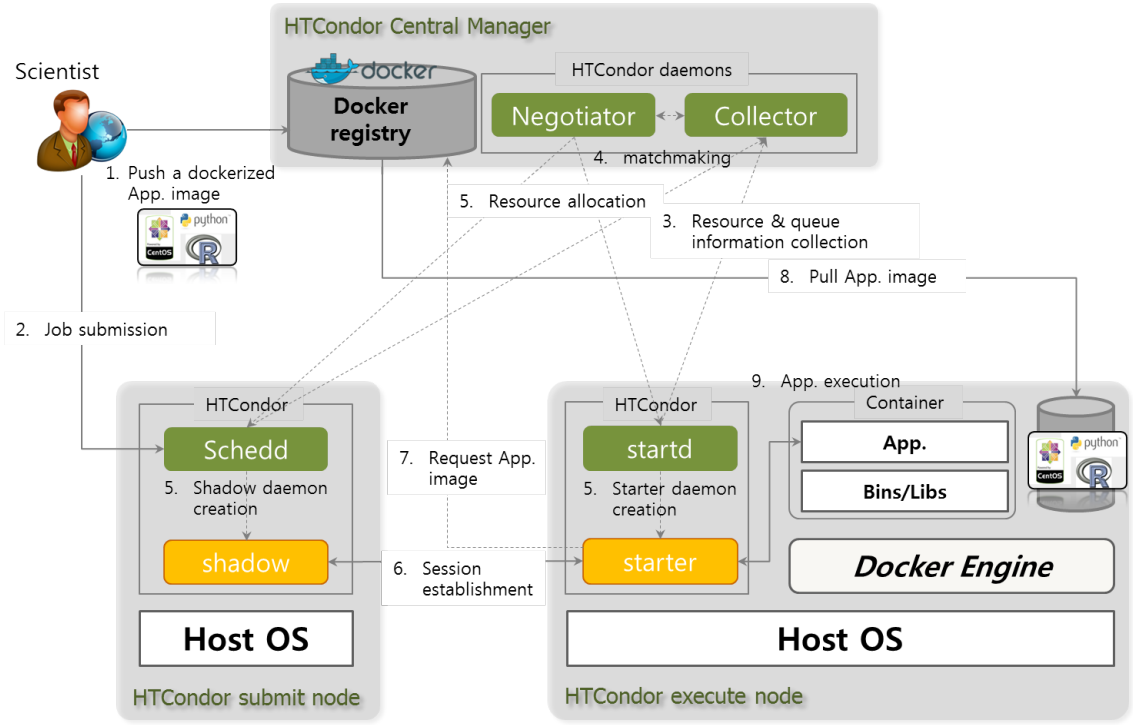


Fig. 2: Container-based cluster management platform implementation.

These ClassAds contain the state of the daemons, the resources, and the queue in the HTCondor pool.

- NEGOTIATOR: It is responsible for the match making in the HTCondor pool. Specifically, it contacts each SCHEDD that has waiting resource request and allocate available resources to those requests.
- SHADOW: It acts as the resource manager for the request. For example, Jobs that are linked for standard universe perform remote system call using this daemon. It runs on the machine where a job was submitted.
- STARTER: It sets up the execution environment and monitors the running job.

3.2.2 Implementation using HTCondor and Docker

Table 1: Hardware and software specifications.

Hardware spec.	
CPU	Intel(R) Xeon(R) CPU E5640@2.67GHz * 2ea
Memory	32GB
HDD	Western Digital WD 500GB 7200 RPM
Software spec.	
OS	CentOS release 6.5 Final
Job & resource scheduler	HTCondor 8.0.7
Container-based virtualization	Docker 1.1.2
Image management	Docker-registry server (dev) 0.8.0

Table 1 shows the hardware and software specifications of the system used herein. First of all, as shown in Fig. 2, a scientist creates a dockerized application image in advance for running his scientific workflow, pushes it to the docker registry, and creates a shell script file (*launch_docker.sh*) to launch dockerized application in execute nodes. Caution needs to be taken when a file to be used as argument must be forwarded into the container. To this end, the working directory of the host where execution file reside has to be mounted into the container, using *–v* option of the docker as follow:

Table 2: launch_docker.sh.

```
#!/bin/bash
sudo docker run -v $(pwd):/data docker_image /data/execute_file
```

To submit the shell script prepared in this manner to the HTCondor scheduler, a HTCondor job description file (Table. 3 is an example of job script) is created and submitted to the HTCondor SCHEDD. In the case of HTCondor, STARTD daemon reports the resource status of executable machines and SCHEDD daemon reports the job queue status to COLLECTOR daemon in ClassAd format at regular interval [11]. NEGOTIATOR match-makes job ClassAd and resource ClassAd based on data collected by COLLECTOR

to determine execute machine where tasks are to be executed. Once execute host to run tasks is determined via the matchmaking by NEGOTIATOR, SCHEDD and STARTD launch SHADOW and STARTER respectively, then a session is established between the two launched daemons. Through this session, *launch_docker.sh* file and argument files required for execution are transferred to the execute host and STARTER executes script file. At this time, STARTER daemon checks if dockerized application image is available in the local host where STARTER daemon is run and, if not available, pulls the dockerized image by the user in advance from the docker registry to execute dockerized application. Upon task completion, result files are transferred to submit node via SHADOW daemon and the user can confirm the results at submit node to which task was submitted.

Table 3: An example of HTCondor job script.

```
universe = vanilla
executable = launch_docker.sh
output = output file
transfer_input_files = execute_file
queue 100
```

Utilizing HTCondor and docker in this fashion brings about two advantages as follows. First, it is possible to implement with ease a container-based cluster management platform proposed. Secondly, such approach is applicable to a multi-cluster system like PLSI since HTCondor ensures that multiple cluster resources are utilized for a single scientific workflow.

4. Evaluation

This section analyzes the performance of docker, a container-based virtualization approach, and evaluates the performance of cluster system implemented on HTCondor and docker.

4.1 Micro-Benchmarks

Two widely known benchmark tools, unixbench and sysbench, were used to measure the performance of docker with the following measurement results:

- *unixbench*: The unixbench is a benchmark tool designed to measure overall system performance and provide a variety of benchmark results such as Whetstone, Drystone, file copy, pipe throughput, etc. Fig. 3 shows the index values of each item measured by unixbench tool. The index values of docker for all items except for the pipe-based context switching are found to be 90% or more when compared to the native performance. Pipe-based context switching test measures system performance by increasing integer through a pipe. The pipe-based context switching test is more like a real-world application [19]. In this value, docker shows 75% of the native performance. Fig. 4 shows the system benchmark index scores measured at 5699.5 and 5492.7 for native performance and docker shows its performance at 96% when compared to the native performance.
- *sysbench*: The sysbench is a tool using a variety of scenarios to measure CPU, memory, and File I/O performance results. Table 4 shows the benchmark results. First item shows CPU time taken to process 10,000 events of arithmetic operations using decimal fractions. It is confirmed that docker and native performances are identical. Second item shows sequential or random memory I/O performance in an assigned memory buffer and docker is measured to be better than native performance by 3% $\sim$ 5%[3]. Last item shows sequential or random file I/O performance results using 128 GB test files created in local disk. Form Table. 4, it can be

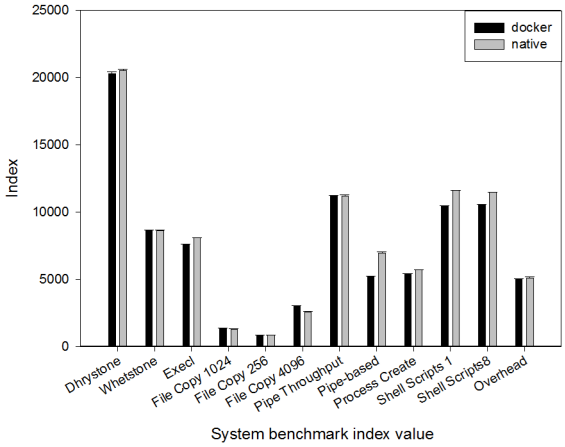


Fig. 3: unixbench benchmark results.

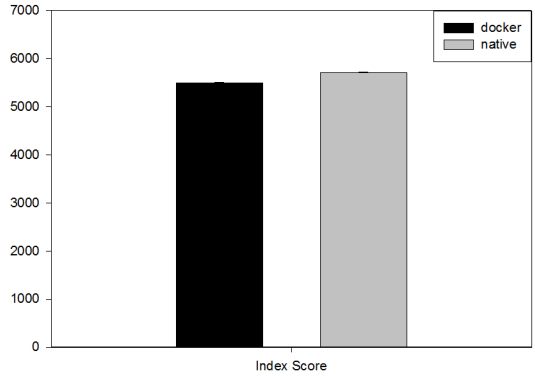


Fig. 4: The system benchmark index score.

[3]This benchmark results conflict with our intuitive understanding. The analysis of this problem should be explored in the near future.

confirmed that docker is found to show performance almost identical to native performance.

When the unixbench and sysbench measurements are compared in this section, docker is found to show no significant performance decline in comparison with native performance. Many studies conducted recently also show that container-based virtualization technology has a near-native performance [9].

Table 4: Sysbench benchmark results.

Test item	Option	Docker	Native
CPU	Total time	24.5 sec	24.5 sec
Memory	Sequence write	2.21 GB/sec	2.14 GB/sec
	Random write	3.41 GB/sec	3.25 GB/sec
	Sequence read	3.82 GB/sec	3.63 GB/sec
	Random read	3.67 GB/sec	3.50 GB/sec
File I/O	Sequence write	104.8 MB/sec	105.2 MB/sec
	Sequence read	91.7 MB/sec	91.7 MB/sec
	Combined R/W	1.5 MB/sec	1.5 MB/sec

4.2 Macro-Benchmarks

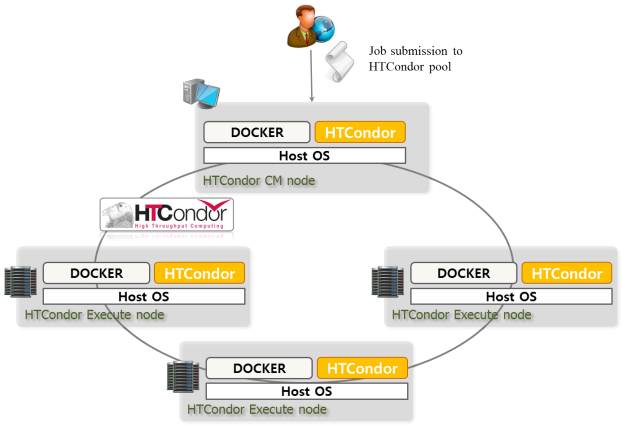


Fig. 5: The container-based cluster management system using docker and HTCondor.

This section presents the performance measurements of a cluster system implemented on docker and HTCondor for supporting scientific workflows. First of all, as illustrated in Fig. 5, HTCondor pool consisting of 1 central manager node and 3 execute nodes in accordance with the hard ware specification in Table 1 was configured. In addition, a docker registry was installed in the central manager node and docker clients in the execute nodes.

To measure the performance of the system implemented in this manner, a program to calculate *Pi* with a Monte-Carlo technique was implemented using C, JAVA, Python, and R. It picks points at random inside the square 10,000,000 times and checks to see if the point is inside the circle. Then, a simulation workflow to run the implemented program 100 times to reduce errors was created and submitted to HTCondor scheduler to compare execution time.

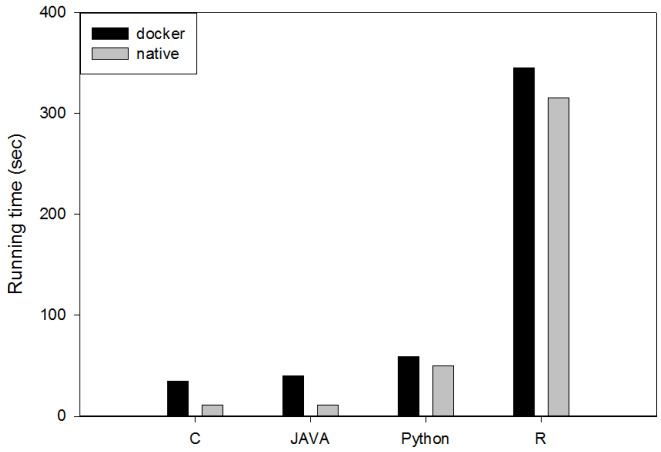


Fig. 6: Monte-Carlo simulation results (100 times).

Fig. 6 shows the Monte-Carlo simulation results with and without docker. As the figure confirms, difference in docker and native execution times is shown, depending on implemented languages. Namely, when executed with docker, execution time significantly increased 3.2 and 3.6 folds when compared to native performance in the cases of C and JAVA respectively while it increased only by 18% and 9% respectively in the cases of Python and R. The biggest factor underlying such difference is that in a simulation using docker, image loading time is different among implementation languages.

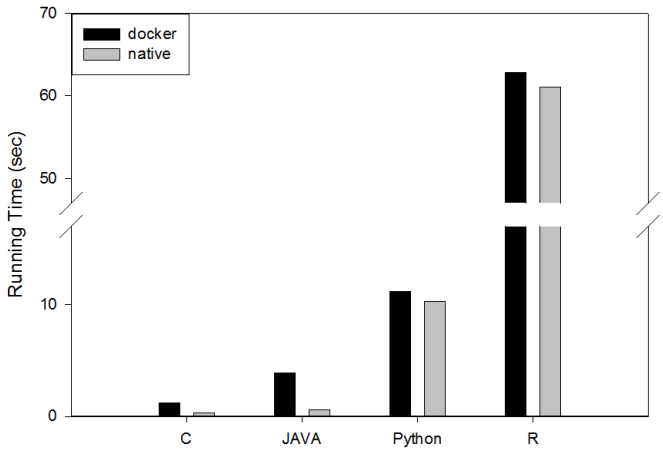


Fig. 7: Monte-Carlo simulation results (1 time).

Fig. 7 shows one simulation execution time of docker and native status. To exclude the possibility of image transfer

time, the dockerized image had been pulled in advance on each execute host. As the figure shows, execution time increased by 8% and 2% respectively in the cases of python and R while it rose very significantly 3.9 and 6.9 folds respectively in the cases of C and JAVA.

5. Conclusion

In this paper, we presented a container-based cluster management platform to provide a service environment desired by users. To provide a dynamic service environment, the virtualization technology is widely used. However, due to the inevitable overhead of hypervisor-based virtualization, container-based virtualization technologies such as Linux VServer, OpenVZ, and LXC are utilized recently. In this paper, we introduced an implementation case in which docker and HTCondor are interlocked. In addition, we conducted micro-benchmarks using unixbench and sysbench for CPU, memory, and file I/O performance and macro-benchmarks using Monte-Carlo simulation workflow for the performance of cluster system. Our results showed that docker had a near-native performance and image loading time is different among implementation languages.

References

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," *IEEE Computer*, vol. 40, no. 12, pp. 24–32, Dec 2007.

[3] K. Chen, J. Xin, and W. Zheng, "Virtualcluster: Customizing the cluster environment through virtual machines," in *Proc. of IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008*, vol. 2, Dec 2008, pp. 411–416.

[4] P. Ruth, P. McGachey, and D. Xu, "Viocluster: Virtualization for dynamic computational domains," in *Proc. of IEEE International Cluster Computing, 2005*, Sept 2005, pp. 1–10.

[5] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, "Dynamic provisioning of virtual organization clusters," in *Proc. of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Washington, DC, USA, 2009, pp. 364–371.

[6] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *Proc. of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, Washington, DC, USA, 2010, pp. 43–52.

[7] N. Regola and J.-C. Ducom, "Recommendations for virtualization technologies in high performance computing," in *Proc. of IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2010, pp. 409–416.

[8] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 275–287, Mar. 2007.

[9] M. Xavier, M. Neves, F. Rossi, T. Ferreto, T. Lange, and C. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Proc. of 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2013*, Feb 2013, pp. 233–240.

[10] "Docker." [Online]. Available: www.docker.com

[11] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[12] "PLSI: Partnership & leadership for the nationwide supercomputing infrastructure." [Online]. Available: http://www.plsi.or.kr

[13] E. Walker, "Benchmarking amazon EC2 for high-performance scientific computing," *LOGIN*, vol. 33, pp. 18–23, 2008.

[14] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in *Proc. of the 19th ACM International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2010, pp. 395–401.

[15] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Proc. of IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2010, pp. 159–168.

[16] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, 2011.

[17] "NERSC: National energy research scientific computing center." [Online]. Available: https://www.nersc.gov

[18] R. Raman, M. Livny, and M. Solomon, "Matchmaking: distributed resource management for high throughput computing," in *Proc. of The Seventh International Symposium on High Performance Distributed Computing, 1998.*, Jul 1998, pp. 140–146.

[19] "Unixbench." [Online]. Available: https://code.google.com/p/byte-unixbench/

Shareable, Persistent, In-Memory, Read-Only Data

Ralph Butler and Chrisila Pettey
Department of Computer Science
Box 48
Middle Tennessee State University
Murfreesboro, Tennessee, USA
ralph.butler and chrisila.pettey(@mtsu.edu)

Abstract – *Ensuring that multiple processes can access required data in a timely fashion is a critical aspect of successful parallel processing. The larger the data set, the more profoundly the overall run time is affected by reading the data and integrating it into the appropriate data structures and/or communicating the data between processes. Either each process must read and configure the data, or they must waste time in requesting and receiving data from another process. What is described in this paper is a Python-specific technique for having persistent, in-memory data that multiple processes can access without the overhead of loading or communicating the data.*

Keywords:. Client-Server, Shareable Data, In-Memory Data, Python.

1 Introduction

There are times that researchers are faced with a workflow that involves running multiple tests on the same data. This could be anything from performing comparative sequence analysis among many genes to data mining customer data. The general method for handling this type of experimentation is to generate a process for each test. The problem with this paradigm is that if the data set is large, then each process must spend the time necessary to read in the data and configure the internal data structures. If it takes an hour to read in the data and integrate it into the appropriate data structures needed for the algorithm, then each process will spend an hour reading prior to doing computation.

One possibility for alleviating this overhead would be to use SQLite [5]. SQLite allows processes to read and write directly to the database on disk without the overhead of communicating with a server. This model removes the intermediate server communication costs, nonetheless, disk I/O can frequently be too costly.

Another possibility would be to use Python remote objects [3]. Pyro allows the programmer to easily program processes that communicate over a network. Using Pyro we could have a server hold the configured data, and the client processes could access the data by using a remote object call. However, the Pyro model has the drawback of client/server communication costs.

Another possible solution we examined would be to use the shared data space capabilities of the Python multi-processing module [2]. As with Pyro, communication costs, specifically those with the proxy-wrapped objects, can frequently be prohibitive.

What we propose is a method for reading in the data and integrating it into the data structures once, then allowing multiple processes to access all of the data without the overhead of message passing or shared memory programming. Specifically, we propose a persistent, in-memory, read-only database. In this paper there will be several references to the "database". By this we merely mean our in-memory collection of data. It is critical to note that we do **not** use a DBMS.

2 How it's Done: Unix/Python Sleight of Hand

At the highest conceptual level, what we want is a database server that would remain running in memory so that clients could access the data without having to read in their own copy or request a copy from the server. At first glance, this might seem impossible. However, if you combine the power of the Unix *fork* command with the Python *exec* statement along with the Python feature of allowing you to dynamically add attributes to a module, the seemingly impossible becomes feasible. This procedure is explained more fully in section 2A while section 2B contains sample code segments.

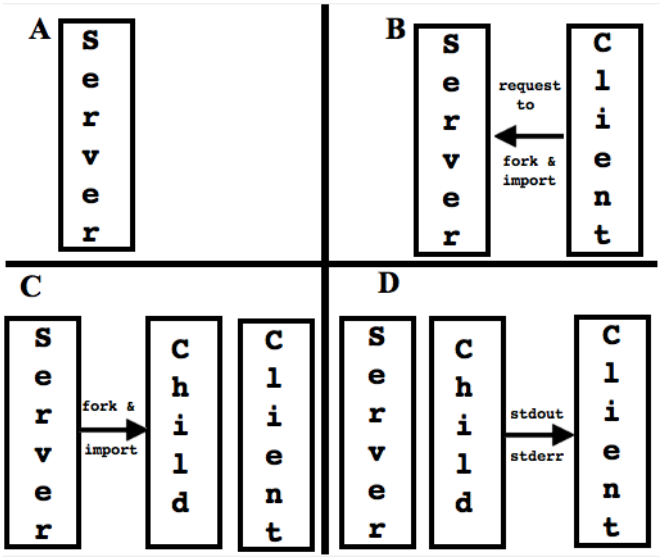


Figure 1. Setting up the persistent, in-memory, read-only database.

2A The Magic Explained

Figure 1 illustrates the four steps required to achieve a persistent, in-memory, read-only database. In part A of the figure, the database server is started. The server software must read in the data and integrate it into the appropriate data structures for use by the clients. In our experience, reading even large volumes of data is quite fast. However, the integration can be extremely time consuming. Once the in-memory database is created, the server starts up and listens for client messages on a socket. In part B of Figure 1, a client has connected to the socket and sent a message requesting that the server fork a new process in which the *existing* interpreter executes the client program. In part C, the server has forked a child process, set it's output to go to the socket the client is connected to, and used the Python *exec* command to execute a copy of the client code. Part D shows the forked child process executing and sending all output to the client.

As mentioned before, the key to making this process work is the combination of the Unix *fork* command, the Python *exec* command, and the Python feature of dynamically adding attributes to a module. When we use the first of these, the Unix *fork* command, we exploit our knowledge about what a fork actually does as opposed to what people often think it does. It is convenient to think of *fork* as copying the entire address space, and in earlier versions of Unix, that is exactly what happened. However, in more modern versions of Unix, e.g., Linux, a copy-on-write (COW) semantics is used – i.e., if a forked process does a write then that portion of the address space will be copied and changed, but if the forked process only reads, then there is no copy. This implies that if a large database is loaded into memory before the fork, none of it has to be copied unless one of the processes makes a change. In that case, only the portion that changes would be copied. Since the clients in our experiments usually perform computations using the data without making changes to it, the overhead of copying is eliminated. To illustrate this, consider Figure 2, which shows the relevant portions of a typical memory layout of each of the server, child, and client processes. What is of interest to us here is that the database would be in the *Dynamic Data Items* section of the memory layout. When the server forks a child process, the things that are available to the child include whatever is in the *Dynamic Data Items* section. This means the child gets access to the database without having to copy it or read it in.

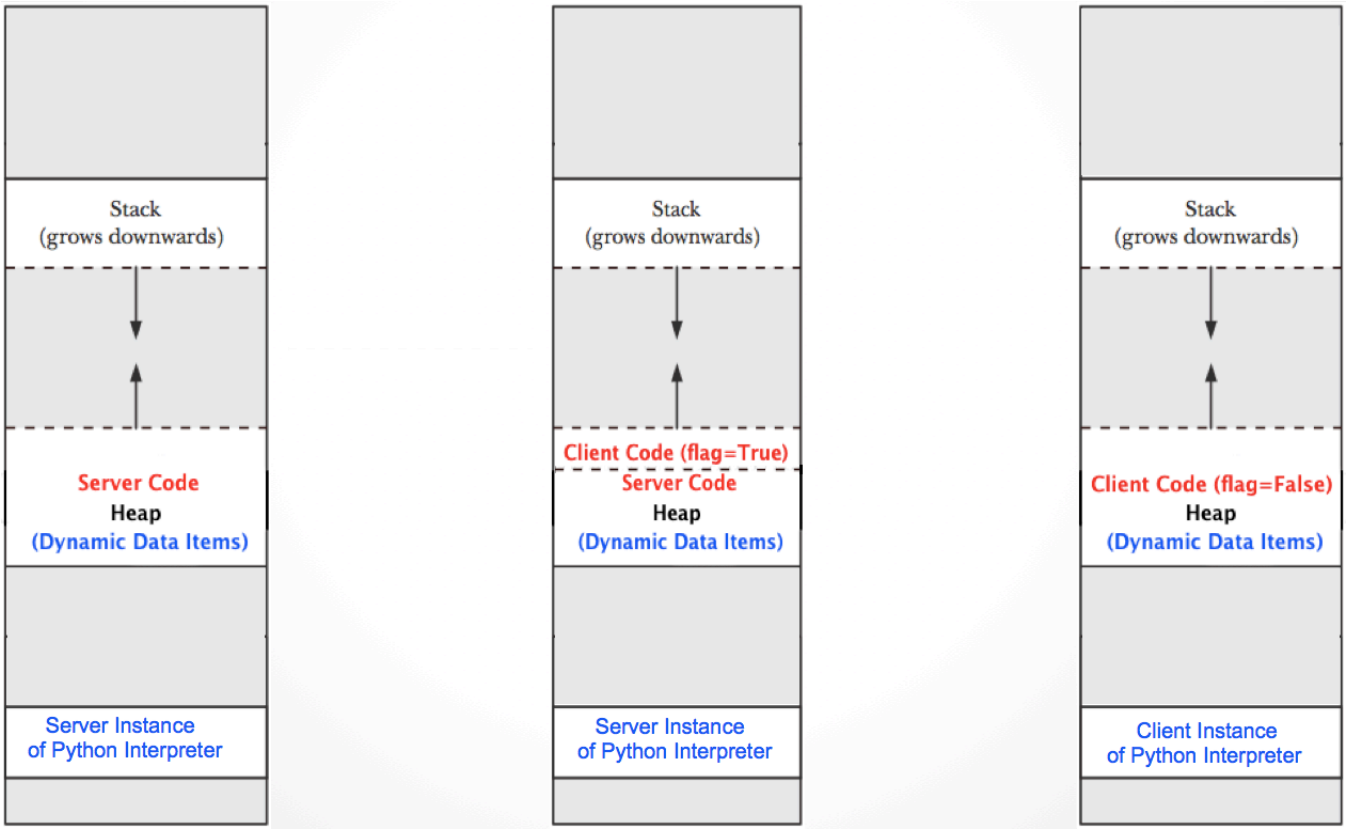


Figure 2. Shown left to right, portions of the memory layout of the server, child, and client processes.

The second key component of the process is the Python *exec* statement. Often when *fork* is used, it is accompanied by a Unix *exec*. However, the Unix *exec* command overlays the contents of the current process' address space with the new program and heap and stack space. If we were to use the Unix *exec*, then the database would be lost. Given that the *fork* command keeps the current instance of the interpreter running, we do not need to do a Unix *exec*. Instead we take advantage of the Python *exec*. The Python *exec* statement dynamically executes new Python code using the *current* instance of the Python interpreter. In this case, the *exec* statement is given the client code to execute.

At this point we have a child process that has access to the original copy of the database, can execute the client code, and can send the resulting printed output back to the client. However, the fact that the child has access to the database does not mean that it knows where the database is located in the address space. So this raises the question, "how can the child process obtain a reference/pointer that it can use to access the database?" The answer lies in taking advantage of the unusual Python functionality that treats an imported module as an object and allows attributes to be added to objects dynamically. Therefore, if the server dynamically adds a reference to the data as a named attribute of a module before the fork is performed, then the child process will have the reference to the named attribute of the module because it has access to everything the parent has.

A final observation that should be made is in regards to the client code that is executed by the child process. The client is only supposed to request that the server fork a child that will execute a copy of the client code, and then listen for output from the child. The child, on the other hand is supposed to execute some computation that the client wants executed. This is accomplished by having the server set an environment variable prior to calling the Python *exec*. Thus, the client code acts as a client if the environment variable is not set. Otherwise it performs the computation. If the copy of the client code in the child happens to execute an import for a module that the server has already imported, then the child process will simply gain a reference to the module that has already been loaded by the server.

2B Sample Code

Figures 3 – 6 contain example code that demonstrates the techniques described above. They contain extensive comments to make it clear how each step is accomplished.

```
# Import module that contains any code that needs to be shared between server and client
import sharedModule

# Read the data and integrate it into the data structures
bigdata = {}
bigdataFile = open('bigdata')
for line in bigdataFile:
    (key,val) = line.strip().split('\t')
    bigdata[key] = val
bigdataFile.close()

# Attach to the shared module a reference to the in-memory data structures
sharedModule.bigdata = bigdata

# Fork a new process to handle client connections
server = socketserver.ForkingTCPServer(("",1234),MSGHandler)
server.serve_forever()
```

Figure 3. Excerpt from the server code.

```
class MSGHandler(socketserver.BaseRequestHandler):
    def handle(self):
        while 1:

            # Receive a message from a connected client
            msg = self.request.recv(1000)
            # If client disconnected close socket
            if not msg:
                self.request.close()
                break

            # If valid request from client
            if msg.startswith('/'):

                # Create a child process
                rc = os.fork()

                # If I am the child process
                if rc == 0:

                    # Alter file descriptors so that stdout/stderr are routed to client
                    os.dup2(self.request.fileno(),sys.stdout.fileno())
                    os.dup2(self.request.fileno(),sys.stderr.fileno())

                    # Retrieve full path name of client code, and command line arguments
                    msg = msg.split()
                    fullpathname = msg[0]
                    sys.argv = msg

                    # Inform the child that it should execute the client code as an app
                    # rather than as an I/O client
                    os.environ['ENVFLAG'] = '1'

                    # Run the client code
                    exec(open(fullpathname).read())
                    break
```

Figure 4. Excerpt from server/child code.

```
# Import module that contains any code that needs to be shared between server and client
import sharedModule

# This is the line of code that helps this program to figure out if it is running as a
# client or a child process.  It must be executed before anything else.  The client will
# never return from this call.  (See Figure 6)
sharedModule.setup_client(__file__)

# From this point forward, only the child is executing the code.

# Obtain a reference to the data without having to read and integrate it.
bigdata = sharedModule.bigdata

# A trivial demo that the data is accessible
for key in bigdata:
    print(key,bigdata[key])
```

Figure 5. Excerpt from client code that is executed by both client and child.

```
# This code is in sharedModule

def setup_client(pgmName):

    # If I am a client process
    if 'ENVFLAG' not in os.environ:

        # Set up and connect to server
        from socket import socket, AF_INET, SOCK_STREAM, MSG_WAITALL
        sock = socket(AF_INET,SOCK_STREAM)
        sock.connect(('localhost',1234))  # note double parens

        # Send a message to the server telling it to execute this program with
        # its arguments
        msgToSend = ........ ## path_to_pgm and cmdline_arg_vals
        sock.sendall( msgToSend )

        # Loop receiving stdout/stderr and print them
        while True:
            msg = sock.recv(1000)

            #If no more stdout/stderr then break
            if not msg:
                break
            print msg

        # Close connection and terminate
        sock.close()
        sys.exit()
```

Figure 6. Excerpt from shared code.

3 Addressing Some Security Questions

This paper is not a treatise on security practices. However, we do admit that the concept of allowing clients to connect to a server does pose a potential security risk worth discussing.

It should be noted that, from the server point of view, the method proposed in this paper assumes that if the client can successfully authenticate, the server will trust messages sent by the client. This means the server will trust code that the client gives it to execute. Depending on the security technique used, this may place part of the security burden on the author of the client code. For some situations trusting client code might be unacceptable, but for our purposes we are willing to trust the client code as long as there is a valid authentication. We have two primary reasons for trusting the client code. First, the server is not running with root permissions. Second, as long as the client can authenticate, we assume that the code will not be malicious. We do admit that the code could contain bugs. Figure 7 shows an example of three clients and the forked children associated with them along with the server. Given this example and the possibility of a bug in client code, one question to answer is: if the code to be executed by the forked child for client 1 contains a bug, will that affect the other two forked children, the server, or the clients? It should be noted that the forked children have no more permissions than their associated clients. Therefore the other clients and children and server are insulated from the buggy behavior of a forked child to the same degree that totally separate processes would be protected from the behavior of each other.

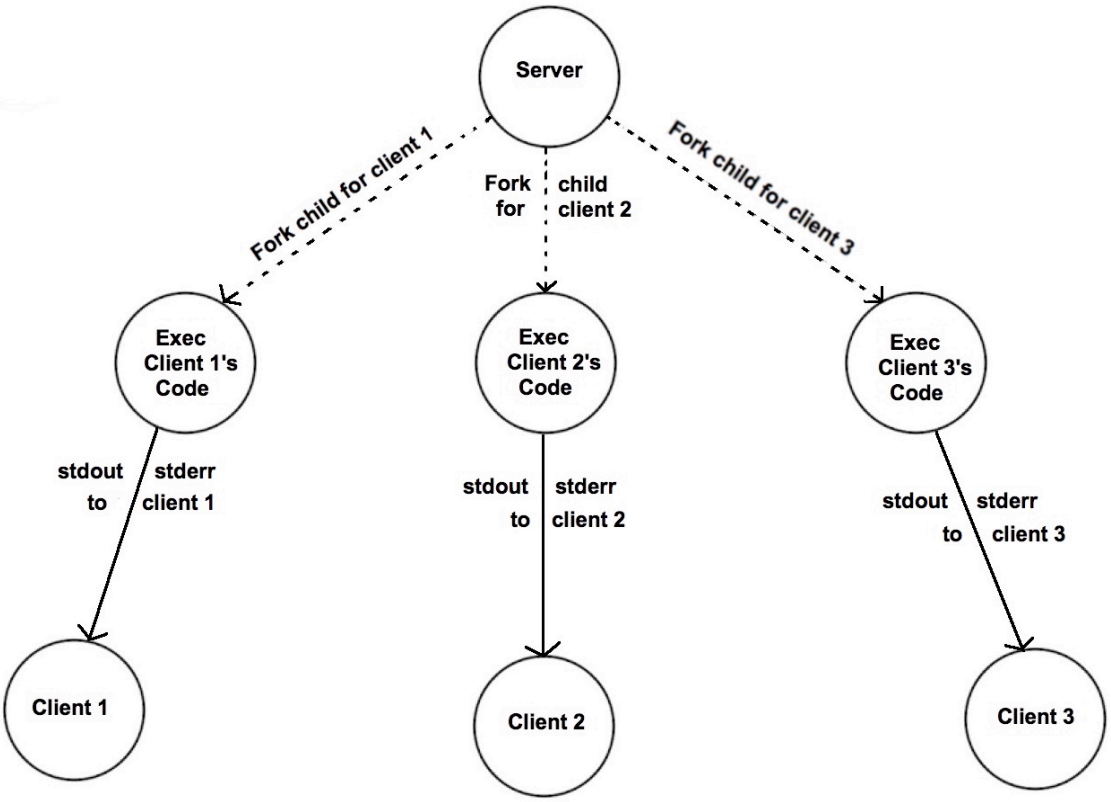


Figure 7. Example server with clients and children.

In our working environment we do not ordinarily require extraordinary security measures. For our purposes, we either work as individuals or we work in small teams that have shared file space. In this kind of situation, you are more assured of a secure environment if you use Unix domain sockets and make sure they are in a protected portion of the file system. This gives you the same level of protection as the file system.

If it is necessary to use INET domain sockets, then we have two models. Whenever possible, our first model is to execute only on machines that are behind a firewall. However, in the more risky case where we must connect

from outside the firewall, we provide additional security measures. One example measure is to have the client and server share a secret password/phrase. Then when the client connects to the server, the server issues a challenge to the client in the form of a large random number. The client then replies with a hash value derived by applying a digest algorithm (e.g. MD5 or SHA) to the concatenation of the challenge number and the password. Alternatives might include the use of SSL, secure socket layer, protocols.

4 Experiments

As an example of the power of the paradigm presented in this paper, this section contains the results of two experiments – one with a relatively small data set, and one with a moderately large data set. Both experiments were run on the Maple machine at Argonne National Laboratory (64 cores, 504 GB memory). The data sets are large files of genomic sequence information, and while this paper is not about bioinformatics, it is somewhat necessary to mention some terminology prior to describing the results of the experiments.

A PEG is a protein-encoding gene that could exist as either DNA (atg...tag) or as a protein sequence (M...V). While not particularly relevant, we do note that we typically work with the protein sequence. There are various things that can be done with PEGs [1]. One that we do is to try to determine their functions by performing some type of comparative analysis. For example, the function of the PEG *fig|100226.1.peg.3032* is *Imidazolonepropionase (EC 3.5.2.7)* which is part of the *Histidine Degradation* process. This analysis would frequently involve multiple genomes. A single genome is made up of multiple PEGs. For instance, the *Streptomyces coelicolor A3 (2)* genome contains 8154 PEGs with 3.5 MB of protein sequence in those PEGs. *Streptomyces coelicolor A3 (2)* is just one of the 25,442 prokaryotic genomes – totaling 137GB of data – that we retrieved from the SEED database [4].

In our first experiment the data set was approximately 1 GB, and consisted of a small, interesting, subset of the original 137 GB dataset. It took the server approximately 2.5 minutes to read in the raw data and configure the internal data structures. We then ran two clients. One client was run to determine the function of a single PEG (*fig|100226.1.peg.3032*) in the genome *Streptomyces coelicolor A3 (2)*. The other client was run to determine the functions of all 1600 PEGs that participate in known subsystems in the same genome. The first client ran in 0.6 seconds. The second client ran in 4.3 seconds. Both clients were able to run 2.5 minutes faster because they did not have to read and configure the data.

The first experiment was typical of the kind of work we do, and large enough for a nice proof of concept, but small enough for debugging purposes. The second experiment was to prove that the concept scales by using a much larger dataset. To accomplish this goal, the server read in all PEGs for all 25,442 genomes, then the clients visited every single PEG and examined the first few bytes of its protein sequence. For this experiment, it took the server 54 minutes to load the data set and integrate it into useful data structures, after which a client was able to execute in 2 minutes 14 seconds. The in-memory database model saved the clients 54 minutes each as well as conserving memory by alleviating the need for multiple copies of the database (137 GB raw data, and 44 GB memory size of the running process).

5 Conclusions

In this paper we offer a Python-specific technique for having shareable, in-memory, persistent data. The work presented in this paper arose primarily out of necessity. Testing, debugging, and actual experimental runs were time consuming due to the need to load and configure data for every run. By having the data be shareable and in-memory at all times, we can more quickly debug and test new client code, and we can more quickly run actual experiments. Admittedly, this solution is dependent on using Python, but since a significant portion of what we do uses Python, this is not a problem for us, and the savings in time is worth it. We would comment that the read-only part of the discussion is not strictly necessary. However, because of the copy-on-write rule, if changes are made to the data, then the page containing that data will be copied, and thus the performance will be slower. For our purposes, it is not necessary to change the data, so we are able to get the optimal benefits from the technique.

6 References

[1] Overbeek, R. *et al*, "The subsystems approach to genome annotation and its use in the project to annotate 1000 genomes," Nucleic Acids Res, 2005 Act 7; 33(17): 5691-702.

[2] Python multi-processing module https://docs.python.org/2/library/multiprocessing.html (last accessed 3/18/2015).

[3] Python Remote Objects. https://pypi.python.org/pypi/Pyro4 (last accessed 3/21/2015).

[4] The SEED. http://www.theseed.org/wiki/Home_of_the_SEED (last accessed 3/18/2015)

[5] SQLite. http://sqlite.org/ (last accessed 3/21/2015)

Locality Aware Work-Stealing based Scheduling in Hybrid CPU-GPU Clusters

A. Tarun Beri[1], **B. Sorav Bansal**[1], **and C. Subodh Kumar**[1]
[1]Indian Institute of Technology Delhi, New Delhi, India
{tarun,sbansal,subodh}@cse.iitd.ac.in

Abstract— *We study work-stealing based scheduling on a cluster of nodes with CPUs and GPUs. In particular, we evaluate locality aware scheduling in the context of distributed shared memory style programming, where the user is oblivious to data placement. Our runtime maintains a distributed map of data resident on various nodes and uses it to estimate the affinity of work to different nodes to guide scheduling. We propose heuristics for incorporating locality in the stealing decision and compare its performance with a locality oblivious scheduler. In particular, we explore two heuristics that focus on minimizing the cost of fetching data that is non-local. These heuristics respectively minimize the number of remote data transfer events, and the number of remote virtual memory pages fetched. Finally, we also study the impact of different placements of the initial input, like block cyclic, random and centralized, on the scheduler.*

We implement and evaluate these schedulers within Unicorn, a heterogenous framework that decomposes bulk synchronous computations over a cluster of nodes. Compared to a locality oblivious scheduler, the average observed overhead of our techniques is less than 8%. We show that even with this overhead, average performance gain is between 10.35% and 10.6% in LU decomposition of a one billion element matrix and between 12.74% and 14.55% in multiplication of two square matrices of one billion elements each on a 10-node cluster with 120 CPUs and 20 GPUs.

Keywords: Locality aware, Work stealing, Hybrid CPU-GPU clusters, Distributed computing

1. Introduction

Distributed shared memory based frameworks like Unicorn [1], Global Arrays [2] and X10 [3] allow programming styles simpler than message passing. User only accesses "memory" and the underlying data packing and communication is managed transparently by the runtime. Further, the computational tasks are also scheduled and load-balanced by the runtime. In this paper, we specifically aim to reduce the time for which computation is blocked behind network latency induced by remote memory access. We do this by scheduling data transfer early, overlapping it with other computation. Further, a number of heuristics are proposed to schedule computation close to data. In particular, an affinity is computed in a distributed fashion from partial information available at each node and input to a greedy scheduler. We develop these optimizations within Unicorn [1], a parallel programming framework for clusters populated with both CPUs and accelerators like GPUs.

Traditionally, locality-aware scheduling has centered on cache-affinity and focus has been on improving cache reuse. However, we argue that *node-affinity* is equally important for hybrid clusters, especially because significant time can be lost in fetching remote data and different devices are able to consume data at different rates. GPUs, being computationally more aggressive, cause more performance loss than CPUs if they need to wait for remote data transfer. For maintaining optimal data throughput (for achieving peak performance), avoiding GPU stalls on data is important and having à priori knowledge of data locality is useful, which Unicorn provides. In distributed environments, node affinity is often left to the user to specify [3]. In this paper, we instead explore inferring affinity based on the data accessed by the computation. We then inform the scheduling algorithm with this affinity value.

Unicorn [1] decomposes user's tasks into many independent *subtasks*. Subtasks are concurrent, and dependencies are only between tasks. Unicorn transparently schedules each spawned subtask to execute on an available computing device in the cluster. Information about data that a subtask seeks to use is specified early. Establishing subtask to node affinity based on this anticipated usage then can guide preferential scheduling of subtasks on nodes where most of its required data is resident. This, however, may not be optimal. Scheduling computation close to the largest resident data does not consider the time required to transfer the remaining data, which may offset the transfer time that is saved. For instance, the remote data may be highly dispersed among other nodes. Hence, we present other approaches, which instead of merely maximizing data locality focus on minimizing the transfer time of non-local data.

Note that all computations of affinity require an analysis of the set of addresses accessed by each subtask. This information, even if statically provided, is too large to store and process and reductions are necessary. This is the subject of this paper.

Many parallel and distributed programming frameworks, including Unicorn, employ randomized work-stealing for load balancing. Data locality of random work-stealing has been extensively studied for shared memory programming

and it has been largely found to be cache-unfriendly [4]. In this paper, we study the data locality of random work-stealing for computations distributed on hybrid CPU-GPU clusters and find that random work-stealing remains node-memory unfriendly as well. We have incorporated locality-aware strategies in work stealing as well to improve this.

As a base case, we start with trying to schedule a subtask on a node where where most of its input resides, maximizing *local data*. As shown in section 4, this technique reports an average performance gain of 12.7% over a non locality-aware work-stealer, while multiplying two square matrices of size $32768 * 32768$.

Next, we experiment with two strategies that instead of maximizing local data, target the time to fetch remote data. Our first strategy minimizes the number of remote data *transfer events* or requests. It is based on the observation that the incurred data fetch latency grows with data fragmentation and the number of data transfer requests. Accessing closely placed remote data is less expensive than accessing discontiguous remote data, which may cost additional latency. We observe an average performance gain of 14.55% with this heuristic over locality oblivious scheduling for matrix multiplication.

Our second strategy directly optimizes for the amount of *remote data*. It minimizes the total number of virtual memory pages to be fetched from remote nodes. Compared to *Unicorn's* locality oblivious scheduling, this strategy reports a performance gain of 12.74% for matrix multiplication.

These strategies behave differently for different experiments. For instance, our block LU factorization experiment respectively reports an average performance improvement (over *Unicorn's* locality oblivious scheduler) of 10.35% for *local data* heuristic, 10.35% for *transfer events* heuristic and 10.61% for *remote data* heuristic. We present more details on these techniques, including their overheads, in section 4.

Finally, we experiment with a few common placements of the initial input. A good scheduling strategy should adapt to the change in input data availability at various nodes in the cluster. We experiment with four different initial data placements - *centralized*, *row cyclic*, *column cyclic* and *random*. In the *centralized* scheme, all input data is placed on one node and all others contain no data initially. In the *row cyclic* scheme, blocks of rows of a pre-defined size are cycled through the cluster nodes in order. The same is done with the columns in the *column cyclic* scheme. In the *random* scheme, 2D blocks of a pre-defined size are kept on randomly selected nodes in the cluster. Results indicate that affinity based heuristics outperform the locality-oblivious scheduler.

The primary contributions of this paper are:

1) We dynamically estimate affinity of computation to nodes based on partial residency information. We show that efficient affinity based scheduling is possible even without full residency information.
2) We evaluate multiple heuristics to compute subtask-node affinity scores, considering data locality and fragmentation.
3) We study the benefit of incorporating affinity in work-stealing in heterogeneous environments. We also evaluate its robustness to different initial data placements.

2. Related Work

Data transfer overheads dominate many parallel applications. Locality aware scheduling is an effective way of reducing data transfers and the vital time spent in communication. Locality awareness has been extensively studied for both CPUs and GPUs. On CPUs, locality aware thread schedulers focus on improving data cache reuse. On GPUs, the focus is on enhancing accelerator kernels by scheduling GPU threads on streaming multiprocessors with better locality.

The locality issue in multi-threaded computations has received a lot of attention in the past. Acar et al. [4] minimize cache invalidations in random work stealing to develop a cache aware work-stealer for a single-core SMP. Similarly, Philbin et al. [5] describe an algorithm that determines a thread execution order that minimizes L2 cache misses. Tam et al. [6] and McGregor et al. [7] group threads based on cache locality for multi-threaded computations on multi-core processors. Vaswani et al. [8] present an analytical model to evaluate the effect of cache affinity on shared memory multi-processing. Intel TBB [9] enhances cache hits by creating an affinity between an iteration and a worker thread, which tends to execute the same iteration over and over. SLAW [10] is an adaptive locality aware scheduler for multi-core SMPs that uses programmer provided locality hints. Huang et al. [11] extends OpenMP [12] for specifying programmer controlled locality that minimizes the cost of data accesses.

Among the GPU locality aware schedulers, Nugteren et al. [13] reorder GPU threads automatically for improving memory coalescing and bank locality. Sugimoto et al. [14] improve cache locality for memory intensive texture-based volume rendering by dynamically varying the width and height of thread blocks so that memory access strides for warps are minimized. Unkule et al. [15] improve memory performance by automatically restructuring GPU kernels to better exploit data locality at the register and shared-memory levels. Lee et al. [16] analyze nested parallel computational patterns (like Map Reduce) for data locality and map them to the target GPU's multi-dimensional thread hierarchy.

FLAME [17] and MAGMA [18] are linear algebra systems that support dynamic scheduling on multiple GPUs. StarPU [19] is another framework with scheduling capabilities on multi-CPU plus multi-GPU architectures but most of its schedulers are locality oblivious. XKaapi [20], however, is a comprehensive runtime system with a locality-aware work-stealing scheduler for a single node application using both multi-core CPUs and many-core GPUs.

Some middleware have also been proposed for improving locality awareness of data intensive applications. SLAM [21]

is one such system that employs a distributed file system and a data-centric scheduler to reduce data transfers while reading from the file system. Similarly, VisDSI [22] proposes a locality aware I/O solution for data visualization.

In this paper, we extend ideas in these systems to build an efficient locality-aware work-stealing scheduler for a cluster of nodes with CPUs and GPUs. We study various scheduling heuristics with the aim of minimizing the time spent in data transfer. Our techniques are implemented in Unicorn parallel programming framework [1], which is briefly described in the next section. We start with a simple *node-affinity* based work-stealing scheduler that maximizes the use of local data and gradually build other heuristics, which focus on minimizing the remote access latency of non-local data.

3. Scheduling

We explore affinity based scheduling within the context of Unicorn [1], a unified parallel programming framework for CPUs and accelerators like GPUs. We first describe Unicorn's scheduling algorithm.

3.1 Unicorn

Unicorn models CPUs and accelerators as bulk synchronous computing devices that operate in logically distinct phases of local computation and synchronization. An application programmer in this framework provides coarse-grained interdependent tasks, and decomposes each into independent and concurrent computation modules called subtasks. These subtasks are autonomously scheduled by Unicorn runtime on the available computing devices. All network communications are layered over MPI [23].

For input and output, Unicorn tasks use an abstract entity called address space. A task may read/write any number of disjoint address spaces; each is logically shared by the subtasks but generally physically distributed across nodes in the cluster. Subtasks operate on an address space using transactional memory semantics, i.e., they check-out memory in a local view before working on it and check-in memory back to the global shared view once their computation is over. The local view visible to a subtask is user controlled and is called subtask's *memory subscription*.

For efficiency, an address space has a designated owner node that manages a distributed directory that maps addresses to locations in the cluster. As subtasks execute and write to an address, the corresponding directory entry is updated locally by the node executing the subtask. Local views are not invalidated until the end of the tasks following transactional semantics. At the end of the task, all directory changes are combined in batch mode by the address space owner. If the location of an address changes from one node to another, the former node is sent an update message. Thus, the owner node always knows the true locations of all addresses. Other nodes know true locations of addresses they have written to in a previous task. For other addresses they may not know the location and route their data transfer requests through the owner.

Unicorn uses subtask stealing to balance load. At the start of a task, Unicorn's scheduler equally divides the available subtasks among all devices in the cluster. Each device executes its assigned set and after it executes the last subtask in the set, it becomes ready to steal. It randomly selects a victim device which parts with a contiguous chunk of its outstanding subtasks and assigns them to the stealer. In case the victim has nothing to be stolen, a fail message is returned. The stealer then chooses another random victim. This continues till the task is completed.

3.2 Affinity based scheduling

In this paper, we extend Unicorn's scheduling to minimize wait for remote data by preferentially scheduling subtasks at nodes where their required data are likely to be already in the local view. To effect this we compute affinities of subtasks to nodes. Note that in Unicorn, a node's address space directory is guaranteed to contain true locations only for addresses in its local view. Hence, a node cannot compute the affinity of a subtask to other nodes. An alternative would be to force update the entire address space directory on every node (after every task) but a broadcast of this magnitude is impractical for performance reasons. Another possibility is to compute affinities of all subtasks centrally on the address space owner, whose directory entries are complete. However, a task typically uses many address spaces, each with a potentially different owner. Thus, even this alternative requires an expensive synchronization because a subtask's affinity must be based on the location of all of its data.

Hence, in this paper we explore heuristics using partial affinity information. We let each node examine the address ranges subscribed by all subtasks, only computing each subtask's affinity to itself. This is a much smaller list with size equal to the number of subtasks in one task. This can now be centrally gathered and processed. Unicorn's scheduler initially computes only the number of subtasks to assign to devices on each node, under the premise that all subtasks are created equal. We allow this step to proceed normally. Then we resort to a greedy approach to tie specific subtask IDs to devices on specific nodes. Nodes pick subtasks in a round-robin fashion. Each node picks the remaining subtask with the highest affinity score for it. If a node reaches its assigned count, it is skipped. Within the nodes, blocks of subtasks are assigned to its devices in the proportion the original scheduler determines. For our small experimental cluster of 10 nodes, the measured overhead of centrally collecting and remapping scheduler assignments is less than one millisecond. Significantly more time is spent in examining subtask subscriptions. This overhead is discussed in section 4.

Among the contributions of this paper is the computation of subtask-node affinity. A natural affinity score may be the size of data resident in the local view of the node. However,

this does not always afford the best speed-up. We also study other heuristics that consider the size of remote data instead. We study two variants of remote data affinity – one counts the number of data requests sent to remote nodes and the other that counts the number of address space pages fetched from remote nodes. Both these scores are found by first querying non-local regions from the address space directory and then combining these regions into as large contiguous chunks as possible. Unicorn's network layer allows generalized 1D and 2D data packing and we consider that in estimating the number of requests (i.e., the number of chunks) and the size of request (the total size of chunks divided by the page size).

In Unicorn, the victim assigns a number – call it s – of subtasks to the stealer on the basis of their relative rates of subtask execution (i.e., the number of subtasks executed per second before the steal operation). We retain that principle, but the actual subtasks assigned are the ones which have high affinity scores for the stealer but low scores for the victim. The victim chooses the s subtasks with the highest difference between their affinity to the stealer versus to the victim. As an aside, the stealer's affinity scores are not computed by the victim. We also do no include it with every steal request. Rather, we piggyback nodes' affinity scores on other data transfer. Since stealing happens near the end of the task, a stealer's affinity array is highly likely to reach all potential victims with negligible overhead. Nevertheless, if the affinity scores have not reached earlier, it comes with the request. In section 4, we report performance improvements with this scheme as compared to Unicorn's locality oblivious work stealer, which may allow a subtask with entire data on the victim's node to be stolen by a device on some other node with potentially no data, resulting in sub-optimal performance.

Note that evaluating all subtasks (on all nodes) for determination of affinity scores is a limitation of Unicorn's address spaces. As reported in section 4, this has non-negligible overhead. We explore evaluating fewer subtasks on all nodes. This compromises the accuracy of affinity scheduling but saves the time spent in computing affinity scores. In the next section, we analyse the impact of reducing the number of subtask subscriptions analyzed by each node.

4. Experiments and Analysis

In this section, we evaluate several node-affinity based work-stealing schedulers employing different Unicorn benchmarks. Our experiments were performed on a cluster of ten nodes, each equipped with two 6-core Intel Xeon X5650 2.67 GHz processors with 48 GB of memory. All the machines are powered with two Fermi generation Tesla M2070 GPU cards, each having 448 cores running at 1.15 GHz and 5 GB of GDDR5 memory. The machines run CentOS 6.2 with CUDA 5.5. For communication, we use Open MPI [24] 1.4.5 (over SSH) over a QDR InfiniBand [25] network.

We report our experiments on three benchmarks – image convolution, square matrix multiplication and block LU factorization. We have chosen these benchmarks as they have been well studied in the parallel domain and they make good candidates to stand for a wider range of applications. Image convolution is computationally moderate while being low on data transfer (part of which is overlapped with compute by *Unicorn's* pipeline). In contrast, matrix multiplication involves massive data transfers and is computationally expensive as well. LU factorization is an iterative interdependent series of tasks. Image convolution is also iterative: a sequence of filters is applied. The purpose of studying different kinds of applications is to understand their response to different locality-aware scheduling heuristics. All experimental results are based on three trials.

We first briefly discuss the implementation of these benchmarks over Unicorn and then study their responses to different scheduling heuristics. The results highlight that locality oblivious scheduling does not give optimal results for hybrid CPU-GPU clusters mainly because it does not account for the time spent in fetching of remote data. Our heuristics, on the other hand, attempt to minimize the time spent in data transfer.

In our image convolution experiment all color channels of a 24-bit RGB image of size $43008 * 32768$ are convolved with a $31 * 31$ filter. The input image is stored in a read-only address space (initially distributed randomly across the cluster nodes), logically divided into 336 blocks of size $2048 * 2048$. Each block is convolved using a separate subtask. However, because convolution at boundaries requires data from adjoining blocks, the input memory subscription of a subtask overlaps with other subtasks', potentially at all four boundaries. The output image is generated in a write-only address space. We convolve the filter 10 times with the image and each iteration is carried out in a different task. The experiment has a time complexity of $O(nm)$, n being the size of the image and m that of the filter.

In the matrix multiplication experiment two dense square matrices of size $2^{15} * 2^{15}$ each are multiplied to produce the result matrix. Each input matrix is stored in a read-only address space and the result matrix is stored in a write-only address space of the task. Both input matrices are initially distributed randomly across the cluster nodes. The output matrix is logically divided into $2048 * 2048$ sized blocks and computation of each block is assigned to a different subtask (which subscribes to all blocks in the corresponding row of the first input matrix and all blocks in the corresponding column of the second input matrix). The CPU subtask callback is implemented using a single-precision BLAS [26] function and the GPU callback uses the corresponding CUBLAS [27] function. The experiment has 256 subtasks and each runs a computation with time complexity $O(n^3)$.

In the in-place block LU decomposition [28] experiment,

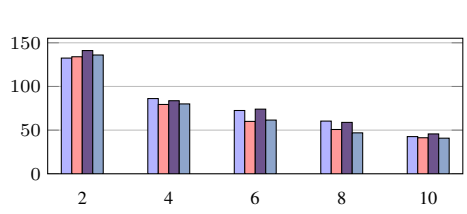
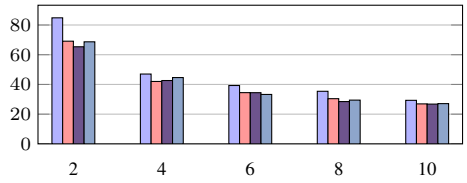


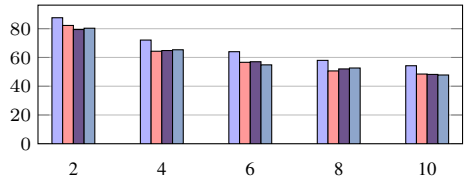
Image Convolution

Nodes	Data transferred (GB)					Data transfer events x 1000					Avg. subtask latency (secs)				
	2	4	6	8	10	2	4	6	8	10	2	4	6	8	10
Unicorn	18.2	30.5	33	36.2	35.7	6.5	14.9	18.0	20.3	21.8	6.6	3.6	2.9	2.5	2.4
Local Data	0.6	0.9	1.1	1.3	1.3	5.9	13.4	15.9	16.9	18.6	7.4	3.8	2.7	2.3	2.2
Transfer Events	9.1	21.2	33.6	37.9	38.4	2.5	7.7	12.7	16.4	19.3	7.4	3.9	2.9	2.5	2.3
Remote Data	0.6	1.0	1.2	1.2	1.3	6.5	13.1	15.0	18.0	18.9	7.5	3.8	2.7	2.3	2.2



Matrix Multiplication

Nodes	Data transferred (GB)					Data Transfer Events x 1000					Avg. subtask latency (secs)				
	2	4	6	8	10	2	4	6	8	10	2	4	6	8	10
Unicorn	8.0	23.9	36.8	49.1	55.9	0.3	1.0	1.5	2.1	2.6	7.3	7.2	7.4	7.3	6.8
Local Data	7.7	19.6	26.0	34.3	37.6	0.3	0.8	1.1	1.6	1.9	8.1	7.7	7.4	7.0	6.0
Transfer Events	7.2	21.2	28.2	34.7	38.4	0.2	0.8	1.3	1.7	2.0	8.2	8.2	7.8	6.7	6.1
Remote Data	7.3	19.9	26.5	33.8	38.9	0.3	0.8	1.3	1.6	1.9	8.2	7.8	7.7	7.0	6.5



LU Decomposition

Nodes	Data transferred (GB)					Data transfer events x 1000					Avg. subtask latency (secs)				
	2	4	6	8	10	2	4	6	8	10	2	4	6	8	10
Unicorn	15.2	28.6	35.6	41.1	44.7	1.0	1.8	2.3	2.6	2.8	4.3	3.9	3.7	3.5	3.6
Local Data	6.3	16.4	22.2	25.6	28.2	0.4	1.1	1.4	1.6	1.8	4.6	4.4	4.1	3.9	3.7
Transfer Events	5.9	16.1	22.0	25.8	28.8	0.4	1.0	1.4	1.7	1.9	4.6	4.3	3.9	3.8	3.8
Remote Data	6.3	16.1	21.1	25.9	27.8	0.4	1.0	1.4	1.6	1.8	4.5	4.3	4.0	3.9	3.8

Fig. 1: Locality aware scheduling - Graphs plot Execution Time (secs) versus Nodes

the input matrix ($2^{15} * 2^{15}$) is kept in a read-write address space (initially distributed randomly across the cluster nodes) and is logically divided into $2048 * 2048$ sized blocks. The matrix is solved top-down for each of the 16 diagonal blocks. For a matrix divided into $n * n$ blocks, solving for each diagonal block (i, j) involves three tasks – LU decomposition of the diagonal block (i, j), propagation of its results to other blocks in its row $(i, j + 1...n)$ and column $(i + 1...n, j)$ and propagation of these results to other blocks underneath $(i+1...n, j+1...n)$. The first of these three tasks is executed sequentially while the other two are executed in parallel. Time complexities of these tasks are $O(n)$, $O(n^2)$ and $O(n^3)$, respectively. One task is spawned per diagonal block which, in turn, executes 3 tasks within, making a total of $3n-2$ tasks (where n is the number of diagonal blocks). The parallelism within tasks (i.e., the number of subtasks) reduces as we move down the matrix because the number of blocks to be solved in parallel decreases. The CPU subtask implementation uses single-precision BLAS functions while the GPU implementation employs the corresponding CUBLAS routines.

All our experiments are written with no particular spatial ordering of subtasks. Thus, the adjacent subtasks of these experiments do not necessarily execute on adjacent address space regions. This makes a better case for studying the effectiveness of affinity heuristics. The Image convolution experiment has the smallest memory footprint with a total input size of 3.94 GB, followed by LU decomposition with 4 GB input. Matrix Multiplication has two input address spaces of 4 GB each making the total input size 8 GB.

For the three benchmarks, Figure 1 plots the performance of *Unicorn's* locality-oblivious scheduler as well as the performance of the *local data* based affinity and compares these to *remote data* based affinity (*transfer events* and *remote data*). The figure also records the cluster-wide data transfers and subtask latency incurred in these experiments.

Results show that all our heuristics perform better than *Unicorn's* locality oblivious scheduler at nearly all data points. For image convolution experiment, a maximum gain of 22.3% (over *Unicorn's* scheduler) is observed with *remote data* heuristic for the eight node case. This is attributed to a massive data transfer reduction from 36.2 GB (for Unicorn) to 1.18 GB. A gain of similar magnitude is not reflected in execution time because much of the transfer latency is hidden behind other computation for the experiment. We find that Unicorn's work stealing results in most of the subtasks actually being executed by GPUs (being more powerful than CPUs for a SIMD computation like image convolution) where computation of a subtask is overlapped with the data transfer of the next. Even reducing the overall data transfer by more than 96% does not make this compute bound pipeline of GPU subtasks any faster.

For the matrix multiplication experiment, we observe the maximum gain of 19.56% for the eight node case with the *transfer events* heuristic. This result is attributed to a 29.25% reduction in data transfer, a 22% reduction in data transfer events and a 0.6 sec gain in average subtask latency. Note that this is a communication bound experiment and the percentage reduction in data transfer has resulted in a similar gain in performance after accounting for the 9.6% overhead

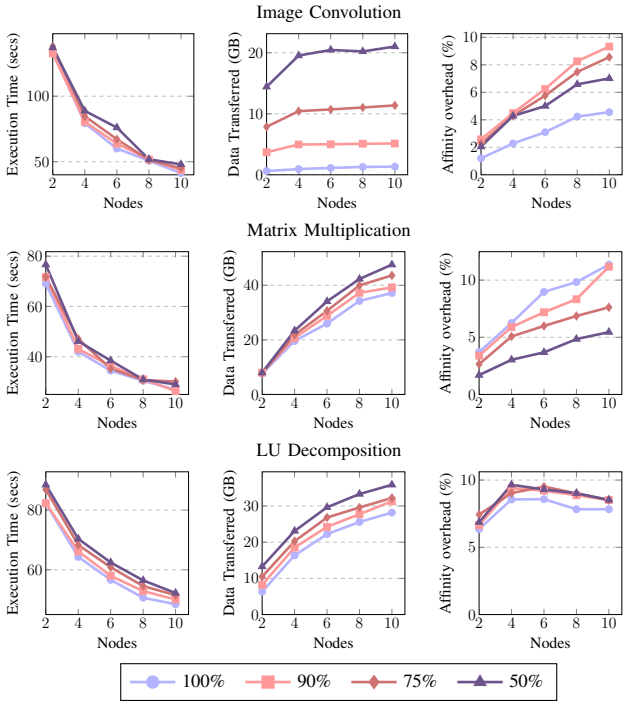


Fig. 2: Incremental affinity subtask reduction

in affinity computation.

For the LU decomposition experiment, the maximum gain of 12.67% is achieved by the *local data* heuristic for the eight node case. This heuristic resulted in a 37.6% reduction in total data transfer and has a reported overhead of 8%. The experiment has moderate performance gains as compared to data transfer savings as it is an iterative experiment, with a mix of compute and communication bound tasks per iteration.

All our heuristics perform fairly closely to each other. *Local data* heuristic reports the lowest execution time for experiments with small memory footprint (image convolution and LU decomposition) when the number of nodes involved is not more than six. When the number of nodes increases to eight or ten, the *remote data* heuristic outperforms others. As far as reduction in data transfer is concerned (in comparison to Unicorn), we expect iterative experiments like image convolution and LU decomposition to do better than the non-iterative matrix multiplication like experiments as the benefits of reducing data transfers are realized every iteration. We find that *transfer events* heuristic is not able to bring down data transfers in image convolution by the same margin as other heuristics, resulting in its poor performance as compared to the other two. This shows that an indirect heuristic that optimizes data transfer events in an attempt to reduce actual data transfers may not be as suitable as compared to other heuristics that directly target maximizing local or minimizing remote data.

Despite the performance improvements with our heuristics, we observe significant overhead in affinity determination

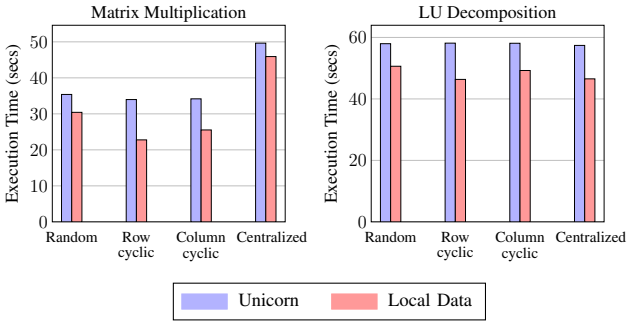


Fig. 3: Impact of initial data distribution pattern

(Figure 2). This is because our affinity determination algorithm evaluates input memory subscriptions of all subtasks of the application task on all nodes. In order to reduce this overhead, we reduce the number of subtasks evaluated per node (for affinity determination) and study the response of *local data* heuristic to this change (Figure 2). The figure plots execution time, data transferred and affinity determination overhead when the number of subtasks evaluated (for affinity computation) per node are gradually reduced from 100% to 50%. Results show that for all experiments, reduction in the number of evaluated subtasks also reduces the effectiveness of the heuristics. On an average, image convolution becomes 11.85% slower when the number of analyzed subtasks at each node reduces from 100% to 50%. Similarly, the impact on matrix multiplication is 8.33% and on LU decomposition is 9.31%. In general, the magnitude of loss in performance despite gains in the affinity task's overhead makes this overall less profitable.

As stated earlier, all our tasks have a random block distribution of the initial data in their address spaces. A good affinity scheduler should be agnostic to the changes in initial data availability pattern. We study *local data* heuristic with different initial placements of data and compare its performance to that of Unicorn's scheduler. Figure 3 evaluates matrix multiplication and LU decomposition (for eight node case) for various schemes like *centralized* (entire address space on one cluster node), *row cyclic* (rows of $2048 * 2048$ blocks placed in sequence on all cluster nodes), *column cyclic* (columns of $2048 * 2048$ blocks placed in sequence on all cluster nodes) and the default *block random* ($2048 * 2048$ blocks placed randomly on any cluster node). Results show that our runtime maintains performance despite the changes in data availability pattern. At all data points, our heuristic results in better performance than Unicorn's scheduler. Further, for the matrix multiplication experiment more favorable distributions like *row cyclic* and *column cyclic* achieve proportionately higher gains with our heuristic in comparison to other less favorable distributions.

Lastly, we study the effectiveness of our steal policy post the initial distribution of subtasks. In the absence of external factors, a good initial subtask distribution is effective in balancing load for a large fraction of the task. However,

towards the end of a task a few devices finish early and try to steal work from others. In this case, we let the victim assign those subtasks to the stealer that are high on affinity scores for the stealer but low on affinity scores for the victim. As compared to Unicorn's affinity oblivious steal, this scheme reduces data transfers by around 2.5% on an average for all three experiments when affinity scores are computed with *remote data* heuristic. This translates into performance improvements up to 3.0%. When affinity scores are computed with *local data* heuristic, however, we observe mostly flat response with our scheme. For image convolution experiment, it reports around 2.5% performance gain as compared to Unicorn's stealing while a 4.3% degradation is observed with matrix multiplication.

5. Conclusions and Future Work

We present a study of locality aware work stealing in the context of distributed shared memory programming on hybrid CPU-GPU clusters. In particular, we augment *Unicorn's* work-stealing scheduler with data locality and study its characteristics. We evaluate two other heuristics that attempt to reduce the time spent in transfer of non-local data across the cluster. These heuristics, respectively minimize the number of data transfer events and the size of remote memory fetched. These heuristics are based on each node computing affinity based only on the data it has. The results demonstrate reasonable performance improvements with these heuristics despite non-trivial overhead in address subscription analysis. Given that Unicorn hides some network latency behind other computation, sometime the gain in overall application speed does not reflect the significant data transfer reduction, but in a loaded network this gain can be useful. We believe that with better analysis of subscription, it is possible to improve affinity scores further.

All the benchmarks presented in this paper are regular. However, we have experimented with a few irregular applications like page rank and initial results are promising, even if incomplete at this time. We believe that experimenting with more irregular applications and using larger clusters may promote better understanding of these heuristics.

References

[1] Beri, Bansal, and Kumar, "A scheduling and runtime framework for a cluster of heterogeneous machines with multiple accelerators," in *Proceedings of the 2015 IEEE 29th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '15, 2015.

[2] Nieplocha *et al.*, "Advances, applications and perf. of the global arrays shared memory programming toolkit," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, May 2006.

[3] Charles *et al.*, "X10: An object-oriented approach to non-uniform cluster computing," in *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*.

[4] Acar, Blelloch, and Blumofe, "The data locality of work stealing," in *Proceedings of the Twelfth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '00.

[5] Philbin *et al.*, "Thread scheduling for cache locality," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS VII, 1996, pp. 60–71.

[6] Tam, Azimi, and Stumm, "Thread clustering: Sharing-aware scheduling on smp-cmp-smt multiprocessors," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07, 2007, pp. 47–58.

[7] McGregor, Antonopoulos, and Nikolopoulos, "Scheduling algorithms for effective thread pairing on hybrid multiprocessors," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS '05.

[8] Vaswani and Zahorjan, "The implications of cache affinity on processor scheduling for multiprogrammed, shared memory multiprocessors," in *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '91.

[9] "Intel Threading Building Blocks," http://www.threadingbuildingblocks.org/.

[10] Guo *et al.*, "Slaw: A scalable localityaware adaptive work-stealing scheduler," in *In 24th IEEE International Symposium on Parallel and Distributed Processing*, ser. IPDPS '10.

[11] Huang *et al.*, "Enabling locality-aware computations in openmp," *Sci. Program.*, vol. 18, no. 3-4, Aug. 2010.

[12] Dagum and Menon, "OpenMP: An industry-standard API for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan. 1998.

[13] Nugteren, Braak, and Corporaal, "A study of the potential of locality-aware thread scheduling for gpus," in *Euro-Par 2014: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, 2014, vol. 8806.

[14] SUGIMOTO, INOb, and HAGIHARA, "Improving cache locality for gpu-based volume rendering."

[15] Unkule, Shaltz, and Qasem, "Automatic restructuring of gpu kernels for exploiting inter-thread data locality," in *Proceedings of the 21st International Conference on Compiler Construction*, ser. CC'12, 2012.

[16] Lee *et al.*, "Locality-aware mapping of nested parallel patterns on gpus," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014.

[17] Quintana-Ortí *et al.*, "Solving dense linear systems on platforms with multiple hardware accelerators," *SIGPLAN Not.*, vol. 44.

[18] "Magma 1.4.1," http://icl.cs.utk.edu/magma/, 2013.

[19] Augonnet *et al.*, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 2, pp. 187–198, Feb. 2011.

[20] Gautier *et al.*, "Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures," in *IPDPS '13*, ser. IPDPS '13, 2013, pp. 1299–1308.

[21] Yin *et al.*, "Slam: Scalable locality-aware middleware for i/o in scientific analysis and visualization," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14, 2014.

[22] Ng *et al.*, "Visdsi: Locality aware i/o solution for large scale data visualization," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, Dec 2013.

[23] Gropp *et al.*, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, no. 6, pp. 789–828, Sep. 1996.

[24] Gabriel *et al.*, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Euro. PVM/MPI Users Group Meeting*, 2004, pp. 97–104.

[25] InfiniBand Trade Association, InfiniBand Architecture Specification, Release 1.1, 2002.

[26] Dongarra *et al.*, "An extended set of FORTRAN basic linear algebra subprograms," *ACM Trans. Math. Softw.*, vol. 14, no. 1, Mar. 1988.

[27] "The NVIDIA CUDA basic linear algebra subroutines," https://developer.nvidia.com/cuBLAS.

[28] Demmel, Higham, and Schreiber, "Block LU factorization," http://www.netlib.org/utk/papers/factor/node7.html, 1995.

Distributed Scheduling in Wireless Mesh Networks using Smart Antenna Techniques

Alissar Sabbah, Abed Ellatif Samhat,

Lebanese University - Faculty of Engineering, Rafic Hariri Campus, Hadath, Lebanon
Email: alice.sabbah@hotmail.com;samhat@ul.edu.lb,

***Abstract*— We investigate in this paper the distributed scheduling in Wimax mesh networks. Unlike scheduling based on conventional omni-directional antenna that generates interference, we propose an algorithm for coordinated distributed scheduling that improves the system performance by taking into account the efficiency of smart antennas to maximize network throughput by maximizing the number of concurrent transmissions without increasing interference. The implementation is achieved using the three-way handshake mechanism with few modifications. The simulations results in terms of simultaneously active links, i.e. network throughput, show great benefits of the proposed scheme in comparison with the conventional one.**

***Index Terms*— Wireless mesh network, Wimax mesh mode, coordinated distributed scheduling, smart- antennas.**

I. INTRODUCTION

Wireless Mesh Networks (WMNs) have emerged as a key technology for next-generation wireless networking to provide high-bandwidth network coverage. WMNs are built by a set of spatially distributed nodes interconnected by wireless links. Nodes can send data to other nodes by using a direct link or through other nodes until it reaches its destination in a multi-hop way. WMN is Mesh networking that can be implemented over wireless networks such as WiFi (Wireless Fidelity), WiMAX (Worldwide interoperability for Microwave Access), etc. Therefore WMNs are undergoing rapid progress and inspiring numerous applications.

Based on the IEEE 802.16 standard [1], the WiMAX MAC (Medium Access Control) protocol is designed in two modes PMP (Point-to-Multi-Point) mode and Mesh mode. The traffic in PMP mode occurs only between the Base Station (BS) and the Subscriber Stations (SS). But in mesh mode, the traffic can occur between SSs or between BSs. There is no need to have direct link from SS to the BS of the mesh network. A node can choose the link with the best quality to transmit data, and with an intelligent routing protocol, the traffic can be routed to reach destination.

The IEEE 802.16 Mesh mode employs the Orthogonal Frequency Division Multiplexing (OFDM) scheme in the physical layer. For channel access among the BS and the SS nodes, Time Division Multiple Access (TDMA) is used. Each SS requests its transmission resources and there is no real distinction between UL and DL. Each SS communicates with its neighbors (transmitting, receiving or idle status). This can be seen as a kind of Time Division Duplex (TDD). Two kinds of scheduling algorithms exist in the 802.16 mesh mode: centralized and distributed. In centralized scheduling, the scheduled transmissions for the SSs are defined by a central unit (the BS) but in the distributed algorithm, all nodes contribute to the scheduling. The distributed scheduling could be coordinated or uncoordinated. In coordinated distributed scheduling each SS has a list of its neighbours and competes for transmission opportunities with them in a coordinated manner to avoid collisions. Uncoordinated scheduling is a try and see mode where collisions can happen.

The IEEE 802.16 standard provides signalling messages for both centralised and distributed scheduling, but leaves the scheduling algorithms open for the vendor's implementation. Centralized scheduling algorithms have been widely studied [2][3][4][5] and few papers investigated distributed scheduling. In [6] an analytical model for the distributed scheduler of the mesh mode is proposed to evaluate the scheduler performance under various condidtions. Based on the model of [6] the authors in [7] analyse the transmission timing of signalling messages and improve the performance of the distributed scheduler by dynamic adaptation of the hold off exponent. [8] investigates the performance of

coordinated distributed under realistic non-quasi-interference model. The paper shows that substantial amount of collisions may exist even with scheduling in 3-hop extended neighbourhood and suggest that, by keeping a balance between the holdoff interval and the reception collision ratio; the optimal overall scheduling latency may be achieved. In [9], the authors present a fair bandwidth allocation algorithm (FEBA) for service differentiation in IEEE 802.16 mesh networks operated in a distributed coordinated scheduling mode. FEBA negotiates bandwidth among neighbours to assign a fair share proportional to a specified weight to each end-to-end traffic flow.

The existing work investigated the scheduling using conventional omni-directional antenna and the interference remains a major issue in this context. In this paper, we propose an algorithm for distributed scheduling that optimizes the system performance by taking into account the efficiency of smart antenna to maximize network throughput by maximizing the number of concurrent transmissions without increasing interference. A smart (directional) antenna offers a longer transmission range and lower power consumption by forming one or multiple beams towards intended receivers only, thus reducing interference.

The rest of this paper is organized as follows. Section II presents the frame structure and the mechanisms related to the distributed scheduling. In section III, the smart antenna technique is explained and the proposed algorithm based on this technique is given. The performance evaluation of the proposed algorithm is done through simulations in section IV. Finally, section V concludes the paper.

II. DISTRIBUTED SCHEDULING IN WIMAX MESH MODE

A. Mesh frame structure

In the distributed scheduling mode, all the stations (BS and SSs) should coordinate their transmissions in their extended two-hop neighborhood. The coordinated distributed scheduling mode uses some or the entire control portion of each frame to regularly transmit its own schedule and proposed schedule changes to all its neighbors. Within a given channel all neighbor stations receive the same schedule transmissions. All the stations in a network have to use the same channel to transmit schedule information in a format of specific resource requests and grants.

A frame in mesh mode operation consists of two parts, the control subframe and the data subframe (Figure 1). The control subframe is dedicated to the transmission of control and management messages. The data subframe is divided into a number of minislots to enable multiple nodes to share access to the medium during the data subframe.

Control subframe serves two functions: network control and schedule control. In a network control subframe, mesh network configuration (MSH-NCFG) and mesh network entry (MSH-NENT) packets provide some basic level of communication for nodes to exchange network configuration information. In a schedule control subframe, the mesh centralized scheduling messages (MSH-CSCH) are used for transmission bursts corresponding to centralized messages, and the rest is allocated to the transmission of the bursts containing mesh distributed scheduling messages (MSH-DSCH) for distributed scheduling. These messages and data structures contains a set of information elements (also denoted in short as IEs) to allow nodes to propagate information about scheduled transmissions (requests and grants), slots available for scheduling and transmission (available resources), to other nodes in the neighborhood. These information elements play a crucial role in distributed scheduling.

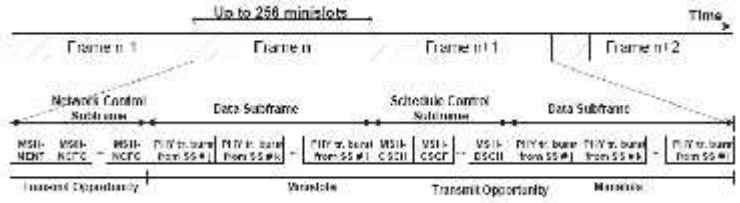


Figure 1:802.16 Mesh Frame structure

B. Mesh distributed scheduling information element

The MSH-DSCH Request IE is used by the node to specify its bandwidth demand for a particular link. Requests shall include parameters including the link indentifier (ID) for which bandwidth is required, the Demand Level and Demand Persistence to quantify the bandwidth required. The value for the field of Demand Level specifies the number of minislots required in a frame to satisfy the bandwidth demand (assuming the current burst profile). The value of the field of Demand

Persistence helps to specify the number of consecutive frames for which the demanded minislots are required.
The MSH-DSCH Availability IEs are used to indicate free minislot ranges that neighbors could issue grant in. It specifies the status of a two-dimensional (frames, minislots) block of minislots.
The MSH DSCH Grant IEs are used for sending grants in response to a bandwidth request as well as for sending a confirmation (grant confirmation) for a received bandwidth grant. A direction field in the grant information element helps to distinguish between a bandwidth grant and a grant confirmation (0 = to granter, 1= from granter).

C. Three-way handshake mechanism

Before data transmission, coordinated scheduling employs a three-way handshake to setup the connections with neighbors. This mechanism (Figure 2) is used to achieve bandwidth reservation for data transmission. It relies on a three-way handshake (bandwidth request, bandwidth grant, bandwidth grant confirmation).
Bandwidth request: based on the availibilty IE, the transmitting node sends a request based on the link ID to identify the link for which the node needs bandwidth, and the number of minislots per frame and their Persistence.
Bandwidth grant: The MSH-DSCH message containing the request IE is received by all the neighbors of the node which transmitted the request. The nodes then process the message to identify if the request is for bandwidth on a link directed to itself. The node to which the request is directed is the granter. The granter looks up its own set of availability information elements to select a subset of availabilities (range of slots and frames) where it is allowed to schedule reception of data transmissions from its neighbors. The number of minislots per frame and their Persistence for the grant is chosen so as to satisfy the request without disturbing other already scheduled data transmissions. The grant is received by all the neighbors of the granter. These then update their availability status to reflect the scheduled reception of data indicated by the grant.
Grant Confirmation: The requester transmits a MSH-DSCH message containing a grant confirmation (with the direction bit set to 0). The grant confirmation informs all the neighbors of the requester of the scheduled transmission. The neighbors then update their availabilities to reflect the newly scheduled transmission. Transmission of data in the reserved slots is allowed only after the transmission of the grant confirmation.

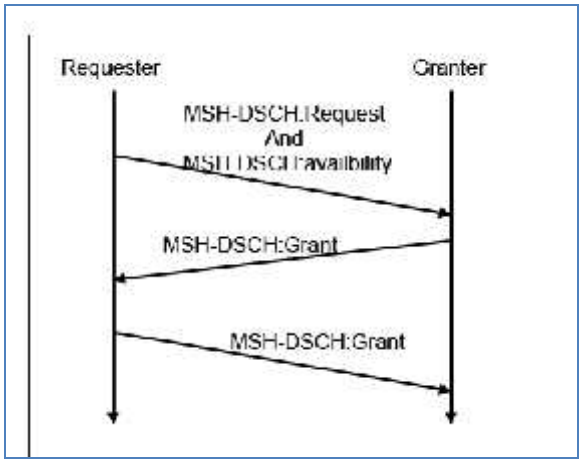


Figure 2: Three-Way Handshake

Note that the status of the slots needs to be changed when minislots reserved for a scheduled transmission are freed (via. cancel request, grant cancel, grant cancel confirmation). The nodes involved in the handshake as well as the passive nodes need to update their availabilities in order to maintain a consistent picture of the resources available at the nodes.

III. SMART ANTENNA BASED SCHEDULING

A. Smart antenna techniques

Omni-directional antennas are inexpensive and simple to build and use, but causing low throughput in mesh networks due to poor spatial reuse. Smart antennas provide better spatial reuse and reduce the interference between simultaneous transmissions to improve the link budget [11][12]. A smart antenna can provide multiple degrees of freedom, which can be used for intended communications. By incorporating a smart antenna with multiple predefined beam patterns in a Wimax node and applying one at a time towards the direction of interest, network throughput can be significantly improved by more efficient spatial reuse. In this context where a node can not receive and transmit at the same time and a node can receive information from different sender on different links, we propose an algorithm that allow several links to be activated simultaneously (in the same minislot at the same time).

A. Proposed algorithm

The mesh network can be modeled by a graph G (N, L) where N represents a set of nodes each being equipped with smart antennas and L represents the direct

communication links. Each activated link is identified by its source destination couple; Lij is the link from the node i to the node j. The system works in a periodical synchronous time-slotted mode. Each node should support a table to take into account the activated link (set of sender nodes, set of receiver nodes) at each minislot. This table is updated based on the exchanged signalling information including availibility and grant IEs. When a node decides to request or to grant sending data it must check this table to find the active link that operate in this time slot. (Figure3)

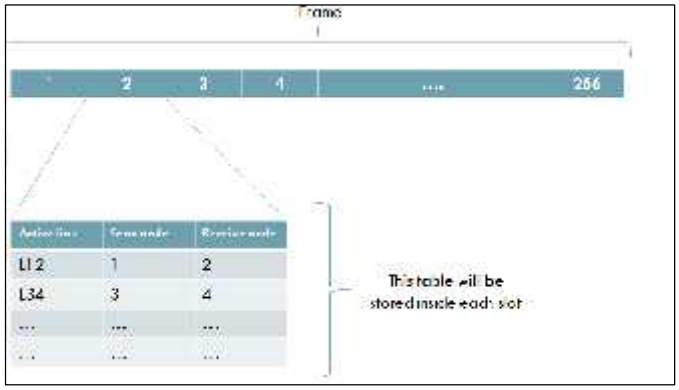
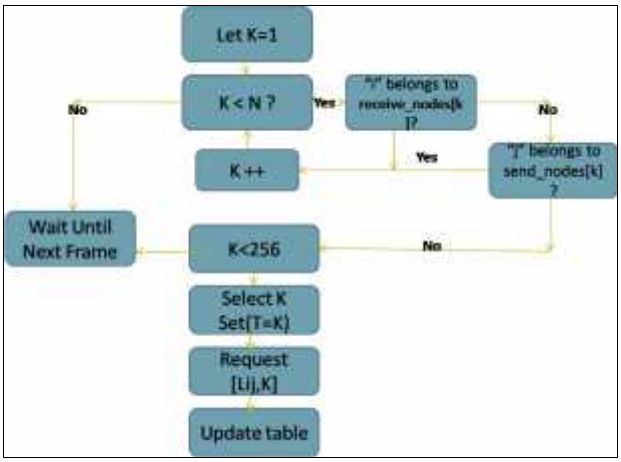


Figure 3: Time Slot Allocation.

Let M be the data minislots in a frame. So based on the three way handshaking method, we propose three algorithms that executed for a frame at each node depending on its status if it:

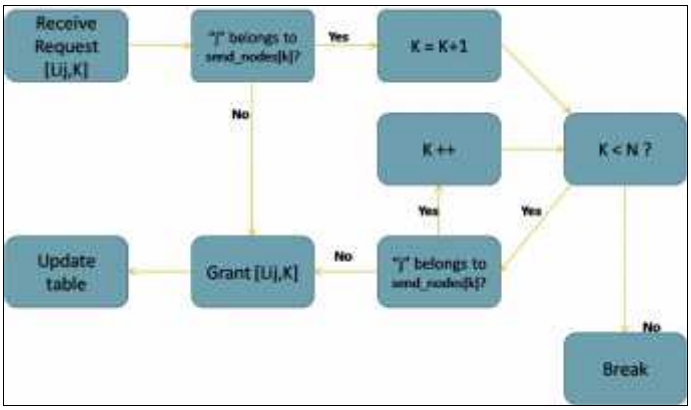
- Request information(Namely the Requester),
- Receive request (Namely the Granter) when it send a Grant.
- Confirm Grant (namely the Requester) when it send Grant Confirmation.

A.1 At The Sending Of A Request:



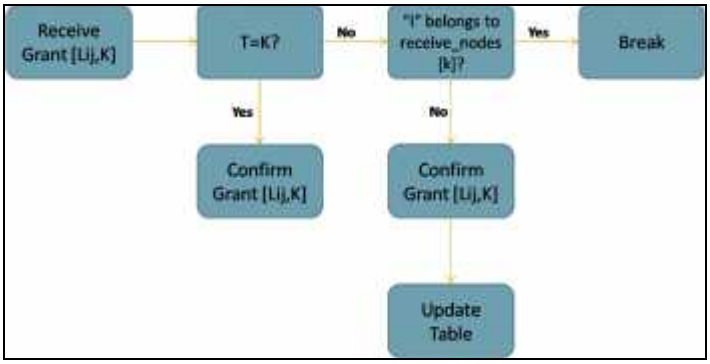
The node here will be named as requester; it will start to check in each frame for the available mini-slot to transmit data. Depending on the algorithm, when a node i (requester) select a TS(time slot) K, and the receiver j. it will check in this TS, if it is belong to the list of receiver nodes(so that it cannot send information), if yes; it will search for another time slot, if no; check again does the granter j belong to list of sender nodes(so that it cannot receive information), if yes search for another TS, if no, select K, request [Lij,K] and update the table. When it updates its table, the table will record then that node I is a sending node and node j is a receiving node and this new state of table will be known by each 2 hop neighborhood in the network.

A.2 At The Reception Of A Request:



The node here will be named as granter it will receive the grant message from the requester and then depending on the algorithm, it will check in its table for the mini-slot specified in the sending message. Does node j (The Granter) in this TS belong to sender nodes (so that it cannot receive information), if no Grant [Lij,K] and update the table. If yes, search for another free time slot where on it the granter does not belong to sender nodes list, Grant [Lij,K] and update the table . Else, keep searching for available TS within the Frame.

A.3 At The Reception Of A Grant:



The requester here when it receive the grant from the granter, will check if T=K (same TS specified in the request message), if yes confirm Grant [Lij,K]. If no, check does node I belong to receiving nodes in the new time slot (so it cannot send information), yes then break the message. No, confirm this new time slot and update the table for all 2 hop neighborhood nodes.

IV. PERFORMANCE EVALUATION

The proposed algorithm is evaluated by simulations using NS2 and the Wimax mesh module (Ns2mesh80216) developed in [5] with required modifications. We compare the implemented algorithm (smart case) with the conventional three-way handshake mechanism used by the FEBA scheduling [5] based on omnidirectionnal antenna (standard case). The performance indicator for comparaison is average simultaneous active links in each scenario. We use two kinds of topologies: BINTREE Topology and Grid Topology (see Figure 4). In each topology, we tried many scenarios by varying the number of nodes and we run different source nodes sending traffic to different destination nodes in each scenario.

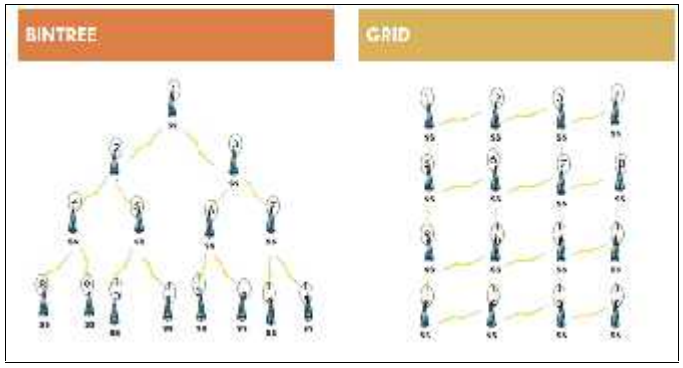


Figure 4: BINTREE Topology VS. GRID Topology.

These figures below are a rough sketch for the network in both scenarios: BINTREE (Figure 5) and GRID (Figure 6), that shows the improvement in the network-number of simultaneous active link using smart antenna compared to standard algorithm. Using Omni-Directional antenna, almost all nodes are blocked so that they cannot receive or send data based on the idea that when two nodes are exchanging information their 2 hop neighborhood must be blocked, However, using smart antenna technique, many nodes can exchange data at the same time since nodes in this technique can send or receive to and from different nodes.

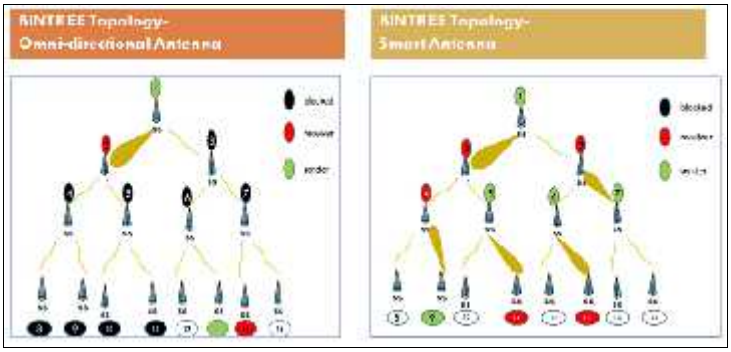


Figure 5: Omni-Directional Antenna vs. Smart Antenna-BINTREE Topology.

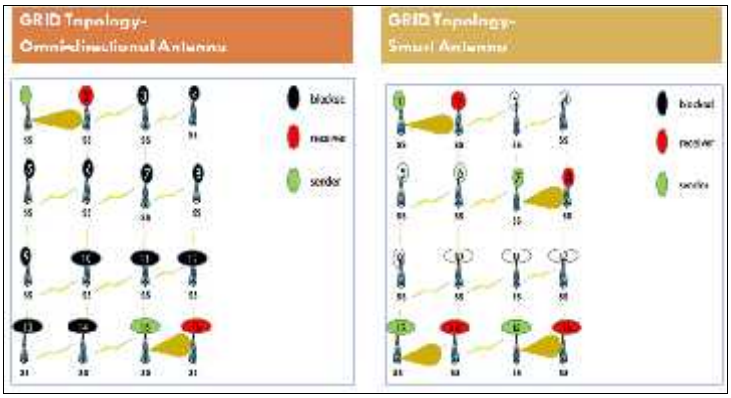


Figure 6: Omni-Directional Antenna vs. Smart Antenna - GRID Topology.

We use two performance indicators to show the improvement in the network:

- Average Active Link Per Time Slot
- Average Throughput

A. Average Active Link Per Time Slot:

Using Smart antenna technique one can see that the Average simultaneous active links increase much better (more than two times) the standard case and the network throughput will then increase. (Figure 7, 8)

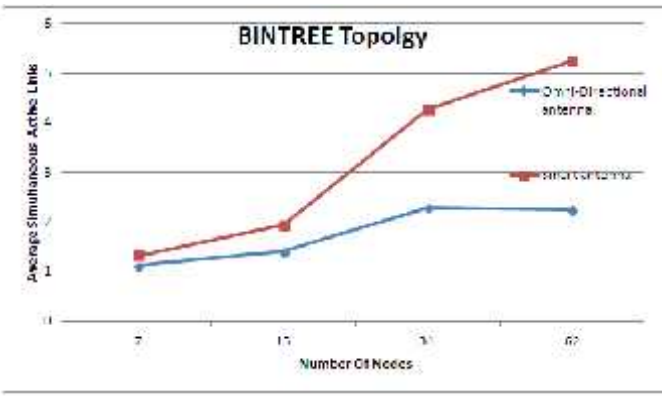


Figure 7: Average Simultaneous Active Links vs. Number of Nodes in BINTREE Topology.

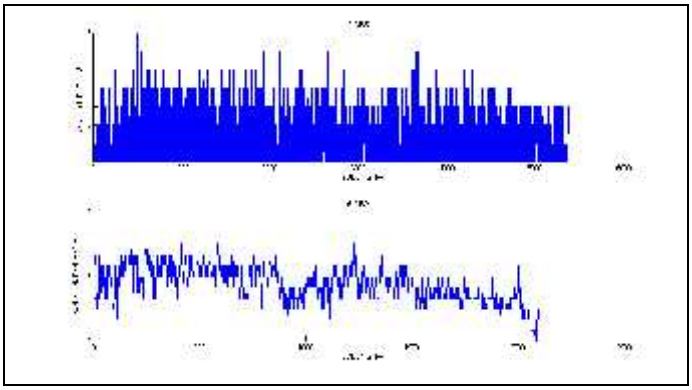


Figure 8: Simulation Result in BINTREE Topology.

The same comment goes for the cases for GRID Topology (figure 9, 10) but the improvement here between smart and Omni-directional antenna is about three times. This can be explained by the fact that in GRID Topology each node may communicate with a number of neighboring nodes more than in the case of BINTREE Topology.

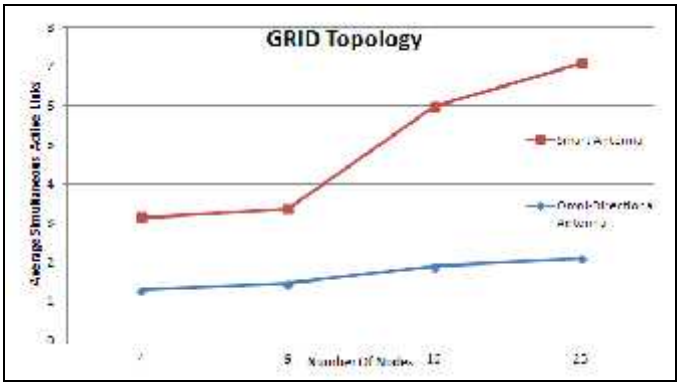


Figure 9: Average Simultaneous Active Links vs. Number of Node in GRID Topology

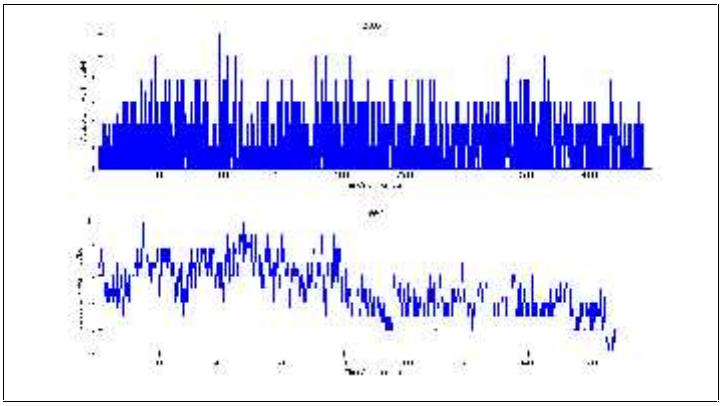


Figure 10: Simulation Result in GRID Topology.

B. Average Throughput

Figure 11 shows the average throughput in MB/S as a function of the number of node in the mesh networks in BINTREE.

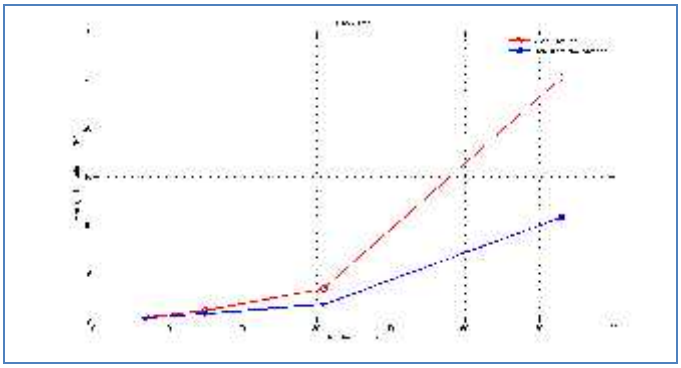


Figure 11: Average Throughput Vs Number of Node in BINTREE Topology.

The results shows when the number of nodes increases, the average throughput increase. Moreover, using smart antenna, it is clear that the average throughput increase much better (more than two times) than the standard case. (Figure 12)

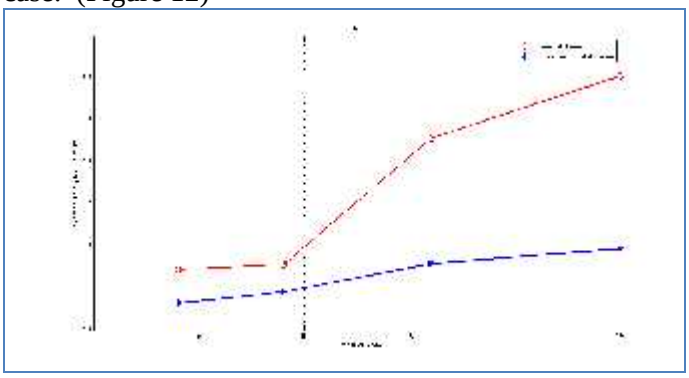


Figure12: Average Throughput Vs Number of Nodes in GRID Topology

V. CONCLUSION

In this paper, we investigated the distributed scheduling in mesh mode and we proposed an algorithm for coordinated distributed scheduling that increases the system capacity using smart antennas. The latter achieve a spatial reuse without increasing interference. The network throughput is maximized by maximizing the number of concurrent transmissions. The simulations results in terms of simultaneously active links show great benefits of the proposed scheme in comparison with the conventional one.

.

REFERENCES

[1] IEEE Standard 802.16e: Air Interface for Fixed Broadband Wireless Access Systems, Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, February 2006.

[2] H. Wei, S. Ganguly, A. Izmailov, and Z. Haas, "Interference-Aware IEEE 802.16 WiMax Mesh Networks", IEEE VTC 2005-Spring. 2005.

[3] H. Shetiya and V. Sharma, "Algorithms for Routing and Centralized Scheduling to Provide QoS in IEEE 802.16 Mesh Networks", in WMuNeP '05: NewYork, USA, 2005.

[4] J. Tao, F. Liu, Z. Zeng, and Z. Lin, "Throughput Enhancement in WiMax Mesh Networks Using Concurrent Transmission," in International Conference on Wireless Communications, Networking and Mobile Computing, Sep 2005, pp. 871-874.

[5] B. Han et al., Performance evaluation of scheduling in IEEE 802.16 based wireless mesh networks, Elsevier Computer Communications, 30 (2007) 782–792.

[6] M. Cao et al., Modelling and performance analysis of Distributed Scheduler in IEEE 802.16 Mesh mode, Proc. of the 6th ACM Int. Symposium on Mobile ad hoc Networking and Computing, 2005.

[7] N. Bayer et al., Improving the Performance of the Distributed Scheduler in IEEE 802.16 Mesh Networks", VTC Spring 2007.

[8] Hua Zhu and Kejie Lu, On The Interference Modeling Issues for Coordinated Distributed Scheduling inIEEE 802.16 Mesh Networks, in Proc. Of the International conference on Broadband Communications, Networks and Systems, 2006. BROADNETS 2006.

[9] C. Cicconetti, I. F. Akyildiz, L. Lenzini, "Bandwidth Balancing in Multi-Channel IEEE 802.16 Wireless Mesh networks," Proc. of the 26th Annual IEEE Conference on Computer Communications (INFOCOM 2007), Anchorage (USA), May 6-12, 2007.

[10] Xiao Xingquan, Sun Xuekang, Xu Baocheng, Guo Zhigang, Proportional-based Distributed wireless-network Cross-layer Scheduling Algorithm, International Conference on Information Technology and Computer Science 2009.

[11] P. H. Lehne and M. Pettersen, "An Overview of Smart Antenna Technology for Mobile Communication Systems," IEEE Communications Surveys, pp. 2-13, Vol. 2, No. 4, Fourth Quarter 1999.

[12] A. Alexiou, M. Haardt, "Smart Antenna Technologies for Future Wireless Systems: Trends and Challenges," IEEE Communications Magazine, pp. 90-97, Sept. 2004.

SESSION

HPC, COMPUTATIONAL SCIENCE, COMPUTATIONAL ENGINES + DISTRIBUTED PROCESSING, AND APPLICATIONS

Chair(s)

TBA

Acceleration of Single- and Multiple-Segment Viterbi Algorithms for Biological Sequence-Profile Comparison on GPU

Alcides C. de Araújo Neto **Nahri Moreano**
Federal University of Mato Grosso do Sul, Brazil
Email: alcides.araujo@hotmail.com, nahri@facom.ufms.br

Abstract— *Over the past few decades the amount of biological data in genomic databases grew up in an exponential rate. Tools such as HMMER use the Viterbi algorithm to find biological sequences that are homologue to a family of sequences represented by a statistical model called profile HMM. Due to the quadratic time complexity of the Viterbi algorithm, this search procedure can demand long execution times depending on database size, sequence size, profile HMM length, and platform used. This paper presents the development and optimization of a high performance solution for the problem of sequence-profile comparison on GPU. We performed a detailed evaluation of several optimizations such as memory optimizations and padding, loop unrolling, multiple streams to enable computation and transfers overlapping, vectorized access to data structures, and tiling. The proposed solution achieved speedups up to 8.8 and 471.7, with respect to HMMER 3.1 execution on a quad-core computer, with and without the use of vector instructions respectively.*

Keywords: Sequence-profile alignment, Viterbi algorithm, HMMER, GPU.

1. Introduction

In the past years, new DNA sequencing technologies have been causing genomic databases to grow in an almost exponential rate. The protein database UniProtKB/TrEMBL [1], for instance, nearly doubled its size every two years since 2000. As a consequence, a huge amount of new genomic data needs to be analyzed, in order to determine their functional content.

The sequence-profile comparison problem, i.e., determining if a newly identified biological sequence is homologous to a known family of sequences is a task of great importance in Bioinformatics. The classification of the new sequence as part of the family allows inferring the function and/or structure of the sequence. HMMER [2], [3], a software solution for conventional computers, is one of the main tools used for this purpose and is based on an important algorithm called Viterbi algorithm [4].

Given the quadratic time complexity of the Viterbi algorithm, comparisons of large sequence databases, long sequences, and long families may result in lengthy execution times. With the rapid growth of biological databases, these execution times become even more critical. Therefore, there is a need for high-performance solutions capable of comparing large amounts of sequences and families in a short time, by the adoption of heuristics and/or exploitation of parallelism.

This paper presents the development of a solution to the sequence-profile comparison problem, using a GPU as execution platform. We apply and evaluate several optimizations such as memory optimizations, memory padding, loop unrolling, multiple streams to enable computation and transfers overlapping, vectorized access to data structures, and tiling. The goal is to achieve a high-performance solution that allows the analysis of large biological databases in an efficient way.

To the authors' knowledge, this is the first GPU-based system proposed for the acceleration of the sequence-profile comparison, which implements the new SSV (Single Segment Viterbi) algorithm, introduced in HMMER 3.1 version, besides the MSV (Multiple Segment Viterbi) algorithm.

This paper is organized as follows. Section 2 introduces the basic concepts needed to understand the sequence-profile comparison problem. In Section 3 we describe related works in parallel sequence-profile comparison. Section 4 presents our GPU solution to this problem and the optimizations we applied to the solution. Section 5 presents the GPU solution experimental performance evaluation and discusses the results obtained. Finally, in Section 6 we draw some conclusions.

2. Sequence-Profile Comparison

A profile HMM (Hidden Markov Model) is a statistical model that represents a family of sequences describing the similarity between members in the form of discrete states. It is based on a multiple alignment of the sequences in the family and represents how conserved each column of the multiple alignment is and which symbols are more likely [5].

A profile HMM representing a family can be used to search for new members of that family in a database of biological sequences [6]. The probability that a sequence S is homologous to a family modeled by a profile HMM λ is determined by finding the sequence of states of λ that produces S with highest probability $P(S|\lambda)$. This score measures the similarity between S and the family modeled, and if it is significant, S is classified as member of the family.

2.1 HMMER

HMMER [3] is a set of tools widely adopted for the analysis of biological sequences. One of its most important tools is *hmmsearch*, which finds biological sequences that are

homologous to a family of sequences, represented by a profile HMM, using the Viterbi algorithm.

The tool uses the following strategy: the sequences pass through a chain of filters, where each filter computes a score for the sequence using a different algorithm and depending on the score, the sequence is discarded or forwarded to the next filter. The initial filters, which receive more sequences, use less precise and faster algorithms, while the final filters, which receive less sequences, use more precise and slower algorithms. Therefore, the initial filters are able to discard a large amount of the sequences being compared, forwarding to the next filter only a small fraction of them.

In HMMER 3.1 version [7], the main filters use the SSV, MSV, and Viterbi algorithms. SSV is the first filter in the chain of filters, while MSV is the second one. Figure 1 shows the profile HMM architecture used by the Viterbi algorithm in HMMER.

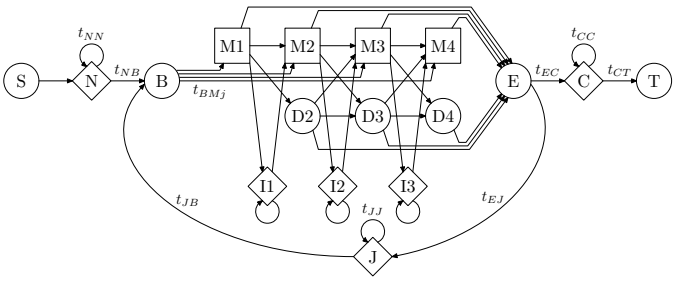


Fig. 1
PROFILE HMM ARCHITECTURE USED BY THE VITERBI ALGORITHM IN HMMER

2.2 Viterbi, MSV, and SSV Algorithms

The Viterbi algorithm for the comparison of a sequence of length L to a profile HMM of length Q is shown in Algorithm 1 and uses the dynamic programming technique to compute the score of the best alignment of the sequence to the profile HMM, using score vectors and matrices corresponding to the states of the profile HMM. The algorithm has time complexity $O(L \times Q)$.

Analyzing the Viterbi algorithm we can identify the data dependences for computing the scores. From lines 3, 4 and 5 of Algorithm 1, we conclude that cells $M[i-1, j-1]$, $I[i-1, j]$ and $D[i, j-1]$ are needed for computing cells $M[i, j]$, $I[i, j]$ and $D[i, j]$, respectively. These dependences prevent the parallel computation of cells in a same row, column, or diagonal of the matrices.

The data dependences in lines 9 and 10 of Algorithm 1 are related to the J state of the profile HMM, which links the end of the profile HMM core to its beginning (feedback loop). This link creates the dependency chain $M[i-1, 1 \ldots Q] \rightarrow E[i-1] \rightarrow J[i-1] \rightarrow B[i-1] \rightarrow M[i, j]$, which prevents the parallel computation of cells in a same anti-diagonal of the matrices.

The MSV algorithm is a simplification of the Viterbi algorithm, obtained by removing states I and D and considering transition probabilities for $M_{j-1} \rightarrow M_j$ as 1.0 [2].

Algorithm 1 Viterbi algorithm

Input: Profile HMM with length Q, emission probabilities P_{em}, transition probabilities P_{tr}, sequence S with length L
Output: Score of the best alignment of S with the profile HMM

1: **for** $i \leftarrow 1$ **to** L **do**
2: **for** $j \leftarrow 1$ **to** Q **do**
3: $M[i,j] \leftarrow P_{em}(M_j, S_i) + max \begin{cases} M[i-1,j-1] + P_{tr}(M_{j-1}, M_j) \\ I[i-1,j-1] + P_{tr}(I_{j-1}, M_j) \\ D[i-1,j-1] + P_{tr}(D_{j-1}, M_j) \\ B[i-1] + P_{tr}(B, M_j) \end{cases}$
4: $I[i,j] \leftarrow P_{em}(I_j, S_i) + max \begin{cases} M[i-1,j] + P_{tr}(M_j, I_j) \\ I[i-1,j] + P_{tr}(I_j, I_j) \end{cases}$
5: $D[i,j] \leftarrow max \begin{cases} M[i,j-1] + P_{tr}(M_{j-1}, D_j) \\ D[i,j-1] + P_{tr}(D_{j-1}, D_j) \end{cases}$
6: **end for**
7: $N[i] \leftarrow N[i-1] + P_{tr}(N, N)$
8: $E[i] \leftarrow max_{1 \le j \le Q}(M[i,j] + P_{tr}(M_j, E))$
9: $J[i] \leftarrow max \begin{cases} J[i-1] + P_{tr}(J, J) \\ E[i] + P_{tr}(E, J) \end{cases}$
10: $B[i] \leftarrow max \begin{cases} N[i] + P_{tr}(N, B) \\ J[i] + P_{tr}(J, B) \end{cases}$
11: $C[i] \leftarrow max \begin{cases} C[i-1] + P_{tr}(C, C) \\ E[i] + P_{tr}(E, C) \end{cases}$
12: **end for**
13: **return** $C[L] + P_{tr}(C, T)$

Algorithm 2 shows the MSV algorithm, which has time complexity $O(L \times Q)$.

Algorithm 2 MSV algorithm

Input: Profile HMM with length Q, emission probabilities P_{em}, transition probabilities P_{tr}, sequence S with length L
Output: Score of the best alignment of S with the profile HMM

for $i \leftarrow 1$ **to** L **do**
for $j \leftarrow 1$ **to** Q **do**
$M[i,j] \leftarrow P_{em}(M_j, S_i) + max \begin{cases} M[i-1,j-1] \\ B[i-1] + P_{tr}(B, M_j) \end{cases}$
end for
$N[i] \leftarrow N[i-1] + P_{tr}(N, N)$
$E[i] \leftarrow max_{1 \le j \le Q}(M[i,j])$
$J[i] \leftarrow max \begin{cases} J[i-1] + P_{tr}(J, J) \\ E[i] + P_{tr}(E, J) \end{cases}$
$B[i] \leftarrow max \begin{cases} N[i] + P_{tr}(N, B) \\ J[i] + P_{tr}(J, B) \end{cases}$
$C[i] \leftarrow max \begin{cases} C[i-1] + P_{tr}(C, C) \\ E[i] + P_{tr}(E, C) \end{cases}$
end for
return $C[L] + P_{tr}(C, T)$

Algorithm 3 SSV algorithm

Input: Profile HMM with length Q,
emission probabilities P_{em}, transition probabilities P_{tr},
sequence S with length L

Output: Score of the best alignment of S with the profile HMM

for $i \leftarrow 1$ **to** L **do**
  for $j \leftarrow 1$ **to** Q **do**
    $M[i,j] \leftarrow P_{em}(M_j, S_i) + M[i-1, j-1]$
  end for
  $E[i] \leftarrow max \begin{cases} E[i-1] \\ max_{1 \le j \le Q}(M[i,j]) \end{cases}$
end for
return $E[L]$

Figure 2(a) shows the score matrix and vectors used in the MSV algorithm and represents the data dependences for computing the scores by arrows. Although it has the same time complexity as the Viterbi algorithm, the MSV algorithm performs fewer computations and has less data dependences, due to states I and D removal. As a consequence, all cells in a same row of matrix M can be computed in parallel, while successive rows must still be computed sequentially.

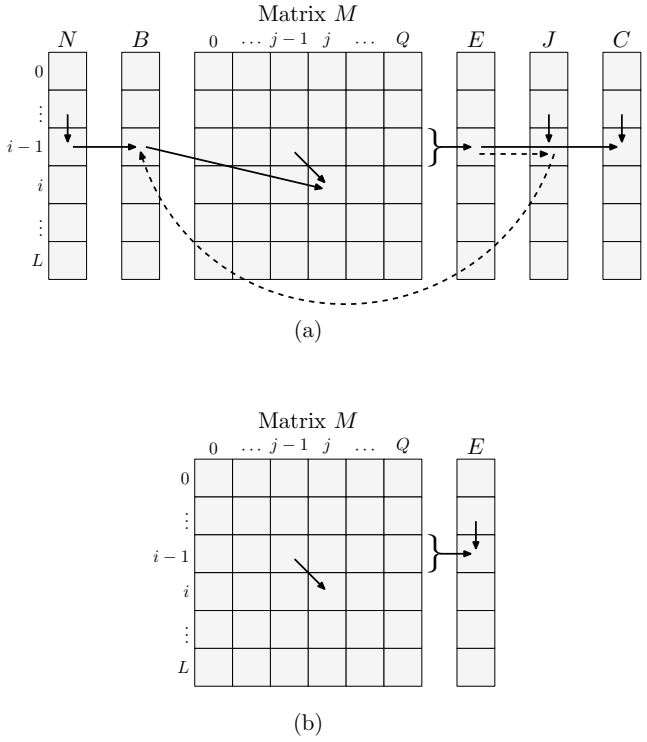


Fig. 2
DATA DEPENDENCES IN THE (A) MSV AND (B) SSV ALGORITHMS

The SSV algorithm is a further simplification of the MSV algorithm obtained by removing the J state, and, as a consequence, the feedback loop. This approach has the goal of improving performance at the expense of accuracy loss, because multi-hit alignments between the sequence and the profile HMM are no longer possible. Algorithm 3 shows the SSV algorithm, which has time complexity $O(L \times Q)$. Despite having the same time complexity as the previous algorithms, the SSV algorithm performs fewer computations and has significantly reduced data dependences, which are shown in Figure 2(b). Again, it is possible to compute in parallel all cells in a same row of matrix M.

3. Related Work

Cluster-based solutions are used to improve HMMER2 performance, exploiting sequence parallelism only in Viterbi algorithm [8], [9]. The idea is that, when comparing a set of sequences to a profile HMM, there are no dependences between the score matrices corresponding to two distinct sequences, therefore the matrices can be computed in parallel.

Most FPGA (Field-Programmable Gate Array) solutions [10], [11], [12], [13] use a systolic array and implement only the HMMER2 Viterbi algorithm, eliminating state J and exploiting anti-diagonal data parallelism, but decreasing similarity score accuracy. Some accelerators exploit limited sequence parallelism [11], [14], while others apply strategies to reduce the accuracy loss [12], [13]. Abbas and Derrien [15] implement a FPGA accelerator for the HMMER3 MSV and Viterbi algorithms, rewriting the recurrence equations to expose more parallelism.

ClawHMMER [16] is an implementation of only the Viterbi algorithm on GPU using the Brook language. It uses profile HMMs with only M, I, and D states and exploits sequence parallelism. The sequences are sorted by length and divided into batches, in order to fit in GPU memory and provide load balance. Executing on a ATI R520, it reached a speedup of 36 compared to HMMER2 executing on Intel Pentium 4 2.8GHz.

Walters *et al.* [17] implements only the Viterbi algorithm on GPU, using the CUDA programming model and exploiting sequence parallelism. The sequences are sorted by length to provide load balance and the inner loop of Viterbi algorithm is unrolled. Score matrices are stored in GPU global memory and accessed with coalescency, while transition and emission probabilities are kept in constant and texture memory. Using a GeForce GTX 8800 Ultra, they reached speedups between 12 and 38.6 compared to HMMER2 executing on AMD Athlon 2.2GHz.

CuHMMER [18] also implements only the Viterbi algorithm on GPU with CUDA, exploiting sequence parallelism. The sequences are grouped based on their length to provide load balance, and transition and emission probabilities are stored in GPU shared or texture memory. Using a GeForce GTX 8800, it reached speedups between 13 and 45 compared to HMMER2 executing on AMD Athlon64 X2 Dual Core processor.

Du et al. [19] implements only the Viterbi algorithm on GPU using CUDA and profile HMMs with only M, I, and D states. Since the J state does not exist, the feedback loop is broken and it is possible to compute M, I, and D anti-diagonal cells in parallel, one anti-diagonal at a time. They implement three different approaches concerning the score matrices storage. Using a GeForce GTX 9800, they reached

speedups between 1.97 and 72.21 compared to HMMER2 executing on Intel Dual Core 2.83GHz.

Ganesan et al. [20] implements only the Viterbi algorithm on GPU using CUDA. They iterate the Viterbi algorithm recurrences, allowing cells of the same row of M and D score matrices to be computed in parallel, while successive rows are computed sequentially. Using a cluster of four NVIDIA Tesla C1060, they reached a speedup of 100 compared to a serial implementation of the Viterbi algorithm executing on AMD Opteron 2.33GHz.

Quirem *et al.* [21] implements only the MSV algorithm on GPU using CUDA. Each sequence to be analyzed is assigned to a different block, and the threads of a block compute the cells in a same row of the score matrix in parallel. Optimizations such as asynchronous data transfer and kernel execution, and the use of pinned memory are applied. They achieved speedups between 10 and 15, executing on a NVIDIA Tesla C1060, compared to HMMER3 executed on the host.

Li *et al.* [22] also implements only the MSV algorithm on GPU using CUDA. They perform coalesced memory access, fetch multiple data chunks from memory at once, keep frequently used data in registers, convert sequences symbols to numbers to simplify operations, sort the sequences by length to provide load balance, perform asynchronous transfers, and store the probabilities in the texture memory. A speculative approach for the MSV algorithm is adopted, where the outer loop is unrolled by a factor of 2 and the score $B[i]$ is computed without considering the transition $J \rightarrow B$. When the speculation fails, the sequence score is recalculated on the host. Executing on Intel Xeon E5506 with a NVIDIA Tesla C2050, they achieved speedups up to 6.5 compared to HMMER3 with SSE executed on a single core.

In general, FPGA accelerators achieve good performance results, at the expense of accuracy loss, while cluster solutions produce accurate similarity results, however with smaller performance gains. CUPS performance results are not reported for the described GPU solutions.

To the best of our knowledge, there are not in the literature solutions implementing the SSV algorithm in GPU or other parallel computing platform, apart from the HMMER 3.1 tool suite. As a consequence, all works described in this section compared their results to previous and slower versions of HMMER, in which the fast SSV algorithm was not used.

4. GPU Solution to Sequence-Profile Comparison

We developed a host-GPU solution for the sequence-profile comparison, using C++ and CUDA. The solution receives as inputs a set of sequences to be compared and a profile HMM representing a family, and computes the similarity score of the best alignment between each sequence and the profile HMM.

4.1 Chain of Filters

Our solution implements the chain of filters shown in Figure 3. Each box represents a filter which executes an algorithm and the arrows represent the paths that a sequence being compared can take. Each sequence is compared by one or more algorithms until it is discarded or accepted, when the final score of the sequence is obtained. The acceptance criteria used after each filter are based on those of the HMMER 3.1 *hmmsearch* tool.

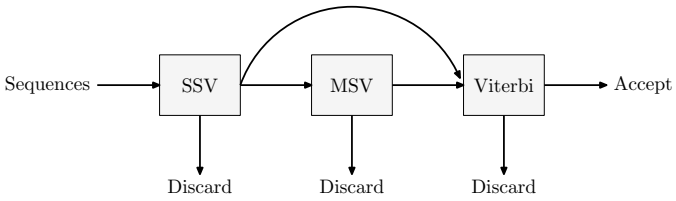


Fig. 3
CHAIN OF FILTERS OF THE PROPOSED HOST-GPU SOLUTION

The initial SSV and MSV filters receive many more sequences than the final Viterbi filter. Therefore, our efforts to improve performance are focused on the initial filters. The SSV and MSV filters are implemented as kernels running on GPU, while the Viterbi algorithm executes on the host processor. The host also controls the path that a sequence shall take: after a sequence is compared by a filter, the host decides, based on the sequence partial score, if the sequence is discarded or forwarded to the next filter.

4.2 Parallelism Approach

Both MSV and SSV kernels were modeled in a way to exploit simultaneously task and data parallelism. We exploit task parallelism assigning distinct sequences to each CUDA block. Since the comparison of distinct sequences are independent from each other, several sequences can be compared simultaneously by executing several blocks concurrently on the GPU. On the other hand, data parallelism is exploited through the many threads of each block, which compute the cells in the same row of score matrix M in parallel.

Figure 4 illustrates this approach, where an execution space is shown with a grid of n blocks, each block with several threads. The distinct sequences S_0, ..., S_{n-1} are assigned to the blocks, so each block performs the comparison of a sequence to the profile HMM (task parallelism). The threads of a block compute in parallel the cells in the same row of matrix M corresponding to the sequence assigned to that block (data parallelism).

4.3 MSV Kernel with Optimized Reduction

The MSV kernel executes a sequential loop with L iterations to compute matrix M and other score vectors, where at iteration i, Q threads compute row i of M in parallel, in a way that thread j computes cell $M[i, j]$. After computing row i, a reduction operation is performed to obtain $E[i]$ as the maximum among the cells in this row. The reduction operation is performed once for each iteration of the sequential loop with L iterations. By the end of all iterations, the score of the best alignment is obtained.

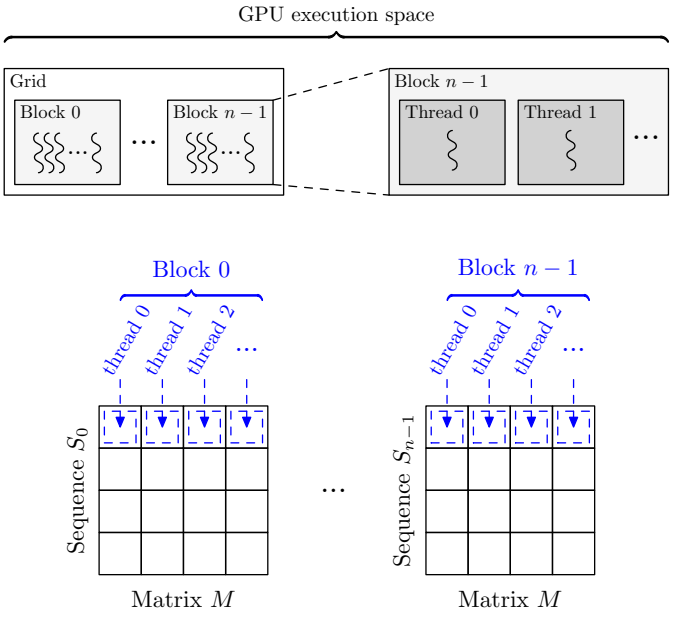


Fig. 4
PARALLELISM IN THE GPU SOLUTION: TASK PARALLELISM (COMPARISON OF DISTINCT SEQUENCES BY DIFFERENT BLOCKS) AND DATA PARALLELISM (COMPUTATION OF CELLS OF SCORE MATRIX M BY THREADS OF A BLOCK)

The reduction operation reduces a collection of data (a vector) into a single value, using an associative binary operation, which in our case is the maximum operation. Finding the maximum of Q numbers in parallel is performed in $\log_2 Q$ sequential steps. At each step, the first half of the vector is compared to the second one, discarding half of the values and writing the maximum values in the first half. This half is used as the input to the next step. By the end of $\log_2 Q$ steps, the maximum value of the vector is obtained at the first position of the vector. The comparisons performed in a same step are executed in parallel by different threads. At the end of each step, a barrier synchronization is needed to avoid race conditions.

In our GPU solution, this reduction operation is optimized as follows. The loop unrolling technique is applied, unrolling the last six iterations of the $\log_2 Q$ steps. These iterations find the maximum among the 64 values ($\log_2 64 = 6$) in the first positions of the vector, which is performed by 32 threads, that is, a warp. Since the threads in a warp execute synchronously, the barriers between these six steps can be removed. Barrier synchronizations degrade performance, therefore minimizing them improves the performance of the GPU solution. The other iterations are also unrolled, in order to simplify index computations and reduce loop overhead, however their barriers must be kept.

4.4 SSV Kernel with Optimized Reduction

Both MSV and SSV algorithms (Algorithms 2 and 3) compute $E[i]$ as the maximum value of row i of matrix M. However, in MSV algorithm $E[i]$ is used to compute $C[i]$, while there is not vector C in SSV algorithm. This latter algorithm finds the maximum value in vector E, in order to obtain the score of the best alignment.

The SSV kernel of the GPU solution executes a sequential loop with L iterations to compute matrix M, where at iteration i, Q threads compute row i of M in parallel, in a way that thread j computes cell $M[i, j]$. After computing row i, each thread j compares $M[i, j]$ to the maximum value it accumulated in the previous loop iterations. Then, each thread j obtains the maximum value of column j of matrix M, in parallel to the other threads.

By the end of all iterations, one reduction operation is performed to obtain the score of the best alignment among the Q maximum values accumulated. Thus, the SSV kernel produces the same result as Algorithm 3, however performing the operations in a different order than that indicated in the algorithm, so that we are able to better map the operations to the GPU hardware and, consequently, to exploit more data parallelism.

Another advantage of this modification is that the number of reduction operations performed in the SSV kernel is decreased by a factor of L, in comparison to the MSV kernel. Besides, the number of barrier synchronizations needed for the reduction operations is also decreased by a factor of L.

4.5 Optimizations

Initially we developed a basic GPU implementation for the sequence-profile comparison, with no optimizations applied. Then, we applied several optimizations to our basic GPU implementation. We evaluated the optimizations separately, in order to identify which ones provide performance gains, and then, we evaluated how they behave in combination with the others. Finally, an optimized GPU implementation was developed, containing the selected optimizations.

Our GPU solution is able to handle input cases with profile HMM model and/or sequence length longer than the maximum number of threads of the GPU used. In order to do that, we implemented both MSV and SSV kernels using the tiling technique [23], with each thread being associated to several cells, instead of only one.

Table 1 lists the main optimizations evaluated and which ones resulted in performance gains and were applied in the final optimized version of the SSV and MSV kernels of the GPU solution. Besides these optimizations, other established optimizations were also applied to both kernels, such as coalesced access to global memory, storing score structure in shared memory, keeping frequently used data in registers and reducing thread divergences.

5. Results and Discussion

The execution platform used in this work consists of a GPU NVIDIA GeForce GTX 570 connected to a host with Intel Core i7-3770S processor and 8GB RAM. We evaluated our solution using the entire UniProtKB/Swiss-Prot [1] database, composed of 540,732 sequences, with average and maximum length of 355.24 and 35,213, respectively. Ten sequence families were selected from the Pfam database [24] for our

Table 1
OPTIMIZATIONS APPLIED TO THE SSV AND MSV KERNELS OF THE GPU SOLUTION

Optimization	Kernels	SSV	MSV
Pinned memory use in host-GPU transfers		•	•
Max operation reduction optimized at compile time		•	•
Representing scores with natural numbers		•	•
Emission probabilities stored in texture memory		•	
Padding in emission probabilities to achieve coalescency		•	
Padding in sequences to achieve coalescency		•	
Sequences sorted by length to provide load balance		•	•
Loop unrolling (factor 8) of outer loop		•	•
Multiples streams to overlap GPU computation and host-GPU transfers		•	•
Vectorized access to sequences		•	
Vectorized access to emission probabilities		•	•
Tiling to improve occupancy and handle long HMMs		•	•

experiments. Table 2 lists the selected families and the length of their corresponding profile HMMs.

Table 2
SELECTED FAMILIES FROM PFAM DATABASE USED IN THE EXPERIMENTS

Family	Profile HMM length
Avian_gp85	256
CABIT	256
DUF530	512
PaRep2b	512
Flu_PB2	759
Totivirus_coat	759
ACR_tran	1,021
RdRP_5	1,271
Bac_GDH	1,528
AvrE	1,774
Average	864.80

In our experiments, we measured the execution time necessary to compare all sequences in the UniProtKB/Swiss-Prot database to each selected family, using our GPU solution and HMMER 3.1 *hmmsearch* tool. HMMER 3.1 was executed on the host, with six different configurations, namely using 1, 2, or 4 cores of the processor, with the SSE2 vector instructions enabled or not.

Table 3 shows, for the main optimizations listed in Table 1, the speedup they produced compared to the basic non-optimized GPU solution, for the kernels SSV and MSV. The empty fields correspond to profile HMMs with length longer than 1024. The tiling technique has not been applied to the basic non-optimized GPU solution yet, therefore this solution is not able to handle these families.

Since basically all optimizations resulted in performance gains for the SSV kernel, most of them were adopted on the final optimized GPU solution. On the other hand, for the MSV kernel, some optimizations did not result in performance gains. In most cases, the problem was related to the use of additional registers and consequently lower occupancy rates of the GPU.

Figure 5 shows the average execution time of the optimized GPU solution and HMMER 3.1 with the six different configurations, using logarithmic scale. These results represent

Table 3
SPEEDUP PRODUCED BY MAIN OPTIMIZATIONS, FOR KERNELS SSV AND MSV, WRT. TO THE BASIC NON-OPTIMIZED GPU SOLUTION

Optimization	Reduction optimized in compile time		Scores as natural numbers		Emission probabilities in texture memory		Padding in emission probabilities		Loop unrolling		Vectorized access to sequences		Vectorized access to emission probabilities		Tiling	
Family/Kernel	MSV	SSV	SSV	SSV	SSV	SSV	SSV	SSV	SSV	SSV	SSV	SSV	MSV	SSV	MSV
Avian_gp85	1.2	1.1	1.0	1.0	1.5	1.5	1.0	2.1	1.6	2.1
CABIT	1.2	1.1	1.0	1.0	1.5	1.5	1.0	2.1	1.6	2.1
DUF530	1.2	1.1	1.1	1.0	1.5	1.4	1.9	3.6	2.1	3.2
PaRep2b	1.2	1.1	1.1	1.0	1.5	1.4	1.9	3.6	2.1	3.2
Flu_PB2	1.1	1.0	1.2	1.1	1.4	1.4	2.1	4.7	2.0	3.5
Totivirus_coat	1.1	1.0	1.2	1.1	1.4	1.4	2.1	4.7	2.0	3.5
ACR_tran	1.2	1.0	1.0	1.1	1.5	1.5	2.8	6.5	2.4	4.9
RdRP_5	-	-	-	-	-	-	1.7	2.9	1.4	2.0
Bac_GDH	-	-	-	-	-	-	1.6	2.6	1.4	1.8
AvrE	-	-	-	-	-	-	1.9	4.0	1.5	2.8

the average running time for comparing all sequences in the UniProtKB/Swiss-Prot database to each selected family from Pfam database. The GPU solution average execution time shows an approximately one order of magnitude improvement with respect to HMMER 3.1 average execution time, when SSE2 instructions are enabled. If these vector instructions are not used, the improvement is approximately three orders of magnitude.

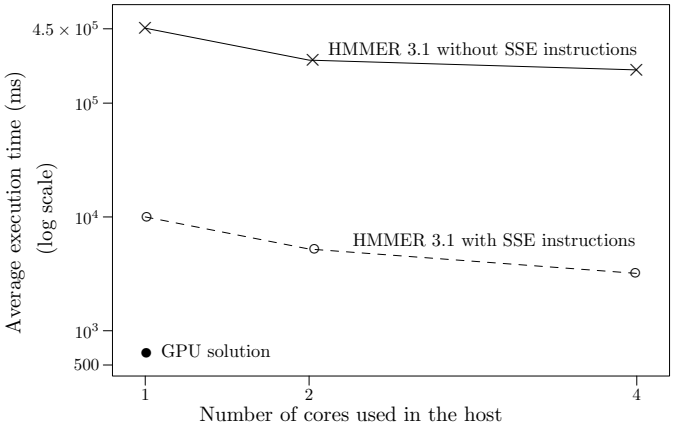


Fig. 5
AVERAGE EXECUTION TIME OF GPU SOLUTION AND HMMER 3.1 (FOR THE SIX CONFIGURATIONS)

Table 4 shows the speedups achieved by the optimized GPU solution with respect to HMMER 3.1 with the six different configurations. The GPU solution achieved impressive speedups up to 1072.0 and 471.7 compared to the sequential single-core and quad-core configurations, respectively. Besides, the GPU solution also overcomes HMMER 3.1 with SSE2 instructions enabled, reaching speedups up to 28.8 and 8.8, for the single-core and quad-core configurations, respectively.

Table 4
SPEEDUPS OF GPU SOLUTION WRT. HMMER 3.1, FOR COMPARING THE ENTIRE UNIPROTKB/SWISS-PROT SEQUENCE DATABASE TO EACH FAMILY (HMMER 3.1 EXECUTED WITH SIX DIFFERENT CONFIGURATIONS)

Family	Without SSE instructions			With SSE instructions		
	1 core	2 cores	4 cores	1 core	2 cores	4 cores
Avian_gp85	401.2	205.8	114.7	6.4	3.4	3.7
CABIT	405.7	208.1	114.9	6.9	3.7	3.4
DUF530	794.0	409.5	255.7	12.2	6.4	4.4
PaRep2b	799.6	413.0	267.5	12.7	6.7	4.5
Flu_PB2	683.0	355.5	289.7	11.6	6.1	3.7
Totivirus_coat	992.1	514.6	327.0	**28.8**	**15.0**	**8.8**
ACR_tran	**1072.0**	**559.5**	456.4	26.3	13.8	8.1
RdRP_5	750.5	397.3	364.5	12.4	6.5	3.6
Bac_GDH	934.9	495.0	**471.7**	18.7	9.7	5.6
AvrE	860.8	455.1	419.4	24.4	12.7	7.6

Another observation is the higher speedups obtained by the families ACR_tran and Bac_GDH. The explanation lies in the lengths of their profile HMMs, 1.021 and 1.528, respectively. The profile HMM length is directly associated with the amount of threads used in the GPU solution and the lengths of these families, combined with the tiling technique applied, provide better GPU occupancy rates.

We also report the performance using the throughput measure CUPS (Cell Updates per Second), which indicates how many cells of the dynamic programming matrices are computed in one second. Table 5 shows the average and maximum GCUPS (10^9 CUPS) achieved by the GPU solution and HMMER3.1 (for the six configurations), for comparing the entire UniProtKB/Swiss-Prot database to all selected families.

Table 5
GCUPS OF HMMER 3.1 (FOR SIX CONFIGURATIONS) AND GPU SOLUTION

GCUPS	HMMER 3.1 w/o SSE			HMMER 3.1 w/ SSE			GPU
	1 core	2 cores	4 cores	1 core	2 cores	4 cores	
Average	0.38	0.73	1.02	20.35	38.75	57.47	286.12
Maximum	0.40	0.79	1.41	26.27	49.98	82.30	372.06

6. Conclusion

We developed of a high-performance GPU solution to the sequence-profile comparison problem, and applied several optimizations such as memory optimizations, padding, loop unrolling, multiple streams and computation and transfers overlapping, vectorized access, and tiling.

We performed a comprehensive performance evaluation using a modest GPU and a large and representative biological data set. The GPU solution achieved impressive speedups up to 1072.0 and 471.7 compared to the HMMER 3.1 tool running on single-core and quad-core computers, respectively. Besides, the GPU solution also overcomes HMMER 3.1 with SSE2 instructions enabled, reaching speedups up to 28.8 and 8.8, for the single-core and quad-core configurations, respectively.

The GPU solution produced the maximum of 372.06 GCUPS, while HMMER 3.1 executing on a quad-core computer with SSE2 instructions enabled produced the maximum of 82.30 GCUPS. Using a more powerful GPU, our solution would achieve an even better performance, since less tiling would be necessary. We can also conclude that, by modeling a problem properly and applying optimizations targeting the platform, GPUs can reach high performances.

To the best of our knowledge, this is the first GPU-based system proposed for the acceleration of the sequence-profile comparison, which implements the SSV algorithm (introduced in HMMER 3.1), besides the MSV algorithm. There are not in the literature solutions implementing the SSV algorithm in any parallel computing platform, apart from the HMMER 3.1 tool suite.

References

[1] T. UniProt Consortium, "Activities at the Universal Protein Resource (UniProt)," *NAR*, vol. 42, no. D1, pp. 191–198, 2014.

[2] S. Eddy, "Accelerated Profile HMM Searches," *PLoS Computational Biology*, vol. 7, no. 10, pp. 1–16, 2011.

[3] Howard Hughes Medical Institute, "HMMER," http://hmmer.janelia.org/, last viewed July 2014.

[4] G. Forney Jr., "The Viterbi Algorithm," *Proc. of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[5] A. Melo and N. Moreano, *Reconfigurable Embedded Control Systems: Applications for Flexibility and Agility*. IGI Global, 2010, ch. FPGA-Based Accelerators for Bioinformatics Applications, pp. 311–341.

[6] A. Krogh, *Computational Methods in Molecular Biology*. Elsevier, 1999, vol. 32, ch. An Introduction to Hidden Markov Models for Biological Sequences, pp. 45–63.

[7] S. Eddy and T. Wheeler, *HMMER User's Guide*, 3rd ed., 2013.

[8] J. P. Walters, R. Darole, and V. Chaudhary, "Improving MPI-HMMER's Scalability With Paralell I/O," in *IPDPS*, 2009, pp. 1–11.

[9] K. Jiang *et al.*, "An Efficient Parallel Implementation of the Hidden Markov Methods for Genomic Sequence-Search on a Massively Parallel System," *TPDS*, vol. 19, no. 1, pp. 15–23, 2008.

[10] K. Benkrid, P. Velentzas, and S. Kasap, "A High Performance Reconfigurable Core for Motif Searching Using Profile HMM," in *AHS*, 2008, pp. 285–292.

[11] A. C. Jacob *et al.*, "Preliminary Results in Accelerating Profile HMM Search on FPGAs," in *IPDPS*, 2007, pp. 1–8.

[12] R. P. Maddimsetty *et al.*, "Accelerator Design for Protein Sequence HMM Search," in *SC*, 2006, pp. 288–296.

[13] Y. Sun *et al.*, "Accelerating HMMER on FPGAs Using Systolic Array Based Architecture," in *IPDPS*, 2009, pp. 1–8.

[14] S. Derrien and P. Quinton, "Parallelizing HMMER for Hardware Acceleration on FPGAs," in *ASAP*, 2007, pp. 10–17.

[15] N. Abbas and S. Derrien, "Accelerating HMMER on FPGA using Parallel Prefixes and Reductions," INRIA, France, Tech. Rep. 7370, 2010.

[16] D. Horn, M. Houston, and P. Hanrahan, "ClawHMMER: A Streaming HMMer-Search Implementation," in *SC*, 2005, p. 11.

[17] J. P. Walters *et al.*, "Evaluating the use of GPUs in Liver Image Segmentation and HMMER Database Searches," in *IPDPS*, 2009, pp. 1–12.

[18] P. Yao *et al.*, "CuHMMER: A Load-balanced CPU-GPU Cooperative Bioinformatics Application," in *HPCS*, 2010, pp. 24–30.

[19] Z. Du, Z. Yin, and D. Bader, "A Tile-based Parallel Viterbi Algorithm for Biological Sequence Alignment on GPU with CUDA," in *IPDPS*, 2010, pp. 1–8.

[20] N. Ganesan *et al.*, "Accelerating HMMER on GPUs by Implementing Hybrid Data and Task Parallelism," in *BCB*, 2010, pp. 418–421.

[21] S. Quirem, F. Ahmed, and B. Lee, "CUDA Acceleration of P7Viterbi Algorithm in HMMER 3.0," in *IPCCC*, 2011, pp. 1–2.

[22] X. Li *et al.*, "A Speculative HMMER Search Implementation on GPU," in *IPDPS*, 2012, pp. 735–741.

[23] D. Kirk and W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010.

[24] R. Finn *et al.*, "Pfam: the protein families database," *NAR*, vol. 42, no. D1, pp. 222–230, 2014.

FPGA-Oriented Design of an FDTD Accelerator Based on Overlapped Tiling

Yasuhiro Takei, Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {takei, hasitha, hariyama, kameyama}@ecei.tohoku.ac.jp

Abstract— *In this paper, we introduce the overlapped tiling to designing an FPGA-based FDTD accelerator by using an OpenCL compiler. The OpenCL compiler for FPGA enables us to reduce the design time of the FPGA-based accelerators. However, the FPGA-based accelerator generated from common OpenCL codes cannot accelerate the processing efficiently in some applications such as an FDTD computation. To accelerate the FDTD computation, global memory access can be reduced by storing the small partition of the electronic and magnetic fields with enclosed areas into the local memory. According to the result of the implementation of the FDTD accelerator on the FPGA, the processing speed with overlapped tiling is far faster than that without overlapped tiling. Moreover, the processing speed is faster than a GPU when the number of grids is small.*

Keywords: FPGA, OpenCL, FDTD method, Hardware accelerator, Ovrerlapped tiling

1. Introduction

Recently, very large scale computing systems are required for processing three-dimensional image processing, electromagnetic simulation, fluid dynamics and DNA sequence and so on. However, the power consumption of high performance computer systems increases year by year. Low power and high performance systems for big data processing are strongly required. FPGA-based accelerators are attracting attention for such high-performance computing systems. The power consumption of FPGAs is about one tenth of that of GPUs. Moreover, very large scale architectures for high performance computing can be implemented on a FPGA because of the advancement of the process technology. One of the major problems of the FPGA-based accelerator is a difficulty of designing the architecture. The software-based design on CPUs and GPUs requires only a software code by using C language or CUDA [1]. On the other hand, the hardware-based design on FPGAs requires the design of circuit modules for calculations, controls and connecting to the host PC by using a hardware design language(HDL). To get the good performance on the FPGA-based accelerator, the knowledge of the circuit design and a very long time for designing the hardware are required.

To solve this problem, Altera Corporation released Altera SDK for OpenCL [2] which is the OpenCL compiler for FPGAs. OpenCL is the programming language for multicore architectures, which is standardized by the Khronos group [3]. The source code of the OpenCL is constituted by a host code and kernels. The initialization, the data-transfer from the host PC to the accelerator and running the kernels are described in the host code. The parallelized computation on the accelerator is described in the kernel code. As a feature of the OpenCL, a common source code can be run on different architectures, such as multicore CPUs, GPUs, Intel Phi processors, CELL processors and so on, by using suitable compilers for each architecture. In order to implement an OpenCL code on the FPGA board, Altera SDK for OpenCL can be used as shown in Fig.1. This compiler generates FPGA-based hardwares for calculation and connecting to the host PC by PCI express automatically from the OpenCL code. Hence, we can implement the FPGA-based accelerators without the HDL design. Figure 2 shows the architecture model generated by Altera SDK for OpenCL. This architecture has kernel pipelines, a memory controller, a PCI express controller and interconnections. Compared with conventional accelerators such as GPUs, this architecture has a high degree of

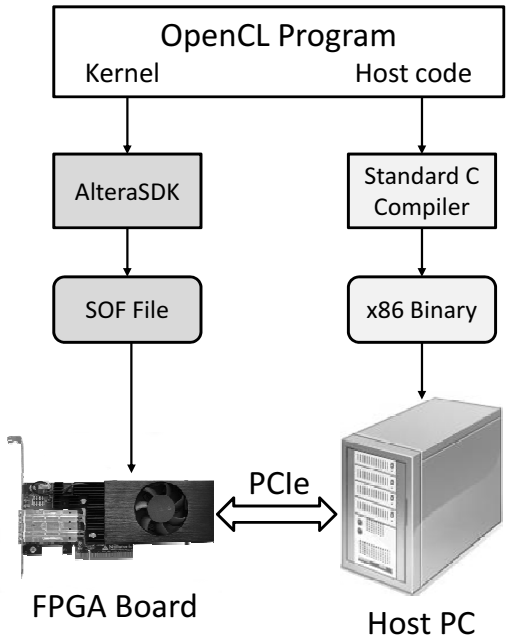


Fig. 1: OpenCL implementation on the FPGA

freedom for changing and optimizing the architecture of the kernel pipelines and interconnections. Altera SDK for OpenCL is used in recent studies such as fractal image processing [4], AES encryption encoding [5] and information filtering [6]. These studies achieved low power and high performance computing compared with CPUs and GPUs. In our previous work [7], we implemented the FDTD method accelerator by using Altera SDK for OpenCL. However the processing speed of the FPGA-based accelerator was slower than that of the GPU. In this paper, we improve the OpenCL code for the FDTD method in order to achieve the better performance on the FPGA. We introduce the overlapped tiling in the electromagnetic field to reduce the global memory access.

2. Implementation of the FDTD method by using OpenCL

The FDTD method [8] has been widely used in electromagnetic simulation, analysis of sound wave and so on. Since the computation of the FDTD method has a high degree of parallelism, there are many works which use computer clusters, GPUs [9], [10] and FPGAs [11], [12] to accelerate the FDTD method. Equation (1) shows the electric field computation and Eqs.(2) and (3) show the magnetic field computation. Electric and magnetic fields in x, y, z directions are denoted by

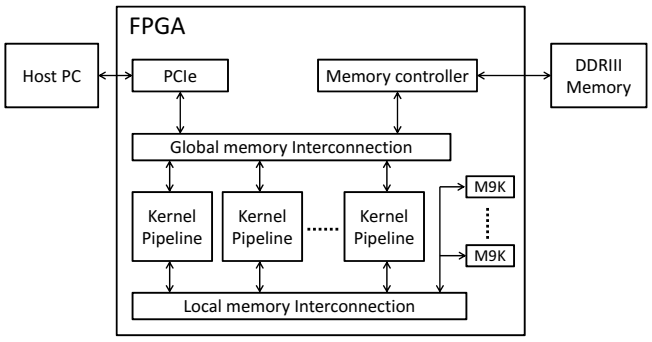


Fig. 2: The architecture model generated by Altera SDK for OpenCL

E and H respectively. The time step is denoted by n and the coordinates of the 2D fields are denoted by i and j. Note that the boundaries of the electric and magnetic fields are calculated differently. Parameters Px, Py, Qx, Qy are determined by the permittivity, the permeability, the size of grids, and the length of the time step. A detailed description of the FDTD method is given in [8].

$$\begin{aligned} E_z^{n+1}(i,j) &= E_z^n(i,j) \\ &-P_y(i,j)\left\{H_x^{n+\frac{1}{2}}(i,j+1/2) - H_x^{n+\frac{1}{2}}(i,j-1/2)\right\} \\ &+P_x(i,j)\left\{H_y^{n+\frac{1}{2}}(i+1/2,j) - H_y^{n+\frac{1}{2}}(i-1/2,j)\right\} \end{aligned} \tag{1}$$

$$\begin{aligned} H_x^{n+\frac{1}{2}}(i,j+1/2) &= H_x^{n-\frac{1}{2}}(i,j+1/2) \\ &-Q_y(i,j)\left\{E_z^n(i,j+1) - E_z^n(i,j)\right\} \end{aligned} \tag{2}$$

$$\begin{aligned} H_y^{n+\frac{1}{2}}(i+1/2,j) &= H_y^{n-\frac{1}{2}}(i+1/2,j) \\ &-Q_x(i,j)\left\{E_z^n(i+1,j) - E_z^n(i,j)\right\} \end{aligned} \tag{3}$$

In our previous OpenCL code in [7], the electric and magnetic field data in the global memory are accessed in parallel. Hence the performance of the execution of this kernel strongly depends on the bandwidth of the global memory.

However, the bandwidth of the global memory on the FPGA is narrower than that of the GPU as shown in Table 1. Hence the FPGA accelerator cannot achieve better performance than the GPU. To achieve high performance computing on the FPGA board, it is important to improve the OpenCL code by reducing the global memory access.

Table 1: Bandwidth of the global memory

FPGA (StratixV 5SGXA7)	GPU(Geforce GTX 580)
25.6GB/s	192.4GB/s

To reduce the global memory access, we introduce the overlapped tiling as described in section 3.

3. Overlapped tiling

Ovelapped tiling is one of the advanced techniques of the loop tiling. Loop tiling divides a big loop into smaller loops to optimize the cache hit rating [13]. This technique is widely used in the stencil computation includes the FDTD computation [14],[15]. In the stencil computation, the value of neighbor grids are used for the computing the value at the next time step as shown in Fig 3. In order to reduce the global memory access by using the loop tilling, the grid data in enclosed area of a tile must be also stored into the local memory.

Hence the overlapped tiling is proposed in order to reduce the communication overhead [16]. Figure 4 shows an example of the overlapped tiling model of the FDTD computation. Let $n \times m$ be the area of the tile and t be the iteration of time steps with the local memory access, $(n+2t) \times (m+2t)$ grids in the electric and magnetic fields are stored into the local memory. This enclosed area of the tile is often called "ghost zone" [17], and the ghost zone is overlapped neighbor tiles. The area of a ghost zone expands as the iteration of the time steps with the local memory access. The FDTD computations in this area are done without the global memory access. After the FDTD computations finish, the values in the tile area is stored into the global memory. The overlapped tiling is often used in the stencil computation on GPUs since the overhead of the synchronization between processing elements can be reduced [17], [18].

Figure 5 shows the flowchart of the FDTD computation with overtapped tiling. The electric and magnetic field data in the tile with the ghost zone is transferred from the global memory to the local memory. Then the FDTD computations in the tile are done prescribe time steps. These FDTD computations are fully pipelined with the pragma of loop unrolling [2], [19]. After the FDTD computations finish, the electric and magnetic field data in the tile are transferred from the local memory to the global memory.

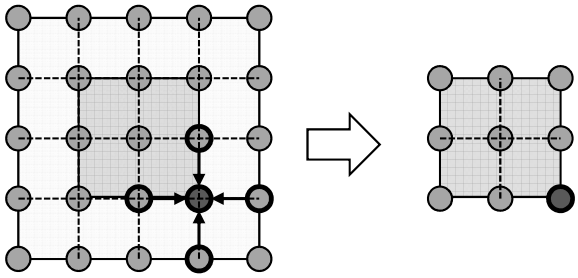


Fig. 3: Stencil computing with neighbor grids

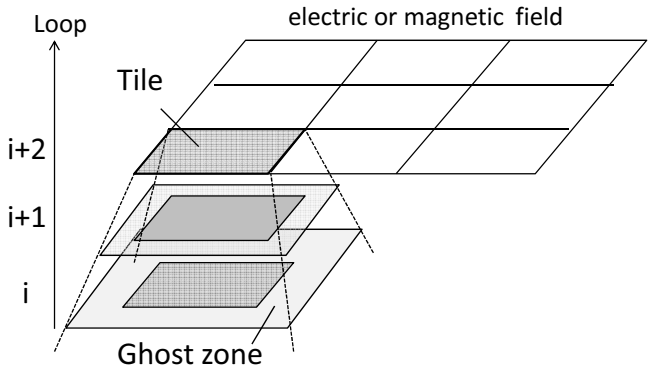


Fig. 4: Overlapped tiling

4. Evaluation

We implement the FDTD accelerator by using OpenCL on "Nallatech P395-D8 FPGA board" [20]. This FPGA board has the Altera StratixV GX C8, four DDR3-SDRAMs(8GB×4) and a PCI-Express. We use Altera SDK for OpenCL 13.1 for the compilation on the FPGA.

Figure 6(a) shows the simulation model. This model has N × N grids (N=128,256,512). The area of tiles as shown in Fig. 4 is 32 × 8 grids. The electric field at (N/2,N/2) is excited as shown in Fig.6(b). The boundary area is considered as the perfect conductor ($Ez = 0$). The single-precision floating-point is used for the calculation.

To optimize the architecture of the FDTD accelerator, it is important to consider with the tradeoff between the amount of the computation and the times of the global memory access. This tradeoff depends on the iterations of time steps with local memory access. Hence we evaluate a relationship between the performance on the FPGA and the iterations of time steps with local memory access.

Table 2 shows the resource usage of the FDTD accelerators with one karnel pipeline as shown in Fig.

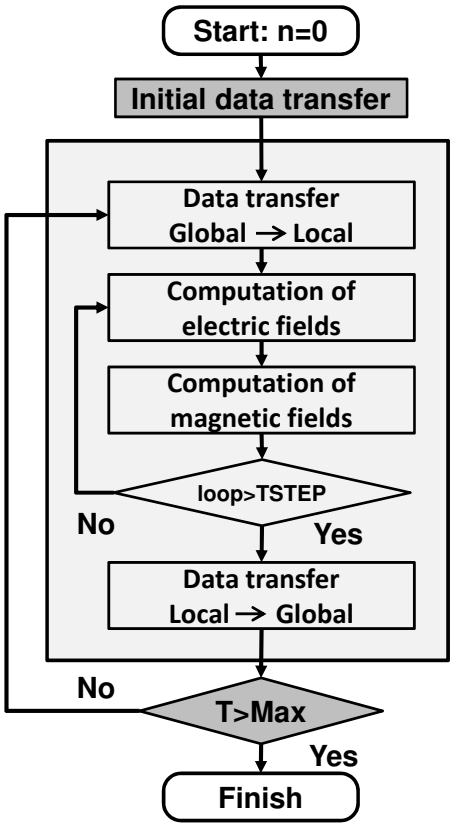


Fig. 5: The flowchart of the FDTD method with overlapped tiling

2. As the time step iterations with local memory access increase, the resource usage becomes larger since the number of the pipeline stages on the kernel pipeline increases by using the loop unrolling.

Figures 7(a) and 7(b) show the relationship of the processing time and iterations of the time steps with local memory access (TSTEP_LOOP) on the FPGA and the GPU (nvidia Geforce GTX 580). As shown in these figures, the processing time on the FPGA is the smallest when TSTEP_LOOP is five. Moreover, most of the processing time of the GPU is larger than that of the FPGA. One of the reasons is that the tile size is not suitable for the warp size on the GPU. These results show that the OpenCL code which is suitable for FPGAs is not always suitable for GPUs.

Table3 shows the comparison of the processing time on the FPGA. The processing speed of the FPGA with overlapped tiling is about 5-20 times as fast as that of the FPGA without overlapped tiling. This result shows that it is effective for accelerate of the FDTD computation to reduce the global memory access by using the overlapped tiling.

Table 3: Processing time of the FPGA with overlapped tiling(s) (Time steps=1000)

Grids	FPGA without tiling	FPGA with tiling
128×128	2.030	0.070
256×256	2.780	0.250
512×512	5.400	1.050

Table 4: Processing time of the CPU, the GPU and the FPGA(s) (Time steps=1000)

Grids	CPU (Corei7 920)	GPU (GTX 580)	FPGA (P395-D8)
128×128	0.249	0.156	0.070
256×256	1.294	0.203	0.250
512×512	11.232	0.249	1.050

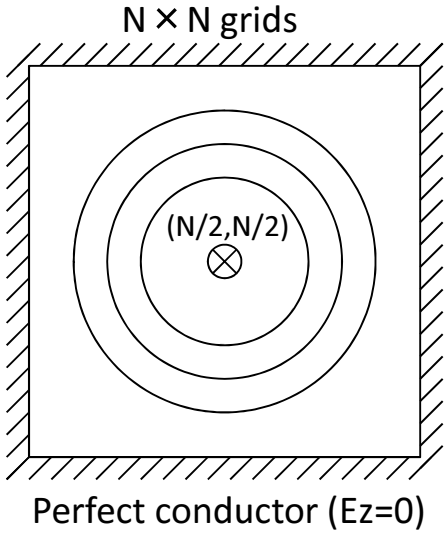
To compare the processing speed of the FPGA-based accelerator with that of CPUs and GPUs, we implement the FDTD method by C language on "Intel Corei7 920", and by OpenCL without tiling on "nvidia Geforce GTX 580". Table 4 shows the comparison of the processing time on the FPGA and the CPU and GPU. When the number of grids is small, the processing speed of the FPGA is the fastest in all devices. On the other hand, the processing speed of the FPGA is slower than that of the GPU when the number of grids is large. In order to get better performance on the FPGA, the degree of parallelism should be increased by implementing more kernel pipelines.

5. Conclusion

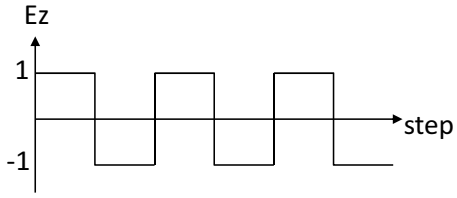
In this article, we implement the FDTD computing with overlapped tiling on the FPGA board by using the OpenCL compiler. The processing speed of the FPGA with overlapped tiling is about 5-20 times as fast as that of the FPGA without overlapped tiling. Moreover, the processing speed of the FPGA is faster than that of GPU when the number of grids is small. To optimize the performance on the FPGA-based architecture, it is important to estimate the performance from the design parameters such as the iterations of time steps with local memory access and the area of the tiles. In future works, we formulate the estimation of the processing time from input design parameters and the optimal design parameters are chosen for implementing the best architecture on the FPGA.

Table 2: Resource usage

Time steps	LEs	FFs	DSPs	RAMs
3	105906(20%)	156517(15%)	12(1%)	1225(48%)
5	144072(26%)	187679(18%)	20(1%)	1403(55%)
6	148848(28%)	201450(19%)	24(1%)	1563(61%)



(a) Simulation model



(b) Excitation of the electric field

Fig. 6: Set up of the simulation

Acknowledgment

This sutdy is supportted by MEXT KAKENHI grant number 24300013 and Grant-in-Aid for JSPS Fellows grant number 15J04973. Also, this study is supported by OTB Transnational Inc.

References

[1] NVIDIA Corporation, "NVIDIA CUDA Programming Guide" Ver2.2.1, 2009.

[2] Altera corpolation, "Altera SDK for OpenCL Programming Guide", http://www.altera.co.jp/literature/hb/opencl-sdk/aocl_programming_guide.pdf

[3] Khronos group, http://www.khronos.org/opencl/

[4] D. Chen and D. Singh, "Fractal Video Compression in OpenCL:An Evaluation of CPUs, GPUs, and FPGAs as Acceleration Platforms", Design Automation Conference (ASP-DAC) 18th Asia and South Pacific, pp.297-304, 2013.

[5] Nallatec, "40Gbit AES Encryption Using OpenCL and FPGAs", http://www.nallatech.com/images/stories/technical_library/white-papers/40_gbit_aes_encryption_using_opencl_and_fpgas_final.pdf

[6] D. Chen and D. Singh, "Using OpenCL to Evaluate the Efficiency of CPUs, GPUs, and FPGAs for Information Filtering ", Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on. IEEE, pp.5-12, 2012.

[7] Yasuhiro Takei, Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Michitaka Kameyama, "Design of an FPGA-Based FDTD Accelerator Using OpenCL ", International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pp.371-375, 2014

[8] H. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media", IEEE Transactions on Antennas and Propagation, Vol.14, No.3, pp.302-307, 1966.

[9] Z. Bo, X. Zheng-hui, R. Wu, L. Wei-ming and S. Xin-qing, "Accelerating FDTD algorithm using GPU computing", International Conference on Microwave Technology & Computational Electromagnetics (ICMTCE), pp.410-413, 2011.

[10] T. Nagaoka and S. Watanabe, "A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis", International Conference on Engineering in Medicine and Biology Society (EMBC), pp.327-330, 2010.

[11] W. Chen, P. Kosmas, M. Lesser and C. Rappaport, "An FPGA Implementation of the Two Dimensional Finite Difference Time Domain (FDTD) Algorithm", ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA), pp.213-222, 2004.

[12] K. Sano, Y. Hatsuda, W. Luzhou and S. Yamamoto, "Performance Evaluation of Finite-Difference Time-Domain (FDTD) Computation Accelerated by FPGA-based Custom Computing Machine", Interdisciplinary Information Sciences, Vol.15, No.1, pp.67-78, 2009.

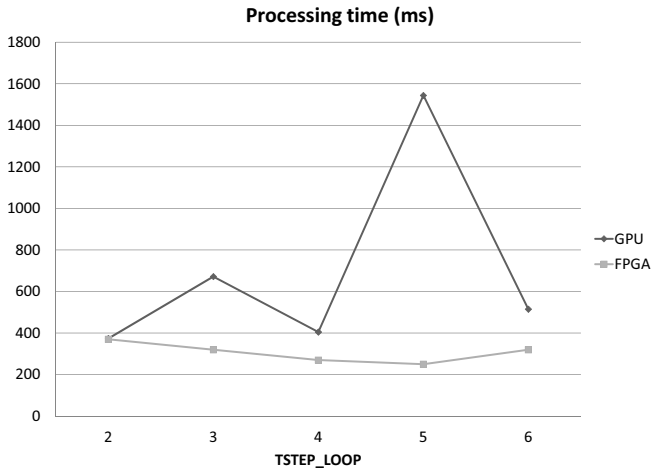
[13] M.E. Wolf and M.S. Lam, 1991 "A Data Locality Optimizing Algorithm", ACM Sigplan Notices, pp.30-44, 1991.

[14] G. Rivera and C. Tseng "Tiling Optimizations for 3D Scientific Computations", ACM/IEEE SC2000 Conference, 2000.

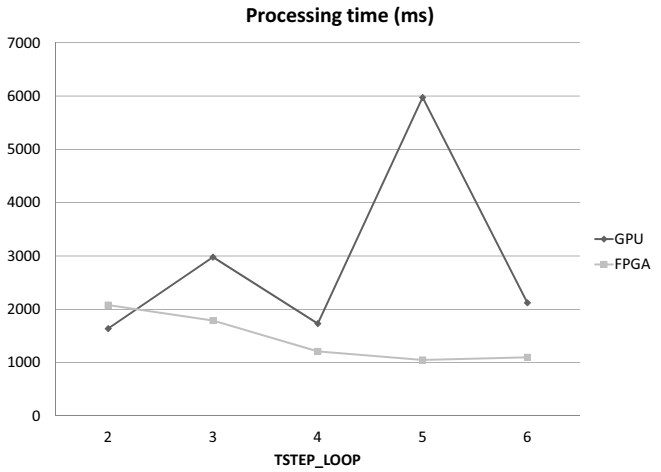
[15] Z. Li , Y. Song, "Automatic Tiling of Iterative Stencil Loops", ACM Transactions on Programming Languages and Systems , pp.975-1028, 2004.

[16] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, P. Sadayappan, "Effective automatic Parallelization of Stencil Computations", ACM SIGPLAN conference on Programming language design and implementation, pp.235-244, 2007.

[17] J.Meng, and K. Skadron, "A Performance Study for Iterative



(a) (N=256)



(b) (N=512)

Fig. 7: Processing time vs time step iterations on local memory

Stencil Loops on GPUs with Ghost Zone Optimizations", International Journal of Parallel Programming 39.1,pp115-142,2011.

[18] J. Holewinski, L.-N. Pouchet, and P. Sadayappan, "High-Performance Code Generation for Stencil Computations on GPU Architectures", In Proceedings of the 26th ACM international conference on Supercomputing (ICS '12), pp.311-320, 2012

[19] Altera corpolation, "Altera SDK for OpenCL Optimization Guide", http://www.altera.co.jp/literature/hb/opencl-sdk/aocl_optimization_guide.pdf

[20] Nallatec, "OpenCL FPGA Accelerator Cards", http://www.nallatech.com/opencl-fpga-accelerator-cards.html

Very Large Scale ReliefF Algorithm on GPU for Genome-Wide Association Study

Kwan-Yeung Lee[1,*], **Pengfei Liu**[2,*], **Kwong-Sak Leung**[3] **and Man-Hon Wong**[4]
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

Abstract— *The advancement in DNA sequencing technology has led to an information bloom on sequencing data and the rise of new data-driven researches like genome-wide association study (GWAS). One major challenge in GWAS is to identify a small set of disease associated single nucleotide polymorphisms (SNPs) out of millions of them in the human genome. Feature selection is a popular technique to reduce the number of features which can boost the efficiency of follow-up analyses. In this paper, we optimized a feature selection algorithm specifically designed for GWAS called Very Large Scale ReliefF (VLSRF) through General-Purpose Computing on Graphics Processing Unit (GPGPU). Experimental results showed that our GPU-based VLSRF and ReliefF achieved up to 100 times speed up relative to a CPU based ReliefF on synthetic datasets. On the other hand, our GPU-based VLSRF was tested on a real dataset from a GWAS Parkinson's disease study in the National Human Genome Research Institute [1]. The SNPs identified by our GPU-based VLSRF are consistent with existing literature.*

Keywords: GPU, feature selection, VLS Relief, Parallization

1. Introduction

Nowadays, next-generation sequencing technologies allow scientists to sequence a large number of samples in a reasonable time and cost [2], [3] which encourages biologist to perform research in data-driven manner e.g. genome-wide association studies (GWASs). In GWAS, statistical or computational analyses are applied to compare the DNA sequences of healthy samples (controls) and patients of a specific genetic disease (cases) in order to identify single nucleotide polymorphisms (SNPs) [4] that are associated to complex genetic diseases like cancer [5] and hepatitide [6].

A single nucleotide polymorphism (SNP) is a genetic variation of a single nucleotide at a specific location of the DNA sequences across the samples in a population. As human is a diploid and each chromosome has two different copies, the genotype of a SNP is composed by two alleles (nucleotides). A SNP can be associated to a genetic disease as a single SNP or as a SNP-SNP interaction. SNPs which are associated to a genetic disease independently could be found through statistical test like p-value testing. Although most of the independent disease associated SNPs were found in various research, those SNPs alone were insufficient in explaining the heritability of those genetic diseases they were associated to. One possible reason is that some disease associated SNP-SNP interactions remain undiscovered. However, it is difficult to detect these interactions. It is well-known that independent marginal effect of each SNP in most SNP-SNP interactions is small [7], [8]. Therefore non-linear SNP-SNP interactions can model the cause of a genetic disease better than linear models. However, detecting non-linear disease-associated SNP-SNP interactions suffers from the curse of dimensionality. One possible approach to improve the efficiency is to reduce the search space through carefully selecting a subset of SNPs that have high potential to be disease associated.

Relief-F is one of the most popular feature selection algorithms in analyzing GWAS datasets. Relief-F is proved to be better in selecting the interacting SNPs than traditional feature selection algorithms like chi-square or information gain which are based on the statistic of each individual feature [9]. Since the accuracy of ReliefF does not scale up well under the large number of SNPs in real GWAS datasets, an enhanced version of Relief-F called Very Large Scale Relief-F (VLSRF) is developed to improve its accuracy. However, VLSRF had a significant higher time complexity than its predecessor [10].

Since ReliefF and VLSRF has a high potential to be parallelized, we propose a GPU implementation on them using CUDA to significantly reduce the time for biologists in prioritizing a list of potential disease associated SNPs for their researches. We believe this can directly accelerate the process of finding SNPs and/or genes that relating to diseases and designing new drugs and diagnosis.

GPU is a high-performance multi-core processor with high computation and data throughput [11]. A modern GPU can perform significantly faster than a CPU in many complex operations like vector and matrix operations or floating point arithmetics. Hence, it has become a popular computing solution for analyses on experimental data in various areas, such as weather forecasting [12], molecular dynamics [13] and fluid-flow [14]. Moreover, it can be programmed easily to perform general purpose computation through high-level programming language like CUDA or OpenCL. CUDA is shown to perform better than OpenCL [15], and CUDA is capable of handling large biological experimental data [16].

*These two authors contribute equally to the work.

Table 1: Encoding Table for SNP Genotype

Orginal Genotype (A = major allele, a = minor allele)	Encoded Value
Missing	0
AA	1
Aa	2
aa	3

In most cases, an allele of a SNP could only has two nucleotide types. A major allele has a nucleotide type that appears more frequently in its corresponding SNP across the population while a minor allele has a nucleotide type that appears less frequently in its corresponding SNP across the population.

In this paper, we are developing a GPU-based VLSRF implementation optimized for GWAS dataset. By exploiting the parallelization power of CUDA, reducing number of feature subsets through a novel feature subset enumeration process and reducing the time cost and space used in storing and loading the dataset through simple compression, our GPU-based VLSRF out-performed CPU-based Relief and CPU-based VLSRF in terms of both performance and accuracy. Furthermore, our experiments demonstrated that our GPU-based VLSRF could detect disease associated SNPs in a real Parkinson's disease dataset.

Here is the layout of this paper. First, related work will be given in section 3. Then, details of our implementation will be discussed in section 4. Finally, experimental results will be shown and analyzed in section 5.

2. Problem definition

Input: A dataset that stores the genotypes of a common list of features (SNPs) of a number of case and control samples. The genotype of each SNP is encoded through the schema shown in table 1.
Output: An association score for each SNP against the target genetic disease of the inputted dataset.

3. Related Work

With the increasing popularity of data mining, the application of feature selection has been widen in various research area [17]–[21]. Generally speaking, there are two different kinds of feature selection algorithms. The first kind utilized one or more statistical properties of each feature such as information entropy and t-test value, to search for potential useful features [22]. The second kind exploits the topology or the structure in the data space [23]–[25] to search for potentially important features by measuring the inner relationships between samples and features.

As SNP-SNP interactions are most likely to be non-linear, pure statistical algorithms which measured the statistical property of each SNP perform poorly in GWAS. Meanwhile structural based algorithms like Relief algorithm has been proven to be efficient and effective in prioritizing SNPs in synthetic GWAS datasets [9].

```
Relief Algorithm
1  Initialize feature score vector w
2  for n = 1 : no. of iteration pre-defined
3      Randomly select a sample x from sample set S
4      Find the nearest hit NH_x and nearest miss NM_x of x
5      for i =  1 : no. of features
6          w[i] = w[i] + |x[i] – NM_x[i]| - |x[i] – NH_x[i]|
7      end
8  end
```

Fig. 1: Pseudocode of Relief Algorithm

Relief is a probabilistic algorithm which evaluates every feature in a dataset in the following steps [24]. First, a sample χ is randomly selected and its pairwise distances with other samples are calculated. Second, the nearest neighbor with the same phenotype (hit) or nearest neighbor with a different phenotype (miss) are found. Third for each feature, its value in the selected sample χ is compared against its value in the corresponding nearest hit and miss. Features that are more consistent within samples from the same class and inconsistent across samples from different classes will obtain a higher score. Otherwise, it will have a lower score. Finally, the score of each feature is updated through repeating the three steps above multiple times. Its pseudocode is shown in figure 1.

On the other hand, ReliefF is a Relief based algorithm which can handle datasets with multiple classes and continuous value features while maintaining a similar time complexity as its predecessor [25]. The major improvement of ReliefF over its predecessor is that it compares each randomly selected sample against a small group of nearest hit and miss during the feature scoring process instead of only one nearest hit and miss. This eliminates the aversive effects of outliners and has improved the overall accuracy. Moreover, ReliefF was further enhanced and a new algorithm called Tuned ReliefF was developed [9]. Tuned ReliefF achieved a higher detection power than its predecessor through repeatedly executing ReliefF algorithm and gradually removing a small number of low ReliefF score features after each ReliefF execution. As ReliefF is executed iteratively, Tuned ReliefF is significant slower than ReliefF algorithm.

Iterative ReliefF is another Relief based algorithm [26]. Similar to ReliefF algorithm, it compares each randomly selected sample to more than one nearest hit and one nearest miss. First, it calculates the probability of each randomly selected sample χ being a outliner and the probability of other sample being the nearest hit or the nearest miss of

```
VLSReliefF Algorithm
1  Initialize global feature score vector w
2  for i = 1 : no. of feature subset
3      Enumerate the i^th feature subset S
4      Form a reduced dataset D' with features in S only
5      Perform ReliefF on D' to calculate local score w'
6      for i =  1 : no. of features
7          w[i] = max(w[i], w'[i])
8      end
9  end
```

Fig. 2: Pseudocode of VLS ReliefF Algorithm.

χ. After that, it compares each randomly selected sample χ against all other samples and updates the score of each feature according to the nearest hit/miss probability of other samples and outlining probability of the selected sample previously calculated. The above two steps will be repeated until the scores of all features converge.

Very Large Scale ReliefF (VLSRF) selects important features from a dataset with enormous number of features through divide and conquer [10]. First, it splits the original feature set into a full collection of feature subsets with a user specific size. The size of feature subset is significantly smaller than the original feature set to ensure that ReliefF can maintain a high detection power on those subsets. Second, the local score of every feature under each feature subset is calculated independently through ReliefF algorithm. Finally, all local feature scores calculated by each independent ReliefF execution are merged and the final global score of each feature is its maximum local score. Although VLSRF maintains a high detection accuracy under extremely large number of features, it has a much higher time complexity than ReliefF as a trade off. To ensure at least one feature subset contains all important features and such that their association can be measured, an exponential number of feature subsets are needed to be evaluated. Its pseudocode is shown in figure 2.

4. Implementation Detail

In this section, we will describe the details of our GPU-based VLSRF implementation. The overall flow chart of this implementation is shown in figure 3.

4.1 Data Compression

As a SNP has at most 3 different genotypes, every feature in a GWAS dataset has a very small domain and 2 bits are sufficient for representing any genotype of a SNP. Therefore, any dataset inputted will be compressed through converting every feature into a 2 bit integer and tightly packing every 4 features into a byte before storing into the host main memory and GPU device memory. This reduces the amount of memory required for storing the dataset significantly and allowed GPU with a limited device memory can store a larger dataset. It also reduced the run-time overhead in copying the dataset from main memory to GPU device memory. Finally, only a constant number of binary operations are needed for accessing the value of a feature from a compressed dataset, so it maintains the performance of ReliefF while improving the efficiency of loading data into GPU device memory.

4.2 Feature Subset Enumeration

To reduce the number of feature subsets to be evaluated, we proposes a new feature subset formation process. In this new process, all features in the inputted dataset are divided into different groups. A feature subset is then generated by combining several feature groups in stead of randomly selecting a number of features from the original set of features. The details of this approach is shown in figure 3.

We apply a binary operation based subset enumeration algorithm called GosperâĂŹs hack [27] to select feature groups for forming feature subsets. In this algorithm, a binary string is representing the membership of each feature group of a subset and it has a length of the number of feature groups. If the i^{th} bit in the binary string is one, then the i^{th} feature group is a member of the current feature subset. Otherwise, it is not a member of the subset. Furthermore, in this algorithm, a binary operation based enumerator is used to enumerate the membership of feature groups of the next new subset in O(No. of features) binary operations based on a given subset. Therefore, every subset can be generated through iterative use of the enumerator. Therefore, this algorithm minimized the time needed in enumerating each feature subset.

4.3 GPU-Based ReliefF Algorithm

The ReliefF algorithm is implemented to be executed entirely under GPU to minimize the data transfer between the host PC memory and the GPU device memory. Some operations in the ReliefF are optimized through exploiting the parallelization processing power of GPU. First, the distance of a pair of sample does not have dependency with the distance of other pairs of samples and can be calculated independently. Second, a list of randomly selected samples are compared against their corresponding group of nearest hits and group of nearest misses in a feature by feature wise manner. These feature value comparisons are independent from each other and can be performed simultaneously. Therefore, two separated CUDA kernel programs are developed for performing these two operations. These two CUDA kernel programs are executed under multiple GPU threads in parallel automatically.

5. Result and Discussion

In order to evaluate the performance of our GPU based VLSRF implementation, numerous experiments were per-

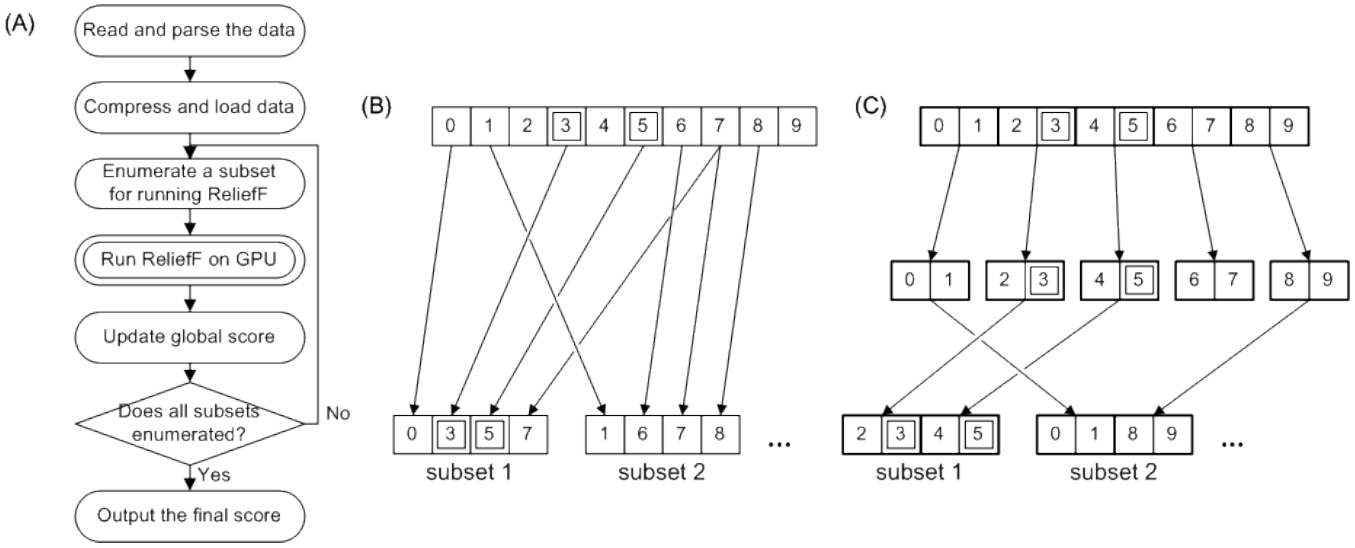


Fig. 3: (A) shows the execution flow of our GPU-based VLS ReliefF. In (A), the block with double line border is performed under GPU and other blocks are executed under CPU. (B) shows an example on the original process of enumerating feature subsets in VLSRF. Under 10 features and process (B), there are totally $\binom{10}{4} = 210$ ways to form feature subsets with 4 features. (C) shows an example on our novel process for enumerating feature subsets in VLSRF under 10 features. First, the original feature set is split into 5 feature groups where each of these group carries 2 features. Second, each feature subset is composed by 2 randomly selected feature groups. As a result, there are $\binom{5}{4} = 5$ total possible feature subsets with size = 4 which is significant fewer than process (B). On the other hand, the new process will still able to select all important features into the same feature subset and ensure at least 1 pass of Relief algorithm can detect these features as long as all possible subsets are tested. As seen in (C), feature '3' and '5' (i.e. the block with double line border) are both important features and they are both the member of subset 1.

formed on various synthetic and real datasets. In this section, we will discuss the results of those experiments in detail.

5.1 Machine Configuration

All experiments on CPU implementations were performed on a workstation with one Intel I5-4670 3.4GHZ CPU and 8GB DDR3 main memory. On the other hand, all experiments on the GPU implementations were performed on a high performance Linux server with two Intel Xeon E5-2670 2.6GHZ CPU, 128GB ECC DDR3 main memory and a NVIDIA GK110GL Kepler GPU.

5.2 Experiment on Synthetic Datasets

We performed experiments to compare the speed and accuracy of our GPU-based VLSRF against CPU-based ReliefF and CPU-based VLSRF. The performance metrics were the ranking of predictive feature and execution time.

5.2.1 Details of Synthetic Dataset Generator

All synthetic datasets were generated by GAMETES which is a dataset generator specifically designed for generating highly accurate GWAS synthetic dataset [28]. We generated four groups of datasets under different heritabilities using GAMETES and each dataset group contained 50 datasets. The configurations on generating those datasets are shown in table 2.

5.3 Runtime Parameters

The parameters used to run the experiments are shown in table 3.

5.4 Predictive feature Ranking Comparison

Numerous experiments were conducted to compare the accuracy between ReliefF and VLSRF. Features in each dataset in the four dataset groups were ranked and the ranking of the two predictive features were compared as the measurement for accuracy. Furthermore, we performed our experiments on VLSRF under two different feature grouping sizes to understand the behaviour of our novel feature subset enumeration procedure. All the results were plotted as charts and shown in figure 4.

In our experiment, for ReliefF all the predictive features were ranked top half in all the experiments. It is clear that ReliefF can differentiate predictive features from other features effectively. The numbers of datasets that the first and second predictive features were ranked top 20% under heritability 0.1, 0.2, 0.3 and 0.4 were 17, 9, 19, and 16 and 9, 18, 25 and 22 out of 50 datasets respectively. Therefore generally speaking, the performance of ReliefF increased with

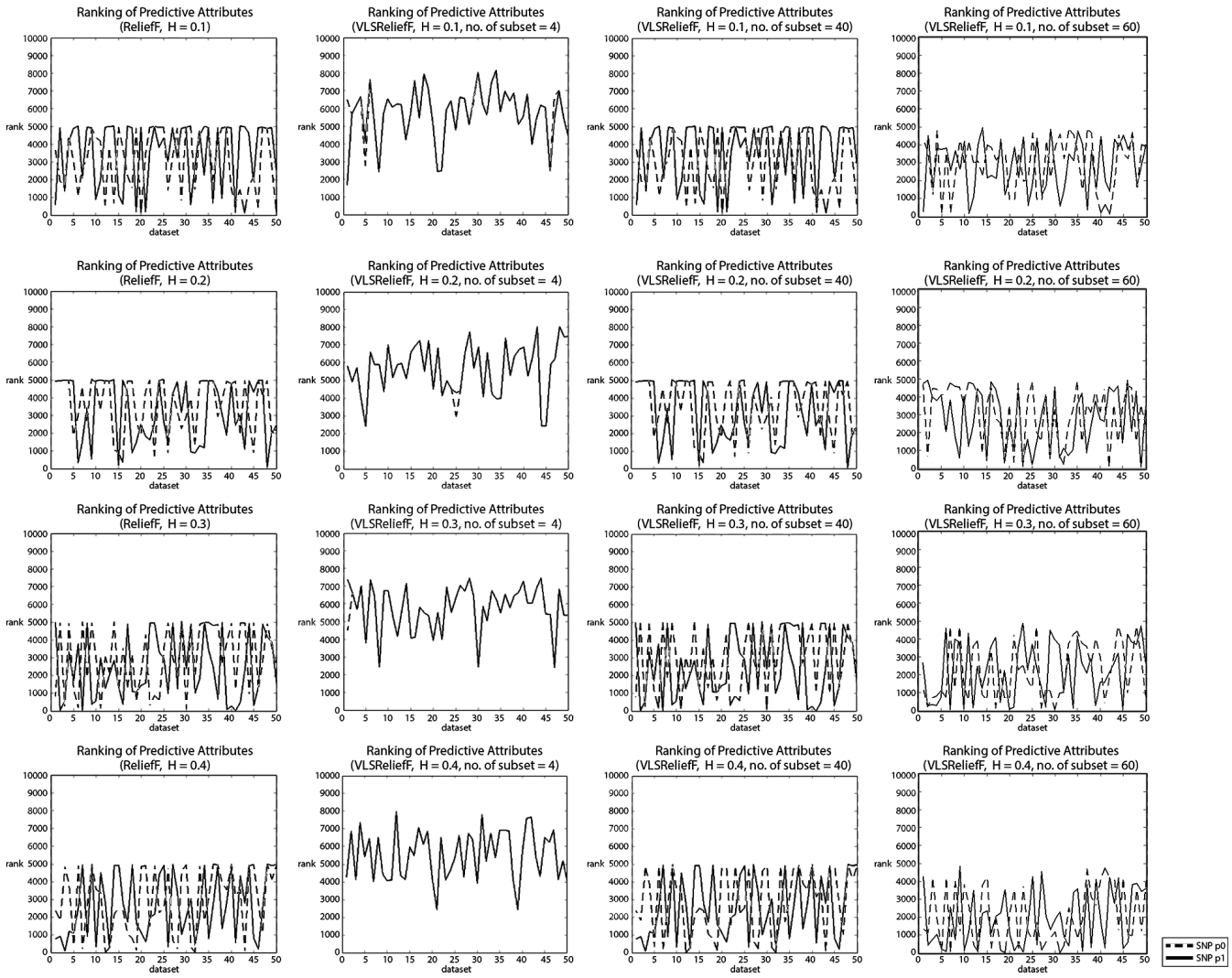


Fig. 4: Ranking of two predictive features of groups of synthetic datasets with different heritabilities under ReliefF and VLS ReliefF algorithm. For each chart, the horizontal axis stands for a dataset and the vertical axis stands for the ranking of predictive features. Moreover, the dotted line and solid line in each chart represent the rankings of the first(p0) and second(p1) predictive feature across the datasets.

the increasing heritability. To conclude, ReliefF performed as expected and it can be used as a benchmark for comparison.

By observing the graph, for VLSReliefF we can see that as the number feature subset increases, the ranks of the two predictive features increases. This shows that under the same number of feature subset enumerated and tested, the scores of the features converge faster with a larger feature group size. This shows that our novel feature subset enumeration procedure is effective in reducing feature subsets needed to be enumerated.

When VLSRF was executed under four feature subsets, only the second predictive feature was ranked top 20% once. Moreover the numbers of datasets that the first and second predictive features were ranked top 20% under heritability 0.1, 0.2, 0.3 and 0.4 were 18, 11, 16, and 16 and 13, 16, 25 and 22 out of 50 datasets respectively if VLSRF is executed under 40 feature subsets. It is obvious that VLSRF do not outperform ReliefF subset in these circumstances. On the other hand, it had a significantly higher accuracy than ReliefF under 60 feature subsets as the numbers of datasets that the first and second predictive features were ranked top 20% under heritability 0.1, 0.2, 0.3 and 0.4 were 15, 13, 26, and 29 and 15, 18, 19 and 26 out of 50 datasets respectively. It is clear that VLSRF can perform better than ReliefF as long as the number of feature subset evaluated is large enough.

Table 2: Parameters for Generating Synthetic Dataset

Parameter	Value	Description
Total No. of features	10000	The number of SNPs in the dataset
Predictive feature No	2 (p0, p1)	The number of disease SNPs in the dataset
Heritability	0.1, 0.2, 0.3, 0.4	Proportion of cases that are associated by disease associated SNP
Population Prevalence	Random	The probability for a random sample having disease
Minor Allele Frequency	0.2	The frequency of allele that occur less in the population
No. of Cases and Controls	1000(500:500)	The number of cases and controls in the dataset
No. of EDM	1	The number of model used in generating the dataset

Table 3: Runtime Parameters for ReliefF and VLSReliefF

Parameter	ReliefF	VLSRF
Number of NNs	1	1
Size of Subset	-	5000
Number of Subset	-	4, 40, 60

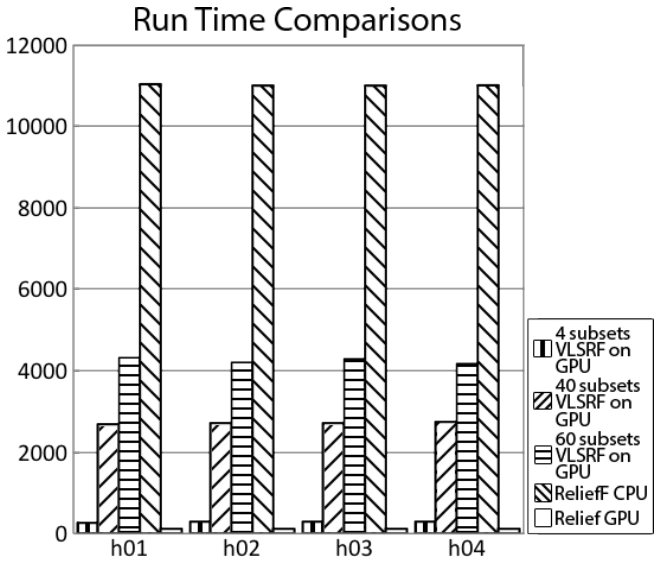


Fig. 5: Run time comparison between CPU and GPU implementations of ReliefF and VLS ReliefF. The horizontal axis stands for heritability of the dataset group and the vertical axis stands for the execution time in seconds.

5.5 Execution Time Comparison

Numerous experiments were conducted to compare the execution time between CPU-based ReliefF and our GPU-based VLSRF. We measured and compared the time needed for both algorithms to finish the execution of the datasets in a dataset group. The experimental time cutoff was 5 hours. All the results were plotted as charts and shown in figure 5.

Our experiments have showed that ReliefF costed around 11,000 seconds and 110 seconds to finish processing all the dataset in a dataset subgroup on CPU and GPU respectively. The GPU version outperformed the CPU version by around 100 times. It is obvious that GPU alone significantly improve the efficiency of ReliefF algorithm.

The run time of VLSRF was not reported as it failed to finish its execution on CPU within the time limit under any number of feature subset we tested. This result is consistent with our understanding of VLSRF. According to section 3, the run time of VLSRF was approximately proportional to the run time of ReliefF by a factor of the feature subset number. Therefore, we could roughly estimate the run time of VLSRF under 4 feature subsets to be 4x11000 = 44000 seconds = 12 hours. Therefore, no CPU based VLSRF could finish its execution before reaching the time limit. On the other hand, GPU-based VLSRF could finish its execution with at most 4300s. This clearly shows that it is far more practical to perform VLSRF on GPU rather than GPU.

The run time of VLSRF under 4, 40 and 60 feature subsets were around 250s, 2700s and 4500s respectively. It is clear that our novel feature selection process is efficient and does not hinder the performance of VLSRF.

5.6 Details of Real Dataset

5.6.1 Dataset Source and Pre-processing

Our GPU-based VLSRF was applied on a Parkinson's disease study called Mayo-Perlegen LEAPS (Linked Efforts to Accelerate Parkinson's Solutions) Collaboration which is originated from National Human Genome Research Institute [1] to evaluate its performance on a real dataset. This study had two tiers and we picked the tier 2 for performing our experiment. Tier 2 had 660 samples (332 case and 332 control) and each sample had 3000 SNPs. Before we performed experiments on this dataset, we had performed data cleansing and converted it into ARFF file format with the encoding scheme described in section 3 such that it could be processed by our programs easily.

5.6.2 Result Analysis

After we computed the score of all SNPs using our GPU-based VLSRF, we ranked those SNPs according to their score and extracted the top 10% SNPs for further analysis. Among those top 10% SNPs, there are 3 SNPs which are associated to Parkinson's disease related gene. These 3 SNPs are rs2287403, rs1979687 and rs2303703 which

are associated to YLPM1, USP47 and MYO10 respectively and these 3 genes were all reported to be associated to Parkinson's disease by a Parkinson's disease database [29]. Furthermore, gene USP47 and MYO10 were separately reported to be associated to Parkinson's disease under 2 more literatures [30], [31]. All these showed that our GPU-based VLSRF algorithm are able to select disease associated SNPs effectively.

6. Conclusion

In this paper, we have demonstrated a new GPU-based VLSRF implementation and tested it thoroughly using multiple sets of synthetic datasets and a real GWAS dataset on Parkinson's disease. Under various experiments, our GPU-based VLSRF was shown to be an effective and efficient algorithm in identifying important SNPs. It outperformed existing CPU based VLSRF and ReliefF in term of speed while maintaining a comparable accuracy with them. Under the experiment of Parkinson's disease dataset, it has identified SNPs consistent to existing literature. All these show GPU-based VLSRF is effective and efficient under both real and synthetic datasets.

References

[1] D. M. Maraganore, M. de Andrade, T. G. Lesnick, K. J. Strain, M. J. Farrer, W. A. Rocca, P. K. Pant, K. A. Frazer, D. R. Cox, and D. G. Ballinger, "High-resolution whole-genome association study of parkinson disease," *The American Journal of Human Genetics*, vol. 77, no. 5, pp. 685–693, 2005.

[2] J. N. Hirschhorn and M. J. Daly, "Genome-wide association studies for common diseases and complex traits," *Nature Reviews Genetics*, vol. 6, no. 2, pp. 95–108, 2005.

[3] W. Y. Wang, B. J. Barratt, D. G. Clayton, and J. A. Todd, "Genome-wide association studies: theoretical and practical concerns," *Nature Reviews Genetics*, vol. 6, no. 2, pp. 109–118, 2005.

[4] D. E. Reich and E. S. Lander, "On the allelic spectrum of human disease," *TRENDS in Genetics*, vol. 17, no. 9, pp. 502–510, 2001.

[5] D. F. Easton and R. A. Eeles, "Genome-wide association studies in cancer," *Human Molecular Genetics*, vol. 17, no. R2, pp. R109–R115, 2008.

[6] N. P. Paynter, D. I. Chasman, G. Paré, J. E. Buring, N. R. Cook, J. P. Miletich, and P. M. Ridker, "Association between a literature-based genetic risk score and cardiovascular events in women," *Jama*, vol. 303, no. 7, pp. 631–637, 2010.

[7] J. H. Moore, F. W. Asselbergs, and S. M. Williams, "Bioinformatics challenges for genome-wide association studies," *Bioinformatics*, vol. 26, no. 4, pp. 445–455, 2010.

[8] J. H. Moore, "The ubiquitous nature of epistasis in determining susceptibility to common human diseases," *Human heredity*, vol. 56, no. 1-3, pp. 73–82, 2003.

[9] J. H. Moore and B. C. White, "Tuning relieff for genome-wide genetic analysis," in *Evolutionary computation, machine learning and data mining in bioinformatics*. Springer, 2007, pp. 166–175.

[10] M. J. Eppstein and P. Haake, "Very large scale relieff for genome-wide association analysis," in *Computational Intelligence in Bioinformatics and Computational Biology, 2008. CIBCB'08. IEEE Symposium on*. IEEE, 2008, pp. 112–119.

[11] M. Harris and D. Luebke, "Gpgpu: General-purpose computation on graphics hardware," in *International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2005 Courses: Los Angeles, California*, vol. 2005, 2005.

[12] J. Michalakes and M. Vachharajani, "Gpu acceleration of numerical weather prediction," *Parallel Processing Letters*, vol. 18, no. 04, pp. 531–548, 2008.

[13] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," *Journal of Computational Physics*, vol. 227, no. 10, pp. 5342–5359, 2008.

[14] P. Bailey, J. Myre, S. D. Walsh, D. J. Lilja, and M. O. Saar, "Accelerating lattice boltzmann fluid flow simulations using graphics processors," in *Parallel Processing, 2009. ICPP'09. International Conference on*. IEEE, 2009, pp. 550–557.

[15] K. Karimi, N. G. Dickson, and F. Hamze, "A performance comparison of cuda and opencl," *arXiv preprint arXiv:1005.2581*, 2010.

[16] R. Jiang, F. Zeng, W. Zhang, X. Wu, and Z. Yu, "Accelerating genome-wide association studies using cuda compatible graphics processing units," in *Bioinformatics, Systems Biology and Intelligent Computing, 2009. IJCBS'09. International Joint Conference on*. IEEE, 2009, pp. 70–76.

[17] S. Beniwal and J. Arora, "Classification and feature selection techniques in data mining," in *International Journal of Engineering Research and Technology*, vol. 1, no. 6 (August-2012). ESRSA Publications, 2012.

[18] F. Min, Q. Hu, and W. Zhu, "Feature selection with test cost constraint," *International Journal of Approximate Reasoning*, vol. 55, no. 1, pp. 167–179, 2014.

[19] J. Tang and H. Liu, "Unsupervised feature selection for linked social media data," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 904–912.

[20] ——, "Feature selection with linked data in social media." in *SDM*. SIAM, 2012, pp. 118–128.

[21] C.-P. Lee and Y. Leu, "A novel hybrid feature selection method for microarray data analysis," *Applied Soft Computing*, vol. 11, no. 1, pp. 208–213, 2011.

[22] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: a unifying framework for information theoretic feature selection," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 27–66, 2012.

[23] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of the ninth international workshop on Machine learning*, 1992, pp. 249–256.

[24] ——, "The feature selection problem: Traditional methods and a new algorithm," in *AAAI*, vol. 2, 1992, pp. 129–134.

[25] I. Kononenko, E. Šimec, and M. Robnik-Šikonja, "Overcoming the myopia of inductive learning algorithms with relieff," *Applied Intelligence*, vol. 7, no. 1, pp. 39–55, 1997.

[26] Y. Sun, "Iterative relief for feature weighting: algorithms, theories, and applications," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1035–1051, 2007.

[27] R. W. Gosper, "Item 175 in beeler, m., gosper, rw, and schroeppel, r., hakmem," 1972.

[28] R. J. Urbanowicz, J. Kiralis, N. A. Sinnott-Armstrong, T. Heberling, J. M. Fisher, and J. H. Moore, "Gametes: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures," *BioData mining*, vol. 5, no. 1, pp. 1–14, 2012.

[29] J. O. Yang, W.-Y. Kim, S.-Y. Jeong, J.-H. Oh, S. Jho, J. Bhak, and N.-S. Kim, "Pdbase: a database of parkinson's disease-related genes and genetic variation using substantia nigra ests," *BMC genomics*, vol. 10, no. Suppl 3, p. S32, 2009.

[30] F. Simunovic, M. Yi, Y. Wang, L. Macey, L. T. Brown, A. M. Krichevsky, S. L. Andersen, R. M. Stephens, F. M. Benes, and K. C. Sonntag, "Gene expression profiling of substantia nigra dopamine neurons: further insights into parkinson's disease pathology," *Brain*, vol. 132, no. 7, pp. 1795–1809, 2009.

[31] K. Gousset, L. Marzo, P.-H. Commere, and C. Zurzolo, "Myo10 is a key regulator of tnt formation in neuronal cells," *Journal of cell science*, vol. 126, no. 19, pp. 4424–4435, 2013.

Cloud-dew architecture: realizing the potential of distributed database systems in unreliable networks

Yingwei Wang[1] **and Yi Pan**[2]
[1]Department of Computer Science, University of Prince Edward Island,
Charlottetown, Prince Edward Island, Canada
[2]Department of Computer Science, Georgia State University,
Atlanta, Georgia, United States

Abstract - *Distributed database systems, which continue to inspire new architectures and new applications, have great potential in the modern computing world. In this paper, we show that the newly-proposed cloud-dew architecture realizes the potential of distributed database systems in the unreliable network environment, and provides the possibility of web-surfing without an Internet connection. Distributed database systems are generic and versatile; the proper applications of distributed database systems and their features will be beneficial to users and service providers.*

Keywords: distributed database system; cloud-dew architecture; peer-to-peer; super-peer; transparency

1 Introduction

A distributed database system is defined as a collection of multiple, logically interrelated databases distributed over a computer network [1]. Combined with other components, distributed database systems [1-3] play central roles in various applications. It is believed that the potential of distributed database systems has not been realized fully as yet [1]. The following paragraph describes one of the promising possibilities of distributed database systems:

"The failure of a single site, or the failure of a communication link which makes one or more sites unreachable, is not sufficient to bring down the entire system. In the case of a distributed database, this means that some of the data may be unreachable, but with proper care, users may be permitted to access other parts of the distributed database" [1].

This description suggests that an application may still work when a communication link fails. If the application is a web application and the communication link is an Internet connection, the possibility exists that the web application may still work when an Internet connection is not available. Today, web applications are daily essentials but an Internet connection is not always available. This potential is very attractive.

As indicated in the above paragraph, the great potential cannot be realized automatically, and "proper care" is necessary.

Is the great potential realizable? What is the proper care to realize this great potential? A newly-proposed architecture [4] shows that it is possible to do web-surfing without an Internet connection. In this case, the proper care is the architecture: cloud-dew architecture.

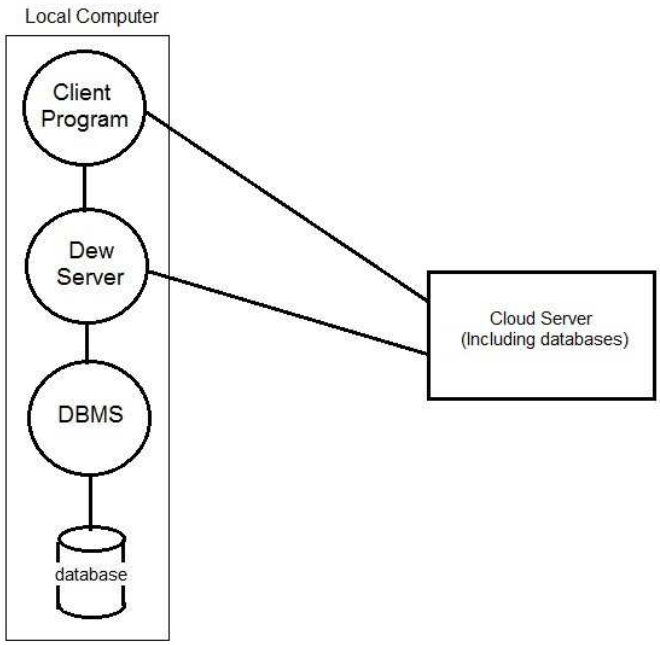


Figure 1: Cloud-dew architecture

Cloud-dew architecture is an extension of the client-server architecture [4]. This architecture is illustrated in Figure 1, and the client-server architecture is depicted in Figure 2 for comparison. A new kind of server, dew server, is introduced in this architecture. A dew server is a web server that resides on a user's local computer. The dew server and its related databases have two functions: first, it provides the client with the same services as the cloud server provides; second, it synchronizes dew server databases with cloud server databases. A dew server has the following features:

(1) A dew server is a lightweight web server. Usually, it serves only one user, the client.

(2) A dew server usually stores only the user's data. The 'size' (i.e., data amount in related databases) of a dew server is much smaller than the 'size' of a cloud server. Metaphorically, a cloud server is as big as a cloud, and a dew server is as small as a drop of dew.

(3) A dew server disappears easily. The dew server's data could disappear for different reasons, for instance: hardware damage and failure or virus infections. Metaphorically, a dew server is as weak as a drop of dew.

(4) A vanished dew server can be recreated because all dew server data has a copy in the cloud servers. Metaphorically, dew will come out again after it disappears as long as a cloud can provide all the necessities.

(5) A dew server is accessible with or without an Internet connection because it is running on the local computer. Metaphorically, a cloud could be far away, but the dew is close to you.

Figure 2: Client-server architecture

Suppose a user stores personal data such as pictures and messages on a website, say http://www.facebook.com. While the data is available publicly, the user cannot access his/her own data if an Internet connection is not available. The user may decide to save a local copy of personal data in his/her own computer. However, saving pictures and messages in files may be awkward and difficult to manage.

Suppose a website, in this case http://www.facebook.com, adopts the cloud-dew architecture. The website will be duplicated onto a dew server running on a user's local computer. The duplication is not exactly copying. Generally speaking, the duplicated website in a dew server (called a dewsite) and the original website could be different in the following aspects:

(1) The dewsite does not need to deal with a global heavy load so that it could be much simpler than the website;

(2) The dewsite will not include the proprietorial script that the website does not want to release. Instead, publicly-known technology will be used to implement similar functionalities;

(3) The content of a dewsite database could be limited;

(4) A new functionality, which will synchronize with the website, will be added to the dewsite.

Once a dewsite duplicating http://www.facebook.com is installed inside a dew server, the user may access the dewsite. A local domain name system (LDNS) could be introduced so that the above-mentioned dewsite can be accessed using the URL http://mmm.facebook.com instead of http://localhost [4]. This URL makes web-surfing without an Internet connection more attractive.

At the beginning, the dewsite does not have the user's personal data. To let the dewsite synchronize with the website, the user needs to grant his/her http://www.facebook.com credentials to the dewsite. These credentials will be recorded by the dewsite and used in the future. The dewsite will be able to synchronize with the website http://www.facebook.com and the user's personal data and his/her friends' related data will be transferred to the dewsite database. The dewsite will always be available even when an Internet connection is not available. If the user makes changes on the dewsite when there is no Internet connection, the synchronization will not occur immediately, but it will be performed automatically when an Internet connection is available later.

If many websites adopt cloud-dew architecture and many dewsites are available for a local computer to host, the potential experience of web-surfing without an Internet connection will become a reality.

In this paper, we analyze the cloud-dew architecture from the distributed database system viewpoint, and further explore the potential of the distributed database systems.

2 Single-super-peer hybrid P2P network

Cloud-dew architecture extends client-server architecture with dew servers. Such extension gives clients the power of servers. The new architecture is very similar to peer-to-peer networks [5-10], with the cloud server (web server) as the central node. Such a new structure can be classified as a hybrid P2P network, or a super-peer system. In this system, some nodes are given special tasks to perform. Apparently, the web server node is a super-peer. In a standard hybrid P2P network, there are two or more super-peers; if there is only one super-peer in the system, this reduces to the client-server architecture [1].

In the cloud-dew architecture, a single-super-peer P2P network may not reduce to client-server architecture because each peer (dew server) is not just a client, but also a server. The database is replicated between the super-peer and the other peers. If the communication link between the super-peer and one peer fails, the peer is partitioned from the whole

system. The dew server on this peer node will provide some basic web services and database services so that the goal of web-surfing without an Internet connection can be achieved.

A peer node not only deals with the super-peer node, but also can perform other actions with other peers. The real P2P features are reflected by these actions and new applications are made possible by these features.

Single-super-peer P2P is worth exploring not only as the underlying structure of cloud-dew architecture, but also as a promising extension of client-server architecture. Single-super-peer P2P is much simpler than multiple-super-peer P2P; therefore, it is easier to implement. However, it is more complicated than client-server architecture so that it can support various distributed database applications.

3 Transparencies

Transparency refers to separation of the higher-level semantics of a system from lower-level implementation issues [1, 11]. In other words, a transparent system "hides" the implementation details from users. There are different forms of transparency. In the context of this paper, the following forms of transparency are of our concern: replication transparency and distribution transparency [1].

Replication transparency refers to whether the users should be aware of the existence of copies or whether the system should handle the management of copies and the users should act as if there is a single copy of the data.

Distribution transparency, or network transparency, refers to that there would be no difference between database applications that would run on a centralized database and those that would run on a distributed database.

Although it is desirable that replication transparency and distribution transparency be provided as a standard feature of DBMSs, this is not always the case. The essences of transparency are: (1) to hide some details; (2) to create an illusion.

Suppose we are accessing a website, say http://www.facebook.com. If an Internet connection is not available, this website will not be accessible. We may replicate the website and the database in the local node so that this website will still be available even though there is no internet connection. If replication transparency and distribution transparency are both kept, we may need to modify the behavior of the browser so that when we want to access http://www.facebook.com, the browser will first try to connect to the website server; if the website server is available, everything is normal; if the website server is not available, the browser will try to connect to the local server and access the replicated website and database.

The transparency hides all the detailed behavior of the browser, hides the communication link failure, hides the existence and the operation of a local website and related database, and creates an illusion that there are no communication link failures and the website http://www.facebook.com is still available.

However, does this transparency arrangement give the user what he/she wants? The ability to still use a website when there is no Internet connection is convenient. Nevertheless, the offline website cannot provide exactly the same services as the online website. For example, the online web application changes the online database status as the user makes changes, but the offline web application will change the online database only once the user is online again. Additionally, the offline web application can only access a portion of the online database. For these reasons, the illusion created by the transparencies is, perhaps, too lofty and is not realistic. A more practical solution is not to keep the transparencies, and to tell the user exactly what is being provided.

In the cloud-dew architecture, a local domain name system is provided. Such a local domain name system is based on the fact that transparencies are not supported. Users know the cloud and the dew. Using the example mentioned above, a user knows the difference between http://www.facebook.com and http://mmm.facebook.com,, and has different expectations for these two related websites.

To summarize the above discussions, although transparency is generally considered a great feature, a concrete application may not want to support transparency for either of the following reasons:

(1) The illusion created by the transparency is not realistic in this application.

(2) The user needs to be involved in some details that otherwise would be covered by transparency.

4 Replication update strategies

Distributed database systems can increase system availability and remove single points of failure by replicating data [1]. In the case of cloud-dew architecture, database replication is the foundation of web-surfing without an Internet connection. Although data replication has clear benefits, it poses the considerable challenge of keeping different copies synchronized. In a cloud-dew architecture application, if there is no Internet connection between the cloud and the dew and the user has changed data at the dew level, the dew changes must be synchronized with the cloud server (the central node) when it is possible. In other words, mutual consistency, which refers to the replicas converging to the same value, is necessary [3].

In terms of replication update methods, two orthogonal dimensions can be used to classify [1]. One dimension is eager update and lazy update; eager update performs all of the updates within the context of the global transaction; lazy update propagates the updates sometime after the initiating transaction is committed. The other dimension is centralized update propagation and distributed update propagation; the centralized update requires that the updates are first applied at a master copy; the distributed update applies the update on the local copy at the site where the update transaction originates. Therefore, four combinations are possible: eager centralized, eager distributed, lazy centralized, and lazy distributed.

In the application context where the cloud-dew architecture is proposed, Internet connections may get lost constantly for an extended period of time. In such a situation, eager replication update is not possible and not necessary. Thus, lazy update is the choice. Between lazy centralized and lazy distributed, the central super-peer node is often not available when the Internet connection is lost. This leaves only one applicable replication update method: lazy distributed update. In this method, the propagation to other copies is done asynchronously from the original transaction, by means of refresh transactions that are sent to the replica sites some time after the update transaction commits. Lazy distributed replication protocols are the most complex ones owing to the fact that updates can occur on any replica and that they are propagated to the other replicas lazily [1, 12-14].

Normally, the dew database is a partial replica of the central database. The central database contains data of all users, but the dew database can only contain data of the current user. Therefore, the dew database is a subset of the central database. In special cases, should the application require, it is possible for the dew database to contain data beyond the central database. There are two situations in which part of the dew database is not replicated to the central database. The first situation is that this portion of data is trivial and only related to detailed execution of the dew operations. The second situation is that this portion of data is too important and the user does not want to take any risk in sending this portion of data to the Internet. In either case, the extra dew data will make more varieties of web application possible.

5 Conclusions

From a distributed database systems viewpoint, cloud-dew architecture's ability to provide a web-surfing experience without an Internet connection is the realization of the distributed database systems' potential. The organization of cloud-dew architecture can be considered as a single-super-peer P2P network. Although multiple-super-peer P2P networks are popular, the single-super-peer P2P network may be a promising extension of the client-server architecture. Transparencies are generally desirable features in the design of distributed database systems, but they may not be always desirable. In cloud-dew architecture, non-transparent solutions are more suitable because users need to be aware of the communications link failure and to expect a realistic replacement. The local replica of a database not only is a subset of the database, but also could have extra local data. The local data is not replicated to the super-peer central database because either the data is too trivial to be replicated or is too important to be replicated. The extra local data could lead to new applications. A distributed database system is a generic, versatile structure; the proper use of its features may bring great inspiration to the computing world.

Acknowledgement

YW would like to gratefully and sincerely thank Prof. Tamer Ozsu at the University of Waterloo. He has discussed with YW about the direction of the cloud-dew architecture in the early stage. His vision and guidance played an important role in YW's research.

References

[1] Ozsu and Valduriez. "Principles of Distributed Database Systems". Spinger Science, 2011.

[2] Marius Cristian MAZILU. "Database Replication"; Database Systems Journal, Vol. I, No. 2, 33—38, 2010.

[3] Davidson, S. B., Garcia-Nilina, H., and Skeen, D. "Consistency in partitioned networks"; ACM Comput. Surv., 17(3):341-370, 1985.

[4] Yingwei Wang. "Cloud-Dew Architecture"; International Journal of Cloud Computing, OPEN ACCESS, http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=ijcc, 2014

[5] Beverly Yang, Hector Garcia-Molina. "Designing a Super-peer Network"; in Proceedings of the 19th International Conference on Data Engineering (ICDE), Bangalore, India, 2003.

[6] Beverly Yang, Hector Garcia-Molina. "Comparing Hybrid Peer-to-Peer Systems"; in Proceedings of the 27th International Conference on Very Large Databases (VLDB), Roma, Italy, 2001.

[7] Ulusoy, O. "Research Issues in peer-to-peer data management"; in Proc. 22nd Int. Symp. On Computer and Information Science, 1--8, 2007,

[8] Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I. "Data management for peer-to-peer computing: A vision"; in Proc. 5th Int. Workshop on the World Wide Web and Databases, 89--94, 2002.

[9] Daswani, N., Garcia-Molina, H., and Yang, B. “Open problems in data-sharing peer-to-peer systems”; in Proc. 9th Int. Conf. on Database Theory, 1--15, 2003.

[10] Valduriez, P. and Pacitti, E. “Data management in large-scale p2p systems”; in Proc. 6th Int. Conf. High Performance Comp. for Computational Sci., 104--118, 2004.

[11] Umar Farooq Minhas, Shriram Rajagopalan, Brendan Cully, Ashraf Aboulnaga, Kenneth Salem, Andrew Warfield. “RemusDB: Transparent High-Availability for Database Systems”; in Proc. of the VLDB Endowment, 4(11), 2011.

[12] Khuzaima Daudjee, Kenneth Salem. “Lazy Database Replication with Snapshot Isolation”; in Proc. International Conference on Very Large Data Bases (VLDB'06), 715--726, 2006.

[13] Khuzaima Daudjee, Kenneth Salem. “A Pure Lazy Technique for Scalable Transaction Processing in Replicated Databases”; in International Conference on Parallel and Distributed Systems (ICPADS'05), 802--808, 2005.

[14] Khuzaima Daudjee, Kenneth Salem. “Lazy Database Replication with Ordering Guarantees”; in Proc. International Conference on Data Engineering (ICDE'04), 424--435, 2004.

OpenCL-Based Design of an FPGA Accelerator for Phase-Based Correspondence Matching

Shunsuke Tatsumi, Masanori Hariyama, Mamoru Miura, Koichi Ito, Takafumi Aoki
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki Aza, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {s_tatsumi@, hariyama@, miura@aoki., ito@aoki., aoki@}ecei.tohoku.ac.jp

Abstract— *This paper proposes a Field Programmable Gate Array (FPGA) implementation of the stereo correspondence matching using Phase-Only Correlation (POC). The use of high-accuracy stereo correspondence matching based on POC makes it possible to measure accurate 3D shape of an object using stereo vision. The drawback of the POC-based approach is its high computational cost. To address this problem, we propose an FPGA implementation of the POC-based correspondence matching. To design the accelerator efficiently, the OpenCL-based design tool is used which allows us to reuse the existing code for Graphics Processing Units (GPUs). Although reusing the OpenCL code for GPUs, optimizing the code for FPGAs is a tough problem because the architectures of GPUs and FPGAs are completely different. The major contribution of this paper is to address the optimization technologies of an OpenCL-based FPGA accelerator. The implementation results demonstrate that the FPGA implementation has the almost same speed as well as much higher energy efficiency.*

Keywords: stereo vision, phase-only correlation, real-time 3D measurement, OpenCL, FPGA.

1. Introduction

Image correspondence is an important fundamental task in a variety of image processing [1] such applications as stereo vision, motion analysis, biometrics, etc. Especially for stereo-vision 3D measurement, high-accuracy and dense image correspondence is essential. For the purpose of accurate and dense 3D measurement, we have proposed a stereo correspondence algorithm using Phase-Only Correlation (POC) [2]. POC is an image matching technique using the phase components in Discrete Fourier Transforms (DFTs) of given images. We have also developed a passive 3D measurement system using stereo vision whose accuracy is comparable with the active 3D measurement system. However, the POC-based correspondence matching is limited due to the high computational cost, since POC is based on Fourier transform. Also, the computational cost of the POC-based correspondence matching is significantly increased when measuring the dense 3D shape of an object. This results in the large computing time on CPU implementation even though the multi-thread technique is used. Another problem of the CPU implementation is its large power consumption. To solve this problem, Graphics Processing Unit (GPU) implementation of POC has been proposed [3] where the OpenCL language is used for design. The GPU implementation is up-to 5 times faster than a CPU implementation. However, its large-power problem is still remaining.

This paper presents a Field Programmable Gate Array (FPGA) implementation of the POC-based correspondence matching, which can achieve high-speed and low-power consumption. To design the FPGA-based accelerator efficiently, the OpenCL-based design tool is used which allows us to reuse the existing code for GPUs. OpenCL is a framework supporting parallel programming in heterogeneous computational environments such as multi-core CPUs and GPUs. It provides efficient parallel computing using both task-based and data-based parallelism [4]. Recently, Altera corp. starts to provide the OpenCL design environment for FPGAs [5]. Although reusing the OpenCL code for GPUs, optimizing the code for FPGAs are a tough problem because the architectures of GPUs and FPGAs are completely different. We describe some optimization techniques such as pipelining and data reusing suitable for OpenCL-based design for FPGAs. The implementation result demonstrate that the FPGA implementation has the almost same speed as well as much higher energy efficiency. The use of FPGAs allows the image processing with high computational cost to be embedded into a small system due to its high energy efficiency.

2. Phase-Based Correspondence Matching

We briefly introduce a Phase-Only Correlation (POC) function (which is sometimes called the "phase-correlation function") [6], [7]. Let $f(n)$ and $g(n)$ be the 1D image signals, where $-M \leq n \leq M$ and the signal length is $N = 2M + 1$. Then, the normalized cross-power spectrum $R(k)$ is defined as

$$R(k) = \frac{F(k)\overline{G(k)}}{|F(k)\overline{G(k)}|} = e^{j(\theta_F(k) - \theta_G(k))}, \quad (1)$$

where $F(k)$ and $G(k)$ are the 1D DFTs of $f(n)$ and $g(n)$, $\overline{G(k)}$ denotes the complex conjugate of $G(k)$, and $-M \leq$

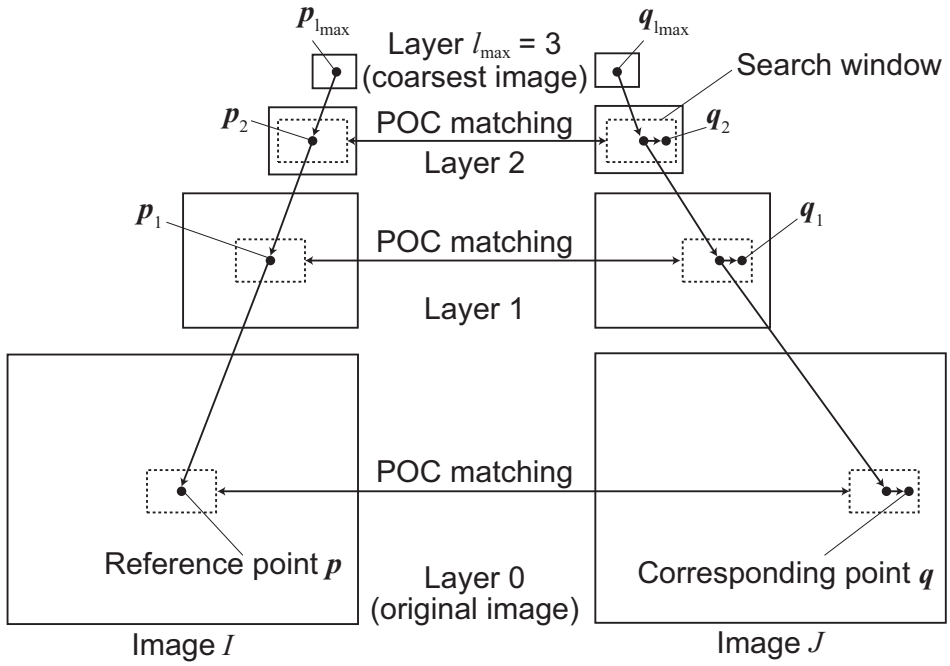


Fig. 1: Overview of the High-accuracy correspondence matching.

$k \leq M$. The 1D POC function $r(n)$ between $f(n)$ and $g(n)$ is given as the 1D Inverse DFT (1D IDFT) of $R(k)$. When two images are similar, their POC function gives a distinct sharp peak. When two images are not similar, the peak drops significantly. The height of the peak gives a good similarity measure for image matching, and the location of the peak shows the translational displacement between the images. We also employ the important techniques for improving the accuracy of 1D image matching for sub-pixel correspondence matching: (i) function fitting for high-accuracy estimation of peak position, (ii) windowing to reduce boundary effects, (iii) spectral weighting for reducing aliasing and noise effects and (iv) averaging 1D POC functions to improve peak-to-noise ratio [2].

In the case of a rectified stereo image pair, the disparity can be limited to horizontal direction [1]. The use of 1D POC makes it possible to achieve high-accuracy correspondence matching with low computational cost. In order to find the accurate correspondence from a stereo image pair, we employ the sub-pixel correspondence matching using POC. Figure 1 shows an overview of the high-accuracy correspondence matching which employs a coarse-to-fine strategy using image pyramids for robust correspondence search. Let $\boldsymbol{p}$ be a coordinate vector of a reference pixel in the reference image $I(n_1, n_2)$. The problem of sub-pixel correspondence search is to find a real-number coordinate vector $\boldsymbol{q}$ in the input image $J(n_1, n_2)$ that corresponds to the reference pixel $\boldsymbol{p}$ in $I(n_1, n_2)$. We briefly explain the procedure as follows.

Step 1: For $l = 1, 2, \cdots, l_{\max} - 1$, create the l-th layer images $I_l(n_1, n_2)$ and $J_l(n_1, n_2)$, i.e., coarser versions of $I_0(n_1, n_2)$ and $J_0(n_1, n_2)$, recursively as follows:

$$I_l(n_1, n_2) = \frac{1}{4} \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} I_{l-1}(2n_1 + i_1, 2n_2 + i_2),$$

$$J_l(n_1, n_2) = \frac{1}{4} \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} J_{l-1}(2n_1 + i_1, 2n_2 + i_2).$$

Step 2: For every layer $l = 1, 2, \cdots, l_{\max}$, calculate the coordinate $\boldsymbol{p}_l = (p_{l1}, p_{l2})$ corresponding to the original reference point $\boldsymbol{p}_0$ recursively as follows:

$$\boldsymbol{p}_l = \lfloor \tfrac{1}{2} \boldsymbol{p}_{l-1} \rfloor = (\lfloor \tfrac{1}{2} p_{l-1\ 1} \rfloor, \lfloor \tfrac{1}{2} p_{l-1\ 2} \rfloor), \quad (2)$$

where $\lfloor z \rfloor$ denotes the operation to round the element of z to the nearest integer towards minus infinity.

Step 3: We assume that $\boldsymbol{q}_{l_{\max}} = \boldsymbol{p}_{l_{\max}}$ in the coarsest layer. Let $l = l_{\max} - 1$.

Step 4: From the l-th layer images $I_l(n_1, n_2)$ and $J_l(n_1, n_2)$, extract two sub-images (or search windows) $f_l(n_1, n_2)$ and $g_l(n_1, n_2)$ with their centers on $\boldsymbol{p}_l$ and $2\boldsymbol{q}_{l+1}$, respectively. The image blocks consist of L lines of N-point 1D signal.

Step 5: Estimate the displacement between $f_l(n_1, n_2)$ and $g_l(n_1, n_2)$ with pixel accuracy using POC-based image matching. Let the estimated displacement vector be $\boldsymbol{\delta}_l$. The l-th layer correspondence $\boldsymbol{q}_l$ is determined as follows:

$$\boldsymbol{q}_l = 2\boldsymbol{q}_{l+1} + \boldsymbol{\delta}_l. \quad (3)$$

Step 6: Decrement the counter by 1 as $l = l - 1$ and repeat from Step 4 to Step 6 while $l \geq 0$.

Step 7: From the original images $I_0(n_1, n_2)$ and $J_0(n_1, n_2)$, extract two image blocks with their centers on $\boldsymbol{p}_0$ and $\boldsymbol{q}_0$, respectively. Estimate the displacement between the two search windows with sub-pixel accuracy using POC-based image matching. Let the estimated displacement vector with sub-pixel accuracy be denoted by $\boldsymbol{\delta} = (\delta_1, \delta_2)$. Update the corresponding point as follows:

$$\boldsymbol{q} \quad = \quad \boldsymbol{q}_0 + \boldsymbol{\delta}. \tag{4}$$

3. FPGA Implementation

3.1 FPGA Programming Model in OpenCL

Figure 2 shows the thread space in OpenCL that has hierarchical structure where the overall computation consists of workgroups. The each workgroup is a set of workitems; a workitem is equivalent to a thread. When designing a kernel, we implement processing for the workitem. Figure 3 shows the memory model in OpenCL. The global and constant memories can be accessed by any workitem; the difference between global and constant memories is that the global memory is a read/write memory while the constant memory is a read-only memory. The local memory belongs to the workgroup, and the private memory to the workitem.

In OpenCL for FPGAs, we can change the size of local and private memories flexibly unlike OpenCL for GPUs. Moreover, a recent high-end FPGA like Stratix V has a large local memory of 50M bits. This good nature of OpenCL for FPGAs can allow us to fully reuse data that are once retrieved from the global and constant memories. As a result, we can exploit the memory bandwidth efficiently. To fully reuse data based on this flexibility of memory structure, we use the following techniques:

- store all coefficients for filters and FFT in the constant memory.
- store the pre-calculated results for resource-consuming calculations such as division and square root.
- cache-oriented design. In OpenCL for FPGAs, a private cache is created for each read-only data array in the global memory.

Another big difference between OpenCL designs for FPGAs and GPUs is that pipelining can be efficiently exploited in FPGAs. Figure 4 shows the relation between a kernel and pipelining. Each instruction in a kernel is implemented as a pipeline stage in a pipeline as shown in Fig. 4 (a). The threads are fed into the kernel pipeline sequentially. This programming style is also effective to save memory bandwidth while keeping the performance (throughput). On the other hand, in OpenCL GPUs, the threads are processed in a data parallel manner which requires a large memory bandwidth. In order to fully exploit the advantages of pipeline design, OpenCL for FPGAs supports "Channel" which can connect the different kernels using FIFO buffers. Figure 5 shows the overall structure of the POC-based correspondence matching using channels. The functions of the kernels are as follows:

make high layer: generating the coarse images
clip image: clipping the search windows from images
fft1d: Fourier transformation for 1-D data
eval cps: computing cross-power spectrum
reorder: reorder data for the following ifft1d
ifft1d: inverse Fourier transformation for 1-D data
find peak: find correspondence by searching a peak

Channels can be used for passing data to kernels and synchronizing kernels. The intermediate data are stored and fed to the next kernel through channels without the global memory. Therefore, the efficient data reuse can be achieved easily. In fact, 10 000 correspondence results are generated in the find peak kernel, and they are fed to the clip image kernel through the channel without other memories in the POC matching shown in Fig. 5 (b).

3.2 Evaluation

For evaluation, we implement the POC-based stereo correspondence matching on a CPU, GPUs, and an FPGA as shown in Table 1. The parameters for the POC-based stereo matching are as follows: The size of the search window is 32 pixels $\times$ 15 lines, the number of layers is 4 and the number of reference points is 10 000. As a CPU, we use Core i7-3960X that has 6 cores (12 threads) running at a clock frequency of 3.3GHz$-$3.9GHz. As GPUs, we use Geforce GTX 580 and Geforce GTX 680 from NVIDIA corp. As an FPGA board, we use PCIe-395 D8 from Nallatech corp. that has a Stratix V FPGA from Altera corp. and 4 DDR3 memories. The maximum frequency of the FPGA design is 167MHz. Table 2 summarizes the resource utilization of the FPGA design, where the upper and lower rows are the specifications of the FPGA (Stratix V D8) and the used resources, respectively.

For evaluation metrics, we use the processing time, power consumption, and power-delay product of each implementation. We measure the power consumption of a whole computer during execution with a power meter (HIOKI AC/DC POWER HiTESTER 3334). Note that, in Table 1, the power consumption is the difference between those of idle and operation state. A power-delay product is defined as the product of the processing time and the power consumption and represents the energy for computation, that is, efficiency. The GPU implementations are 25-27 times faster than the CPU implementation with a single thread and also 4.0-4.3 times faster than the CPU implementation with 12 threads (6 cores). The FPGA implementation is 17 times faster than the single-thread CPU implementation, and 2.7 times faster than the 12-thread CPU implementation. In terms of the processing time, the FPGA implementation has almost same performance as the GPU implementations although the FPGA's

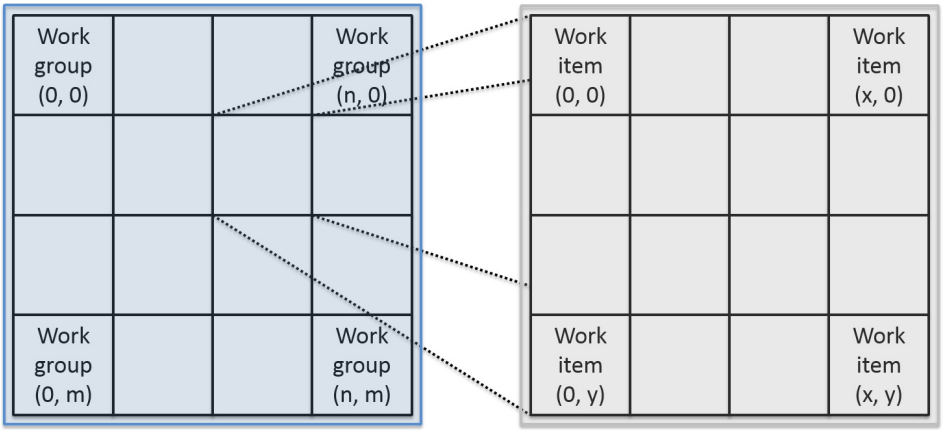


Fig. 2: Thread space in OpenCL.

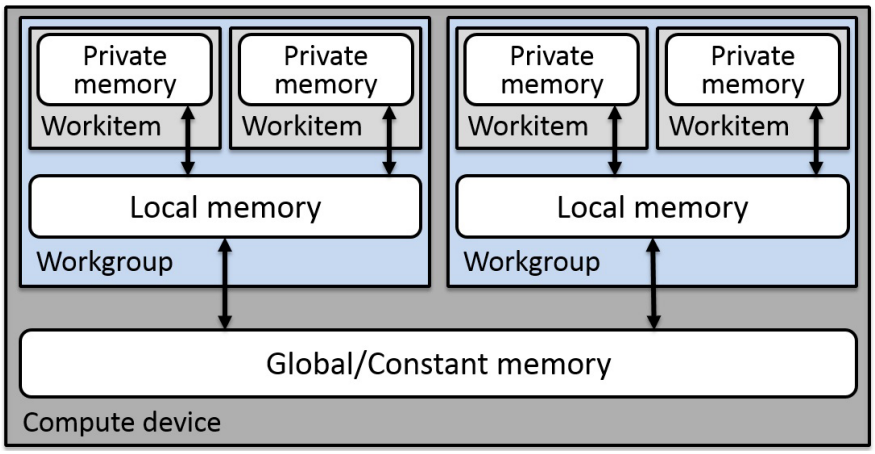


Fig. 3: Memory Model in OpenCL.

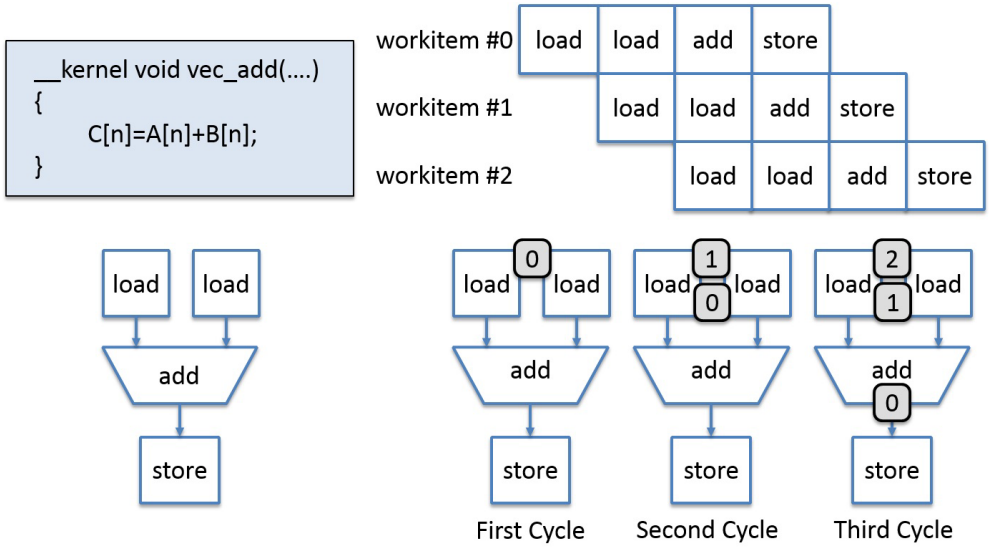


Fig. 4: Pipelining in OpneCL for FPGAs.

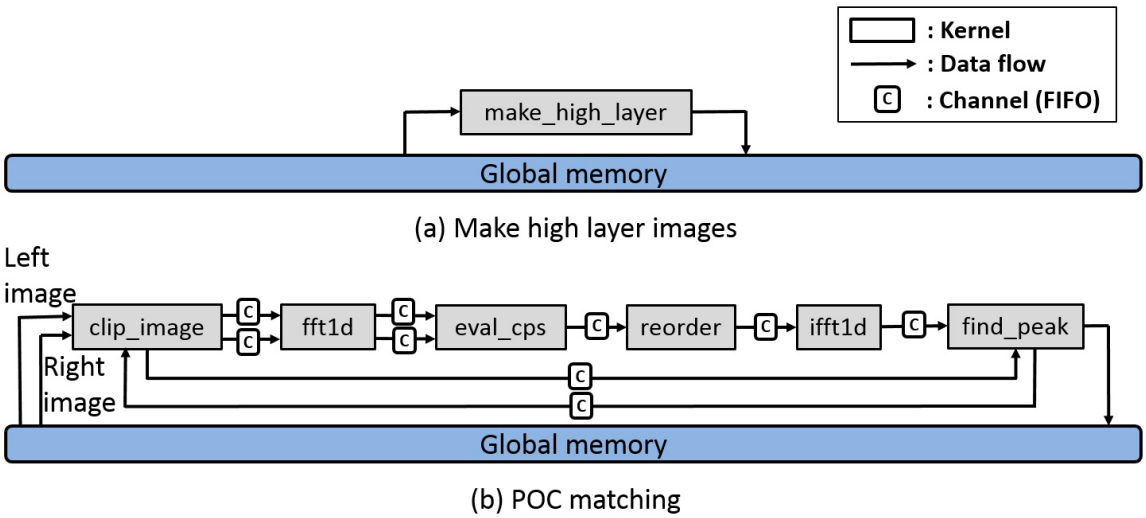


Fig. 5: Overall structure using channels.

Table 1: Performance and power comparisons among CPU-, GPU-, and FPGA-implementations.

	Corei7-3960X (1 thread)	Corei7-3960X (12 threads)	Geforce GTX 580	Geforce GTX 680	FPGA board (Nallatech PCIe-395 D8) *2
Processing time[ms]	394.42	62.92	15.63	14.43	23.11
Power consumption*1 [W]	61.40	162.50	123.90	141.20	16.60
Power-delay product[W×s]	24.22	10.22	1.94	2.04	0.38

*1 Power consumption is the difference between those of the idle time and operating state.
*2 Nallatech PCIe-395 D8 have a Stratix V FPGA (Altera Corp.).

Table 2: Resource utilization of the FPGA design.

	Logic	Registers	RAM blocks	DSP blocks
Stratix V D8	262,400	1,049,600	2,567	1,963
Implementation	189,756(72%)	394,022(36%)	1,375(54%)	346(18%)

memory bandwidth is much smaller than the GPU's one. The power-delay products of the GPU implementations are about 8.0-8.4% of the single-thread CPU implementation, and 19-20% of the 12-thread CPU implementation. The power-delay product of the FPGA implementation is 1.5% of the single-thread CPU implementation, 3.7% of the 12-thread CPU implementation, 20% of the GPU implementation (GTX 580), and 19% of the GPU implementation (GTX 680). The above results demonstrate that the FPGA implementation is much faster and much energy-efficient than the CPU implementations, and that the FPGA implementation has the almost same speed as the GPU implementations and is much energy-efficient.

4. Application example: Real-time 3-D measurement system

Figure 6 shows that we develop a real-time and accurate 3D measurement system using a moving consumer digital camera. In this system, a set of images taken different viewpoints are used for the 3D measurement system. One of the well-known 3D measurement methods is Structure from Motion (SfM) using feature-based correspondence matching [1]. However, only a limited number of 3D points are measured by this method and are not sufficient to measure the fine 3D shape of the object. Addressing this problem, our system employs the algorithm combining SfM using feature-based matching to estimate camera parameters and the POC-based correspondence matching to obtain dense correspondence. As shown in Fig. 7, the procedure of the proposed system consists of 3 steps: (i) correspondence matching based on

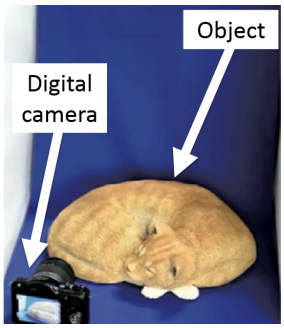


Fig. 6: Experimental 3D measurement system using consumer digital camera.

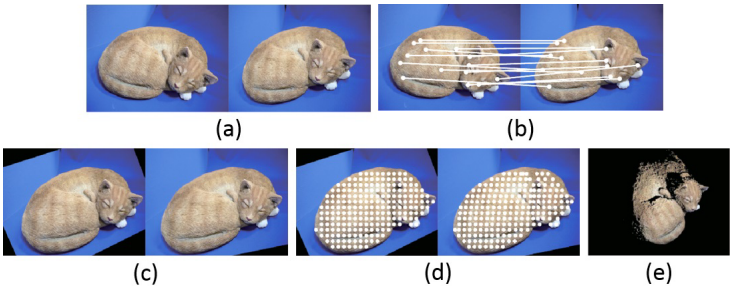


Fig. 7: Processing flow of the proposed system: (a) input stereo image pair, (b) result of SIFT-based correspondence matching, (c) rectified stereo image pair, (d) result of POC-based correspondence matching and (e) 3D measurement result.

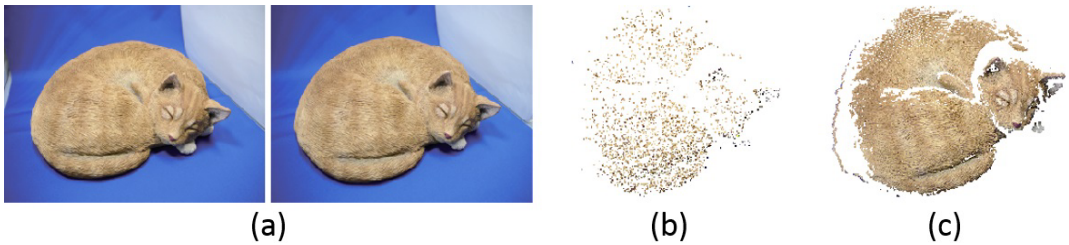


Fig. 8: Results of 3D measurement: (a) input stereo image pair, (b) 3D points measured by SIFT-based SfM (2790 points) and (c) 3D points measured by the proposed procedure (25 243 points).

Scale-Invariant Feature Transform (SIFT) [8], (ii) camera parameter estimation [1] and (iii) 3D shape measurement [9]. In the step (iii), we employ the POC-based stereo correspondence matching implemented on the FPGA board. Figures 8 (b) and (c) show 3D measurement result of the stereo image pair using SIFT-based SfM and the proposed procedure, respectively. The size of images is 1280×960 pixels. The measurement result using the proposed procedure has 25 243 points from two snapshots, while the result using SIFT-based SfM has only 2790 points.

The FPGA implementation of the POC-based stereo correspondence matching makes it possible to obtain dense 3D points of an object in a few seconds. Moreover, its high-energy efficiency would allow the total computing system to be embedded into the camera system.

5. Conclusion

This paper has proposed the FPGA implementation of the POC-based stereo correspondence matching using OpenCL for high-speed and low-power consumption. The key to success is to exploit the pipelining and to fully reuse data based on the flexibility of memory design. As future work, we are planning to generalize the design methodology for this dedicated processor aiming at automatic generation of FPGA-oriented OpenCL codes from other codes such as GPU-oriented OpenCL codes and C/C++ codes.

References

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*], London; New York, Springer-Verlag, 2011.
[2] T. Shibahara, T. Aoki, H. Nakajima, and K. Kobayashi, "A sub-pixel stereo correspondence technique based on 1D phase-only correlation," in *Proc. ICIP 2007*, pp. V–221–V–224, 2007.
[3] M. Miura, K. Fudano, K. Ito, T. Aoki, H. Takizawa, and H. Kobayashi, "Performance evaluation of phase-based correspondence matching on GPUs," in *Proc. SPIE 8856*, Applications of Digital Image Processing XXXVI, pp. 1–9, 2013.
[4] Khronos Group. (2010) The OpenCL Specification. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf
[5] Altera SDK for OpenCL. [Online]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html
[6] C. D. Kuglin, and D. C. Hines, "The phase correlation image alignment method," in *Proc. Int'l Conf. Cybernetics and Society*, pp. 163–165, 1975.
[7] K. Takita, T. Aoki, Y. Sasaki, T. Higuchi, and K. Kobayashi, "High-accuracy subpixel image registration based on phase-only correlation," *IEICE Trans. Fundamentals*, Vol. E86-A, No. 8, pp. 1925–1934, Aug. 2003.
[8] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int'l J. of Computer Vision*, Vol. 60-2, pp. 91–110, Nov. 2004.
[9] M. Miura, S. Sakai, J. Ishii, K. Ito, and T. Aoki, "An easy-to-use and accurate 3D shape measurement system using two snapshots," in *Proc. IWAIT 2013*, pp. 1103–1106, 2013.

SyncSmartv: A Framework for Synchronizing Smart TV Applications with TV Programs

Cédrick Bamba N. and Cesar A.C. Teixeira
Department of Computer Science, Federal University of São Carlos, São Carlos, São Paulo, Brazil

Abstract - *Smart TV applications are typically disconnected from the content of the tuned programming on the TV set. Some TV broadcasters offer specific applications for programs, however these applications are generally just loosely coupled or synchronized to the program. Furthermore, applications are fully under domain of TV broadcasters that have the information about the transmitted content. Based on the fact that synchronized information may arise from third-parties obtained by local content processing or offered directly by the viewers, new opportunities will be opened up for developing a bunch of new interesting applications aiming to promote the user interaction level in TV. Moreover, it will have new user interaction possibilities while using mobile devices and computers, furthermore a new business model will be emerged. In this paper, the SyncSmartv framework is presented and evaluated. The aforementioned framework offers several APIs that facilitate the development of Smart TV applications synchronized with TV program contents. The aim is to provide some facilities for developers to implement applications in this area in a clear approach without being concerned about low-level implementation details.*

Keywords: Connected television, interactivity, smart TV, application integration, television programs, TV channels monitoring.

1 Introduction

The emerge of Digital TV and smart TV has been innovating the way people watch television and changing the passive paradigm where the user could only consume contents. In the Interactive digital TV environment, the user can interact with the TV contents because the TV broadcasters send common contents (i.e: novel, TV show, etcetera) along with their interactive application that can be executed on the TV processor.

However unlike the interactive scenario in iDTV, most of the time the audiovisual content of programs in the Connected TV does not communicate with the native TV applications and other TV platform features. The only basic interaction is the possibility to switch between the things are displaying on the TV screen either program or TV applications. This approach of connecting TV to the internet, where the audiovisual content is offered (without pre-defined mechanisms of communicating with additional TV-platform features) characterizes the difference between Smart TV and iDTV.

A study on the Smart TV markets, conducted by *Strategy Analytics* and *BI Intelligence* [3], assessed the sale of Smart TVs related to the other types of TVs / TV devices (Google's ChromeCast, Apple TV, etc.) predicted sale rate of the next three years. According to the survey whose results are shown in Fig. 1, the sale of Smart TVs is growing to dominate the overall TV market with 33% of global sales of TVs in 2013 and finished in 2014 with the rate of 44%. According to the survey, in 2017, Smart TVs will represent 73% of total sales of TVs throughout the world, 54% in 2015 and 63% 2016.

Accordingly, it would suggest that the paradigm of Smart TVs should be popularized. Namely, however some studies have been done to explain why the Smart TV concept is not yet as popular as the Smartphone's concept. In the following paragraphs we will discuss the result of some of these studies.

Bachelet [1] conducted a research in five European and North American countries in May 2013 and found out that only less than the half of 6115 Smart TV owners who were interviewed, connected their TVs to the Internet.

According to Bachelet [1], two elements are the reasons that why the Smart TV has not yet become popular and been widely accepted by consumers as much as Smartphones:

- The lack of content and interesting applications: although the majority of Smart TVs offer a wide range of content and applications, most of them are irrelevant and are not interesting to users.
- The Poor User Interface: a lack of rich user interface that can integrate TV applications with its audiovisual contents.

Schofield [7] mentioned that "if TVs are going to be truly smart they must do more than offer a wide variety of online video services. Instead they must add advanced functionality including voice control, motion control, advanced advertising, attractive user interfaces and two-way communications with other smart devices – so-called 'second screens'– allowing these devices both to send video to the TV and know what is being watched. Manufacturers should focus less on adding more content and more on improving how users can interact with that content".

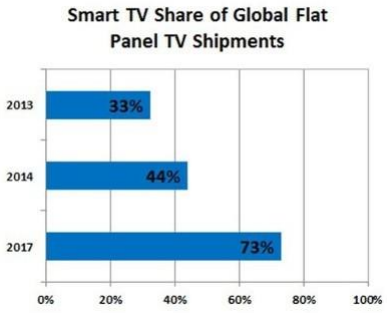


Fig. 1. TVs sales rates [Source: Strategy, 2014].

This suggests that the potential for interactivity of Connected TVs has not yet been properly utilized. Hence new synchronization mechanisms and interaction with TV environment can help to improve the user experience.

Based on the finding that says innovative interactivity has not been promoted and used adequately in Connected TV platforms is a major reason for its limited use. Another reason for low adherence to the interactive resources of Smart TV is because of the lack of facilities that allow the integration of Smart TV applications with their own audiovisual content. The observation of these aspects motivated the current proposed paper.

2 Related Work

Several authors reported some results of building frameworks designed to support TV applications development. However just few of them focus on reuse in Smart TV applications, specifically those ones synchronized with television programming.

Group Share-TV [5] proposes a framework called share-tv used for the development of converged applications centered on TV for GoogleTV and Ginga-J platforms. The share-tv allows the development of TV applications that include a generic mobile application. Ever since this generic application get downloaded from a given available TV IP, installed and run on mobile device, the communication with the TV convergent application will be initiated automatically in order to register and receive shared objects. While the objects are being received, the device show them on the screen allowing interaction on them. Compared to the work reported in this paper, share-tv also provides communication services and reuse, however, this framework is limited to applications based on GoogleTV and Ginga-J platforms. The main difference is that share-tv is used to develop TV convergent applications while the work reported in this paper focuses on building Smart TV applications synchronized with the content of TV programs.

Samsung Smart TV [6] presents a framework called AppsFramework. This framework encapsulates reusable modules for scene management, video playback / music, and so on. This makes it easier for the developer of Smart TV applications to avoid performing complicated sequences of calls to the operating system in order to manage scenes (focusing, showing and hiding events) of an application, for instance. Some of these modules reused in the framework proposed in this paper.

Freitas and Teixeira [4] proposed an architecture for supporting the development of ubiquitous applications in home networks focusing on Digital TV. The proposed architecture consists of communication interface with home devices, a protocol layer for automatic service discovery, and so on. Although the architecture is designed to be implemented in iDTV middleware, some of its reusable artifacts such as the aforementioned ones were used and implemented in the framework proposed in this paper.

The framework for building synchronized smart tv applications with tv programs presented in this paper is different with all the aforementioned works from this scene that it is a reference framework that allows more efficiency and less cost in building multi-platform applications in this field.

3 Synchronization and Notification

To achieve the goal of providing integrated applications to television programming, it is essential to know what is being presented to the audience every moment. Then this information can be published to stakeholders and used to promote synchronization between program and application.

The following subsections are devoted to two issues, synchronization and notification. We discuss some aspects of synchronization that smart TV applications should be concerned to promote integration with television programming and how the demands of notification services are generated.

3.1 Synchronization Aspects

Teixeira et al. [8] studied the synchronization in the context of multimedia applications and considered that synchronization is a mechanism to guarantee that actions can happen according to the defined time reference set by a clock or established by the occurrence of events. It considers the tolerances that vary according to the type of application. In case of television programming the characteristic of event is to be considered as a reference that is one of the important aspects of synchronization in the context of this work.

Three other relevant aspects for the integration of applications with television programming are: Source - refers to anyone who provides information about the occurrence of the event; Coupling - is related to the tolerance allowed by the application in deviations from the reference time; Exposition - categorizes two possible situations: explicit and implicit synchronization information.

3.2 Notification

To promote the used synchronization in the context of this work, we need to establish a communication protocol between the part that needs to receive synchronization signals (TV application) and the one who provides these signals (synchronization services). To do so it is necessary to create a

notification API that is an interface that allows applications to access the various offered notification features for registering such notifications in synchronization service and the last one in turn, will notify the applications about events (related to those notifications that previously were registered) occur in TV programming.

Notifications addressed in this work have various types such as start and end of trading blocs, start, pause and end of the TV programs, sex scenes, violence, crime and some notifications generated by the events triggered by the user. For example, a user can send notification related to the information about his current tuned channel to another user in TV social system.

4 SyncSmartv

SyncSmartv is a reference framework designed to facilitate the development of integrated Smart TV applications. The goal is to create mechanisms that allow developers to build applications in this field in a transparent manner, in a clear approach without being concerned about low-level implementation details. Additionally the framework aims to allow developers to think more about the business logic of their applications; hence, they can build integrated applications with lower cost and effort. In this work, when we talk about integrated applications, we mean those ones synchronized with the television programming.

4.1 Design Considerations

Bosch et al. [2] report that framework development is different from a common application development. This is because of that framework's design needs to cover all relevant features of a particular domain and not just those ones of specific application. This is why it is important to consider the following when developing framework for Smart TV integrated applications. Generally, there are six issues to consider. First, How to synchronize the TV content with Smart TV applications, Second, how mobile devices (smartphones, tablets, and others), which are in the same space with TV can interact with it. Third, how connected mobile devices in the TV environment can detect the presence of available TV service for use. Forth, how Smart TV applications can act over the TV controls such as changing channels, controlling volume and so on. Fifth, how ticker applications must share the remote control with the TV. Finally how a Smart TV application can identify the channel, which is being watched by the user.

In order to build the reference framework to support the issues that was discussed above we considered the set of artifacts in Fig. 2 for extracting the common and reusable modules existing between them. The extracted modules are a part of the SyncSmartv components.

In the right side there are some built applications (shown in gray rectangles) that have some typical characteristics of one or more applications or components located in the left (represented in the form of ellipse). These applications are simple that are not directly interesting for end users. These applications built in order to explore features of Smart TV. An application of the right side presents similar characteristics to a component of application on the left hand while there is a line that joins each rectangles in right hand to the corresponding application in the right and passes over the ellipses.

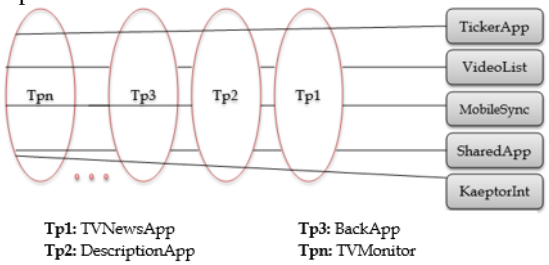


Fig. 2. Artifacts considered supporting the development of SyncSmartv.

The set of artifacts considered in Fig. 2 is composed of nine main applications: the *TickerApp* used to manage the use of remote control between applications and TV; the *VideoList* is in charge of receiving synchronized notifications generated by the viewer and then starts to playback the pre-defined video list; the *MobileSync* uses HTTP protocol to allow two-ways communication and sharing media content among the TV and all the existing devices in the same environment; the *SharedApp* allows the viewer to use the Smart TV as second screen in a synchronized way with the TV programming; the *KaeptorInt* implements some protocols of the third-services providers aiming to facilitate TV applications to receive synchronized notifications; the *BackApp* is responsible for switching channels automatically after that a synchronization event (starting scene crime, for example) is triggered; the *DescriptionApp* allows user to access more details of the product which is in focus in the current scene; the *TVNewsApp* allows user (on a given tuned channel) to be provided with informations related to the status of the broadcasting TV news and the *TVMonitor*, which contains most of the functionalities of the aforementioned applications that is one of the completed and integrated applications developed by the author of this paper as a proof of concept of the framework.

4.2 Development

Below, we illustrate only the domain analysis of the first problem (how to synchronize the TV content with Smart TV applications), that is, how some appropriate artifacts were extracted and used to resolve this problem.

First, the common features are identified while the applications of Fig. 2 were comparing and after a simplified design (can be a textual description) of artifacts needed to implement each feature, was made. Then a feasibility evaluation was done on the artifacts found by analyzing, for example the estimated effort in construction and reuse level, that means how many applications would take advantage of this artifact. Finally, we decided whether it is beneficial to implement them or not.

After identifying the first functional requirement (1. Means to receive synchronization information) for the first problem that already mentioned, two artifacts were drafted: 1. a login

service and connection establishment with the TV channel monitoring service 2. A registration and communication service with the registered channel. In Fig. 3, the use cases in yellow are functionalities of the artifact 1 (login service and connection establishment with the TV channel monitoring service) and in blue are part of the artifact 2 (registration service and communication with the registered channel).

4.2.1 SyncSmartv Architecture

The following figure depicts the architecture of the SyncSmartv. In a quick view this architecture is based on the AppFramework architecture [6] where the Framework layer in gray and Adaption Layer in blue were added. The SyncSmartv integrates a set of tools and components that enable a software engineer to quickly design, develop and deploy new Smart TV integrated application. The main SyncSmartv functionality is offered by using its API. Using the API specification a developer may build various applications such as monitoring applications that detect crime or sex scene and start a playback of some music from the viewer mobile device.

The figure consists of four layers, the first called *Application Manager* is where applications are managed and built using languages such as JavaScript, HTML, CSS and so on. Usually once the application is built, it is executed on the *AppEngine*

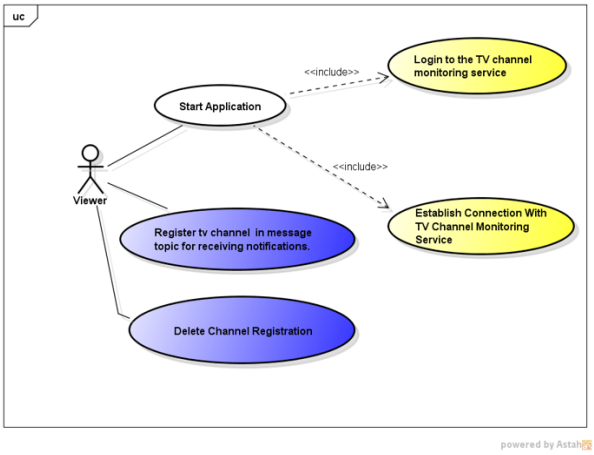
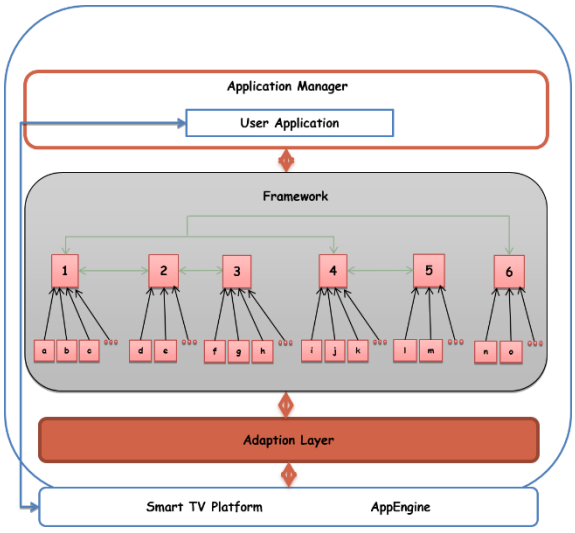


Fig. 3. Part of the Use Case Diagram of TVMonitor.

Smart TV Platform layer. If the developer wants to build an integrated application, it can take advantage of the available features in the *framework* layer. Then this application uses the *Adaption* Layer to implement generic artifacts specified in the framework according to the need of each Smart TV platform used for application execution. Finally, the application is run in the *Smart TV Platform* layer. The next paragrahs provide an overview of SyncSmartv modules that meet the functional requirements listed in Subsection 4.1.

4.2.2 Modules Overview

The SyncSmartv set of modules we present in this section is a set of JavaScript classes and interfaces, bind with synchronization services, which aim to add a level of integration between the Smart TV, the application and for enhancing TV social experience. The *Sync Event Manager* module provides an interface for Smart TV applications for receiving the notifications of the occurring events in television programming. The *Communication Manager* provides the features needed to establish a two-way connection between Smart TV applications and mobile devices (smartphones, tablets, etc.) found in the same environment. The *User Device Discovery* module provides necessary mechanisms for the connected devices such as smartphones, tablets, printers, etc can discover the existence of some available TV services for use. The *TV Control Manager* module provides functionalities for Smart TV applications that act on TV controls. The *Input Manager* provides mechanisms to manage the use of remote control buttons between applications and the TV. The *TV Channel ID* provides mechanisms to detect the channel being watched by the user, through a clear approach.



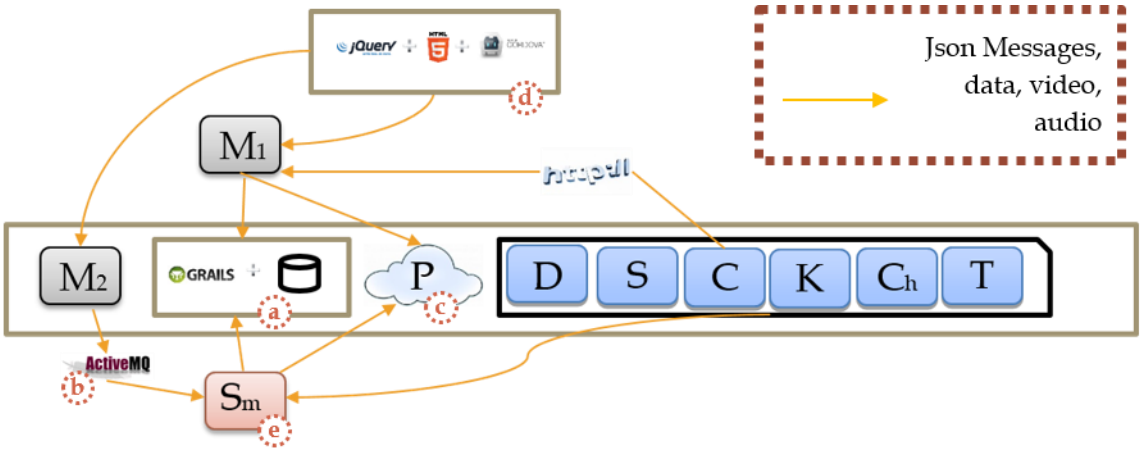
1: Sync Event Manager
a: MPEG-TS b: Audio Matching
c: Third Party Services
2: Communication Manager
d: Bluetooth e: Wifi
3: User Device Discovery
f: UPnP g: Url h: Zeroconf
4: TV Control Manager
i: TV Channel Change j: Mute k: Volume Control
5: Input Manager
l: Register Keys m: Unregister Keys
6: TV Channel ID
n: Image Processing o: Audio Processing

Fig. 4. SyncSmartv Development Framework.

4.2.3 Framework Instantiation

As a proof of concept of the framework, we developed a set of tools to implement test with developers. This set of tools were mainly developed using JavaScript. The Fig. 5 depicts an overview of the set.

The set is composed of two main parts: the *Back-end* used to store the information of users and devices and to allow communication and sharing of media content among different components of the set; and the *Front-end*, which contains applications executed on mobile devices and Smart TV platforms.



M1: First Mobile Application (MobileSync)
M2: Second Mobile Application (AdminSync)
Sm: Smart TV Application (TVMonitor)
P: PHP Server
D: Discovery
S: Synchronization
C: Communication
K: Keymanagement
Ch: Channel identification
T: TV

Fig. 5. Implementation Overview.

Back-end

The *Back-end* of this work provides web services for Smart TV discovery services and for managing the users of social TV systems. The aforementioned services were developed using the Grails framework (Fig. 5(a)). Moreover, the *Back-end* allows communication and sharing the media content among the different applications respectively using Apache ActiveMQ message broker (Fig. 5(b)) and PHP server (Fig. 5(c)).

Front-end

The *Front-end* is based on client-side (PC, Smart TV, Tablet, Smart Phone, etc.) that was developed using Apache Cordova + HTML5 + JQuery (Fig. 5(d)) and JavaScript-based SyncSmartv API's.

5 Framework Validation

5.1 Developing Experimentation

To validate the framework proposed here we used the experimental method proposed by Wohlin et al. [9].
The experiment consisted of a comparative study of the development processes of two versions of *TVMonitor* application, one built with the reuse approach using the proposed framework and other built without this approach.

The experimental phases performed in the study will be expatiated in the following subsections.

5.1.1 Definition of the Experiment

The objective of this study was:

- **To analyse** the use of the proposed framework in the construction of Smart TV applications synchronized with television programing;
- **With the purpose** of evaluation
- **Regarding** the efficiency in terms of time spent and productivity;
- **From a point of view** of software developers;
- **In the context** of undergraduate and graduated in computer science and computer engineering. It is important to point out that in this experiment twelve (12) developers and one object were considered (*TVMonitor*).

5.1.2 Design of the Experiment

Efficiency was selected as dependent variable for this experiment. Independent variables were the development method, the developed application, the development environment and development technologies.

Among the four independent variables, three were kept constant during the study:

- ✓ Application = TVMonitor ;
- ✓ Development environment = Eclipse IDE ;
- ✓ Development Technologies = HTML, CSS, JavaScript and JQuery.

Formulation of Hypotheses

For the formulation of the three hypotheses, the following metrics were considered:

τ - Total time spent by the team for developing Smart TV application synchronized with the TV program;

Þ - Team Productivity in terms of produced lines of code (LOC) per unit time (Þ = LOC / τ);

μ_τ - Average of the spent time by the teams for developing Smart TV application synchronized with the TV program;

$\mu_Þ$ - Average productiveness of the teams in the development of Smart TV application synchronized with the TV program.

We have a null hypothesis and its two corresponding alternatives:

- ***Null Hypothesis (H_0):*** There is no difference between teams who used the proposed framework and teams that did not use while developing *TVMonitor* application regarding the efficiency (ε) of the team.

  H_0: $\varepsilon_{framework} = \varepsilon_{withoutframework} \Rightarrow \mu_{\tau framework} = \mu_{\tau withoutframework}$ *e* $\mu_{Þframework} = \mu_{Þwithoutframework}$
- ***Alternative Hypothesis (H_1):*** Teams who use the proposed framework for building *TVMonitor* application are generally more efficient than those ones who developed without the use of framework.

  H_1: $\varepsilon_{framework} > \varepsilon_{withoutframework} \Rightarrow \mu_{\tau framework} < \mu_{\tau withoutframework}$ *e* $\mu_{Þframework} > \mu_{Þwithoutframework}$
- ***Alternative Hypothesis (H_2):*** Teams using the approach "*without framework*" for building **TVMonitor** application are generally more efficient than those developing with the use of framework.

  H_2: $\varepsilon_{framework} < \varepsilon_{withoutframework} \Rightarrow \mu_{\tau framework} > \mu_{\tau withoutframework}$ *e* $\mu_{Þframework} < \mu_{Þwithoutframework}$.

5.1.3 Implementing the Experiment

In this step we prepared effectively the material needed to support the experiment, that means, the set of objects manipulated during the experimentation and which are defined in 4.2.3. In addition, were prepared documents that allowed the experimenter to exchange information with the participants.

The organization of the data in Fig. 6 is done according to the two development approaches used in this experiment: development with and without the use of the framework reported in this paper.

5.1.4 Analysis and Interpretation of Results

An initial analysis was done on the data collected in Fig. 6. It is important to note that the distribution efforts of the groups in the design and test phases of *TVMonitor* development was constant. However, there is a great discrepancy of development efforts for the groups that used the proposed framework during the implementation of *TVMonitor*. While groups which have not used the proposed framework, in average they spent 4 hours and 59 minutes but those who used the framework spent 2 hours and 08 minutes (a decrease of 57, 2%).

	Group	StartTPro	FinTPro	StartTImp	FinTImp	StartTTest	FinTTest	LCG Auto	LCG Man	TLOC	TT(τ)	TPrd(p)
With Framework	G1	11:20	11:46	12:40	15:21	15:27	16:18	3270	52	2532	3:58	838
	G2	11:56	12:13	12:47	15:19	15:25	16:04	2826	61	2541	3:28	833
	G3	14:30	14:47	15:00	16:10	16:20	16:30	2480	56	2536	1:37	1569
	Average	00:20		2:08		00:33					3:01	1080
Without Framework	G4	14:20	14:51	15:23	18:05	18:10	18:25	1882	361	2243	4:08	543
	G5	8:30	9:10	9:20	15:17	15:25	16:12	1882	427	2309	7:24	312
	G6	8:30	9:02	9:10	14:48	15:00	15:33	1882	392	2274	6:43	339
	Average	00:34		4:59		00:32					6:05	398

StartTPro: The Start Time of the Project;
FinTPro: The Finish Time of the Project;
StartTImp: The Start Time of the Implementation;
FinTImp: The Finish Time of the Implementation;
StartTTest: The Start Time of the Test;
FinTTest: The Finish Time of the Test;

LCGAuto: Lines of Code Generated Automatically;
LCGMan: Lines of Code Generate Manually;
TLOC: Total Lines Of Code;
TT: Total Time;
TPrd: Total Productivity.

Fig. 6: Data Collected.

Testing Hypotheses

The purpose of the t-test (Montgomery, 2000) is to verify that a variable differs between two independent samples, based on the arithmetic average and considering variability of its data items. Then with some degree of significance (α), reject the null hypothesis (H_0) and choose one of the alternative hypothesis (H_1 or H_2). The t-test formula is given by equation (1), where Sx^2 and Sy^2 are

the variances of each sample; *Sp* is the dispersion; and *n* and *m* are the numbers of data items that each sample contains. In equation (2), n+m-2, typically noted by gl is called the degree of test's freedom.

$$t_0 = \frac{x-y}{Sp\sqrt{\frac{1}{n}+\frac{1}{m}}} \quad (1)$$

a)

$$Sp = \sqrt{\frac{(n-1)Sx^2+(m-1)Sy^2}{n+m-2}} \quad (2)$$

Once you have calculated t_o, α and gl, you can check the value of the standard t in t-test distribution to see if t_0 is so significant.

If $|t_0|$ > standard t = $t_{\alpha/2\ gl}$ → **REJECT H_0**,
Otherwise, → H_0 **NOT REJECTED** and no conclusion is drawn from the experiment.

As the dependent variable of the experiment (efficiency of teams) has two treatments (total time (τ) and productivity (Þ)), the application of t-test was performed in two steps too. During this process, we calculated the variance using the following equation:

$$Sx^2 = \frac{\sum_{i=1}^{n}(x-\bar{x})^2}{n-1} \quad (3), \text{ with } n = 3$$

Step 1: t-test (Total Time)

After calculating the variance of each group, we have:

Sx^2 ($_{withoutframework}$)= 2,97723 Sx^2 ($_{withframework}$)= 1,5325
α= 0,2 Sp = 1,501620791012165
$t_{\alpha/2,\ gl} = t_{0,1000,\ 4}$ = 2,1318 t_0 = -2,498498775014366

Then we have $|t_0| > t_{0,1000,\ 4}$ **REJECT the null hypothesis H_0 with 20% of significance.**

Step 2: t-test (Total Productivity)

After calculating the variance of each group, we have:

Sx^2 ($_{withoutframework}$)= 15951 Sx^2 ($_{withframework}$)= 179347
α= 0,02 Sp = 130,1691207621838
$t_{\alpha/2,\ gl} = t_{0,01,\ 4}$ = 4,6041 t_0 = 5,475964737075994

Then we have $|t_0| > t_{0,01,\ 4}$ **REJECT the null hypothesis H_0 with 2% of significance.**

6 Final Remarks

SynscSmartv can be offered to developers in form of a semi-complete source code skeleton that integrates synchronization, notification and TV controls functions. A set of tools that were developed in 4.2.3 with the purpose of a proof of concept of the framework can provide to a developer more facilities in the process of building Smart TV integrated applications. In addition, all functional requirements that were established while planning the development of this framework were implemented and used during the instantiation of the SyncSmartv. In addition, the experiment conducted in this paper could prove statistically that the SyncSmartv can be considered as an important tool to support the developers of applications in this field.

Regarding future work, we plan to: (a) add mechanisms of synchronization through local audio/video processing of TV content to framework; (b) explore and add adjustment mechanisms of synchronization with the purpose of minimizing the delay difference that exists among various forms of television content transmission (radio broadcasting, cable, satellite, etc); and (c) offer more notification API's for supporting the development of social TV systems with the purpose of improving the user experience quality in Smart TV environment.

7 References

[1] Bachelet, C. Most smart-TV owners do not connect their TVs to the Internet: manufacturers must respond. Analysys Mason 2013. Disponível em: <http://www.analysysmason.com/About-Us/News/Insight/smart-TV-May2013/>. Acessado em: mar. 2014.

[2] Bosch, J; Molin, P; Mattsson, M; Bengtsson, P; Fayad, M. Framework Problems and Experiences. In: Fayad, M.; Johnson, R.; Schmidt D. Building Application Frameworks: Object-Oriented Foundations of FrameworkDesign.Nova Iorque : John Willey and Sons, p. 55-82, 1999.

[3] David, W. 2013 Smart TV Shipments Grew 55 Percent. *Strategy Analytics* 2013. Disponível em: <https://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5472>. Acessado em: mar. 2014.

[4] Freitas, G; Teixeira, C. Uma arquitetura de serviços para aplicações ubíquas em redes domésticas centrada em TV digital. In: *XVI Simpósio Brasileiro de Sistemas Multimídia e Web (Webmedia 2010)*, 2010, Belo Horizonte - MG. Anais do XVI Simpósio Brasileiro de Sistemas Multimídia e Web (Webmedia 2010). Porto Alegre: SBC, 2010.

[5] Group Share-Tv. Share-TV: Um framework para desenvolvimento de aplicativos convergentes centrados na TV para as plataformas GoogleTV e Ginga-J. Webmedia '12. São Paulo, 2012.

[6] Samsung Smart Tv. AppsFramework. Disponível em: <http://www.samsungdforum.com/Guide/art00017/index.html>. Acessado em: mai. 2014.

[7] Schofield, J. Smart TVs may be taking off, but they're still not smart enough. ZDNet, 2012. Disponível em: <http://www.zdnet.com/smart-tvs-may-be-taking-off-but-theyre-still-not-smart-enough-7000008042/ >. Acessado em: mar. 2014.

[8] Teixeira, C.A.C.; Cédrick, B.N; Santos, C.A.S, Melo, E.L. Mechanisms of synchronization for multimedia applications. 2015.

[9] Wohlin, C; Runeson, P; Host, M; Ohlsson, M.C; Regnell, B; Wesslén, A. Experimentation in Software Engineering: an introduction. Kluwer Publishers, 2000.

Unpacking a Cluster of Modular Robots

S. Wong, S. Zhu, and J. Walter
Computer Science Department, Vassar College, Poughkeepsie, NY, USA

Abstract— *The problem addressed is the reconfiguration of a system of hexagonal metamorphic robots from an initial arbitrary shape configuration $\mathcal{I}$, to an overlapping straight chain goal configuration, $\mathcal{G}$. We first present algorithm* ONE-DIR *that accomplishes shape change using local contact information and global knowledge of the coordinate of the cell in $\mathcal{I}$ that overlaps $\mathcal{G}$ and direction of tilt of the chain $\mathcal{G}$. No preprocessing or algorithm-generated message passing is used in the* ONE-DIR *algorithm. We compare algorithm* ONE-DIR *to the simpler, but more widely applicable, algorithm we presented in [1] (we call this the* BUTTERFLY *algorithm). We test and compare the performance of algorithms* ONE-DIR *and* BUTTERFLY *using a discrete-event simulator. Our experimental results show that algorithm* ONE-DIR *is more efficient, in terms of time for reconfiguration and number of module moves, than the* BUTTERFLY *algorithm.*

Keywords: Hexagonal metamorphic robots, self-reconfiguration

1. Introduction

Modular robotic systems are composed of one or more robots that connect or release bindings to accomplish a task. The completion of given tasks is made possible by the resultant shape and connectivity of the system. Flexibility in the number of possible target configurations attainable allows such systems to be useful in applications that require modules to occupy different coordinates or spatial arrangements with respect to other modules over time.

When individual robots are homogenous, these systems are known as *self-reconfigurable* (SR), self-repairing, or self-healing. In SR systems, a module senses the position of adjacent modules and follows a set of rules based on contacts with neighboring modules in order to move to a different position in relation to its current environment. It is important that module collisions do not occur during the reconfiguration process; even contact between moving modules must be avoided.

When SR robotic systems are composed of homogenous robots with a regular polygonal shape such as squares or hexagons, the systems are called *metamorphic*. A metamorphic SR robotic system (MSR) is generally composed of a sufficiently large number of modules so that the system can assume many shapes in two- or three-dimensional space, with any module capable of filling any position in any configuration. The ability to effect system-wide shape change in order to facilitate a variety of tasks is one of the main reasons MSR systems are being studied.

Intuitively speaking, reconfiguration algorithms for MSR systems are concerned with transferring a clump of modules (the initial configuration, $\mathcal{I}$) to a different location in the plane (the goal configuration, $\mathcal{G}$), using concurrent movement of individual modules over the periphery of the clump of stationary modules. In our work, all modules are assumed to have identical shape, size, and computational abilities so that any module in $\mathcal{I}$ can be used in any position in $\mathcal{G}$.

This paper presents a new deterministic algorithm that can reconfigure a system of hexagonal metamorphic robots (HMSRs). Our algorithm causes modules to change from a system with an arbitrary (but well-defined) shape to a straight chain. The algorithm requires that modules in $\mathcal{I}$ know the coordinates of the cell in which $\mathcal{I}$ and $\mathcal{G}$ overlap and the direction of the target goal chain, making the algorithm scalable even if very large numbers of modules participate in the reconfiguration. Each module also has the sensory capability to determine the state (occupied or unoccupied) of its 6 adjacent grid cells in the initial formation and at the start of each round.

In the system we model, each hexagonal-shaped module requires an unmoving substrate lattice of identical modules to move across. Modules accomplish movement by changing joint angles as well as releasing and reforming inter-module connections [2]. We speculate that such deformable modules could be fashioned using new smart materials, such as flexible CPUs, to accommodate the range of deformation needed. More recent work by [3] presents the idea of rigid hexagonal modules with sliding motion, but with motion constraints similar to the ones in [2]. Our algorithms will work for this type of module motion as well.

Our algorithms ensure the system will have a dynamically changing shape, while at the same time guaranteeing that a surface of stationary modules exists between each individual module in $\mathcal{I}$ and a cell in $\mathcal{G}$ as the modules fill $\mathcal{G}$. The path between a particular module in $\mathcal{I}$ and its final position in $\mathcal{G}$ is formed just in time for the module to reach that position. The ONE-DIR algorithm we present in this paper has extremely simple local rules for module motion and requires no pre-processing phase. Furthermore, the reconfiguration can be done without message passing and it requires minimal module sensory capabilities.

*This material is based upon work supported by the National Science Foundation under Grant No. 0712911.

Metamorphic systems promise to be useful, for example, in situations that require movement of modules over a surface where minimal human intervention is possible. Floating modules used for human shelters or life rafts could be linked on the surface of the ocean, changing shape for different weather conditions or reconfigured as occupants changed dwellings. Smart solar panels composed of individually mobile modules could move across a roof [4] or the surface of the ocean (e.g., "Hexifloats" [5]) in order to spend the optimal amount of time in direct sunlight or to remain connected while riding out a storm. If used in conjunction with new techniques in human tissue generation, nano-sized modules could be injected into the human body to deliver tissue to a ruptured organ or blood vessel, acquiring the different shapes necessary to navigate complex routes. Such technology might also have applications in the arts, where, for example, installations composed of HMRS's could form different pictures by moving across a wall or floor.

As an example application, consider a system of self-reconfigurable solar panels on the roof of a building. Depending on the distribution of sunlight and shadows, the modules forming the platform could move so that they spend the most time in direct sunlight. Having a limited number of self-reconfigurable panels could be potentially more cost-effective than having under-utilized stationary panels, many of which may not used at different times of the day. Metamorphic systems would also be useful in environments that do not support human life processes such as outer space, toxic waste sites, on or under water, and in nano-scale environments. Karagozler et al. present a system of modules that change shape to provide support for emergency maintenance operations in space [6]. Nano-scale modules have been proposed that, when massed together, become tools for anything from medical procedures to household tasks [7].

2. Related work

Many centralized and decentralized algorithms have been proposed to change the shape of modular systems. Our previous work has addressed the reconfiguration of metamorphic robots from a straight chain configuration to an arbitrary goal configuration ([8], [9]), a process that may require traversal of very irregular surfaces [10] and envelopment of obstacles [11].

Butler et al. worked with a similar rule-based algorithm for square modules [12]. Their paper discusses using their algorithm for hexagonal modules also, but the probabilistic algorithm is limited in terms of its physical constraints on modules.

Miao et al. [3] presented a probabilistic distributed algorithm for target envelopment that uses a curve-shortening technique. Their algorithm also applies to HMSR systems, but with different motion constraints and communication assumptions than are assumed in this paper.

Recent work by Larkworthy et al. [13] demonstrated a time complexity of $O(n)$ for probabilistic reconfiguration of large HMSR systems for algorithms in which at most one module can move in a time step. Larkworthy, et al. used results presented by Ghrist [14] to propose an efficient probabilistic algorithm for multi-move reconfigurations.

Nguyen et al. developed a greedy algorithm that allows for any admissible chain configuration to be transformed into any other admissible chain configuration with the same number of modules [15]. Nguyen's algorithm is centralized while the algorithm presented in this paper is decentralized.

Recent work by Lakhlef, et al. [16], discusses probabilistic reconfiguration of MEMS microrobots. The algorithm they present requires no knowledge of the target configuration and uses message passing to coordinate module movements. In this paper, we present scalable deterministic algorithms that use only the knowledge of a single goal cell and the direction of the goal chain in relation to the initial configuration.

A navigation approach for reconfiguring HMRSs is presented by Zada, et al. [17]. The algorithms assume that modules are independently mobile (needing no substrate) and that modules communicate during reconfiguration. Both of these assumptions make the algorithms in [17] significantly different than the ones described in this or in any of our earlier papers [18], [9], [10], [11], and [19], and therefore comparisons are not meaningful.

3. System Model

3.1 Reconfiguration space

The plane is a lattice partitioned into equal-sized hexagonal cells and labeled using the coordinate system described by Chirikijian [2].

We assume modules in $\mathcal{I}$ are oriented such that each has a flat surface facing south (S) and north (N). This way we can unambiguously describe the east (E) most, west (W) most, inner, and perimeter columns of $\mathcal{I}$.

3.2 Module Assumptions

Each hexagonal module is identical, including sensors, computing capability and motion constraints, and each runs the same algorithm. Each module is the same size as a cell in the lattice and modules always fit within a cell when they are not moving and at the end of each round (i.e., a module cannot be in-between cells of the lattice at the beginning or end of a round of module movement). In terms of specifications, the robot must have the ability to detect robots that are in contact with any of its sides.

The number of modules in $\mathcal{I}$ is assumed to be equal to the number of cells in $\mathcal{G}$.

Physical module constraints for this algorithm include:

1) A module must have a minimum of two adjacent free (unoccupied) sides in order to move.

2) A module must have an adjacent non-moving substrate module to rotate around or slide over.
3) Modules cannot carry, push, or pull other modules, i.e., a module is only capable of moving itself.
4) Each module knows, at all times
   - its location (the coordinates of the cell that it currently occupies),
   - its orientation (which edge is facing in which direction),
   - its neighboring cells that are occupied by other modules using touch sensors, and
   - if a module has not yet moved, it knows which of its neighboring modules have and have not moved during an execution of the algorithm.
5) Modules can either rotate clockwise (CW) or counter-clockwise (CCW) or slide across two faces of a substrate to achieve the same effect. Rigid modules move by sliding over two faces of a substrate [3] while retaining the original orientation of each module (cf. Fig. 1(a)–(e)). Deformable modules move by a combination of rotation and changing joint angles[2] (cf. Fig. 1(f)–(j)).

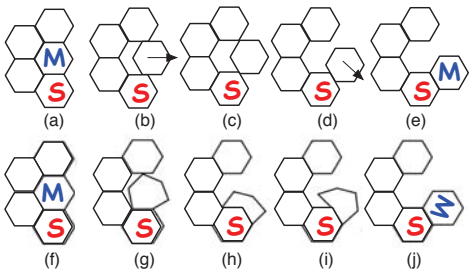


Fig. 1: Movement of rigid module M over substrate S, (a)–(e); movement of deformable module M over substrate S, (f)–(j). For each type of module, the motion constraints are the same.

3.3 Reconfiguration Environment

We assume there exists initially a straight chain goal configuration $\mathcal{G}$ that intersects the initial configuration $\mathcal{I}$ in a single cell. The straight chain $\mathcal{G}$ must intersect with the E-most column of the admissible initial configuration $\mathcal{I}$ in exactly one cell and $\mathcal{G}$ must be oriented from this intersection to either the NE or SE of the cell that intersects $\mathcal{I}$. All modules participating in the reconfiguration know the coordinates of the initial overlapping cell and the direction of the straight chain $\mathcal{G}$ in relation to $\mathcal{I}$. Our algorithms are shown to be correct if all the modules initially in an admissible configuration $\mathcal{I}$ stop moving such that each one is in a cell of $\mathcal{G}$.

$\mathcal{I}$ is *admissible* if

1) every column of $\mathcal{I}$ is contiguous from north (N) to south (S) (i.e., there are no holes in $\mathcal{I}$),
2) every module in a column of $\mathcal{I}$ that is not the W-most column initially has either a module adjacent to its NW, SW, or both NW and SW, and
3) every module in a column of $\mathcal{I}$ that is not the E-most column initially has either a module adjacent to its NE, SE, or both NE and SE sides.

This definition of admissibility is more restrictive than it was in the BUTTERFLY algorithm [1], which does not require all modules in $\mathcal{I}$ to have adjacent modules to either the NE or SE. Fig. 2 (a) gives an example of a configuration that is inadmissible for both the BUTTERFLY and ONE-DIR algorithms, (b) an example of a configuration admissible in the BUTTERFLY algorithm but not in the new algorithm, and (c) an example of a configuration that is admissible both for the BUTTERFLY and for the algorithm presented in this paper.

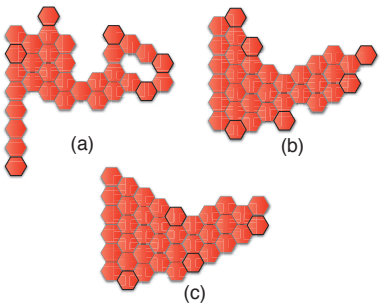


Fig. 2: Initial configurations. The configuration in (a) violates all admissibility requirements, the configuration in (b) is admissible in the BUTTERFLY algorithm, and the configuration in (c) is admissible in both the BUTTERFLY and our new algorithm.

4. Algorithm ONE-DIR Overview

Minimal preprocessing is necessary and no algorithm-generated message passing occurs between the modules in an execution of the ONE-DIR algorithm. An execution starting from an admissible $\mathcal{I}$ is deterministic and finite, lasting from the first time step in which a module moves until the time step in which each module is in a cell in $\mathcal{G}$.

Modules move in synchronous rounds, with as many modules moving in a round as possible. Specific conditions, such as the direction of the module's last move, its current neighboring modules and the direction of the goal chain cells are used in the algorithm to ensure that a module will never collide with another module nor will any module movement divide the system into disjoint connected components.

The algorithm has a module at the N or S of the W-most column (the column furthest from the column that intersects $\mathcal{G}$) begin to move either clock-wise (CW) or counter-clock-wise (CCW), with the rest of the modules in that column moving in the same direction when their formerly occupied neighboring cell is empty.

In [8] and [9], we showed that configuration time is expedited when modules move in both CW and CCW directions,

i.e., bidirectionally. However, correctness requires that our algorithms ensure collisions are not possible when modules enter goal cells. In [1], modules were assumed to have a one-cell look-ahead ability perpendicular to each flat side, to ensure that either the module entering the goal from the N or S delayed one time step, thereby avoiding collision.

Initially, we attempted to write an algorithm in which modules moved both CW and CCW around the perimeter of $\mathcal{I}$, but this strategy required some modules to delay one or more time steps after starting to move (as does the BUTTERFLY algorithm), to avoid collision in goal cells. Any module delay slows down the overall reconfiguration and therefore, in the more time efficient new algorithm presented in this paper, all modules move in the same direction (either CW or CCW), maintaining one empty cell between moving modules and requiring no module delays once movement starts. Modules initially choose which direction to move based on the angle of intersection between $\mathcal{I}$ and $\mathcal{G}$, since moving toward an obtuse angle intersection requires no wait time for any modules with single-cell spacing, as we showed in [8] and [10]. Once moving, modules move in every timestep until they occupy a goal cell, when they stop.

4.1 Algorithm ONE-DIR Details

Each module checks in each time step before it initially begins moving whether or not it is free to begin movement. Free modules have two or three consecutive sides with neighbor contacts and three or four consecutive sides that are not adjacent to occupied cells, as shown in Fig. 3.

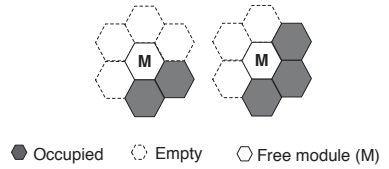


Fig. 3: Local configurations in which a module M is free to begin moving CW. Flip each part vertically for configurations in which a module is free to begin moving CCW.

```
Algorithm ONE-DIR
 1. if (not moved):
 2.       STARTMOVINGIFPOSSIBLE()
 3. else:      // module has moved
 4.       if (not INGOAL()):
 5.            MOVE(dir)
```

Fig. 4: Algorithm run at each module on each time step of reconfiguration.

Here are definitions of variables maintained by the algorithm at each processor:

- *moved* - a boolean variable that is true if the module is moving in the round and false if it has not yet moved or if it is in a goal cell. Initially, this variable is false at every module.
- *goalDir* - the direction in which the goal chain configuration is pointed in relation to $\mathcal{I}$, NE or SE.
- *dir* - the direction in which a module moves, either CW or CCW, over an adjacent substrate module. Choosing the direction of movement depends on *goalDir* because all modules will move toward the obtuse-angle intersection between $\mathcal{I}$ and $\mathcal{G}$. If *goalDir* is NE, *dir* will be CCW, and if *goalDir* is SE, *dir* will be CW.
- *firstGoal* - the cell that is initially both $\in \mathcal{I}$ and $\in \mathcal{G}$.

```
Procedure STARTMOVINGIFPOSSIBLE()
 1. if (goalDir == NE):
 2.       if ((ALLINITCONTACTS() and EMPTYWESTAND(S)) or
             (INFIRSTGOALCOLUMN() and EMPTYWESTAND(N))):
 3.            moved = true
 4.            dir = CCW
 5. else if (goalDir == SE):
 6.       if ((ALLINITCONTACTS() and EMPTYWESTAND(N)) or
             (INFIRSTGOALCOLUMN() and EMPTYWESTAND(S))):
 7.            moved = true
 8.            dir = CW
```

Fig. 5: Procedure STARTMOVINGIFPOSSIBLE().

Other functions used in reconfiguration algorithm ONE-DIR:

- ALLINITCONTACTS(): Predicate called by module i that has not started moving. Returns true only if all modules currently in contact with i have not yet moved.
- INFIRSTGOALCOL(): Predicate called by module i that has not started moving. Returns true only if module i is in same column as *firstGoal* and false otherwise.
- EMPTYWESTAND(X): Predicate called by each module i that has not started moving. Returns true if cells to the NW, SW, and X direction are empty, for X either N or S.
- MOVE(*dir*): Function causing module to move in either a *CW* or *CCW* direction over a substrate module. This predicate is called by each module for which *moved* is true in every timestep in which the module is not in a goal cell.
- INGOAL(): Predicate that returns true if the coordinates of the cell a module currently occupies are on a straight line starting at *firstGoal* in direction *goalDir*. Algorithm execution is terminated at any module for which this predicate is true.

Each module runs the same algorithm in every time step and every module rotates the same direction. The module at the N or S end of the W-most column of modules in $\mathcal{I}$ begins moving over its adjacent neighbor to the E. At the start of reconfiguration, the only module that is allowed to move is the one that has a free local configuration (see Fig. 3), no adjacent modules to the NW or SW, and a free space to either

the N, NE, N and NE, or S, S and SE, or SE, depending upon the *goalDir*. After moving for the first time, each module moves in the same direction in every time step in which it is not in $\mathcal{G}$. All modules move toward the obtuse angle side of the intersection of $\mathcal{I}$ and $\mathcal{G}$, with all modules moving either CW or all moving CCW.

5. Simulation and Analysis

Fig. 6 demonstrates executions during execution of the BUTTERFLY and ONE-DIR algorithms, where the modules are moving from left to right. The BUTTERFLY algorithm forms columns of modules on either side of $\mathcal{G}$, requiring one-cell look-ahead capabilities and associated delay so modules do not collide in $\mathcal{G}$. The ONE-DIR algorithm does not need to use these formations or any delay, since all modules move in the same direction.

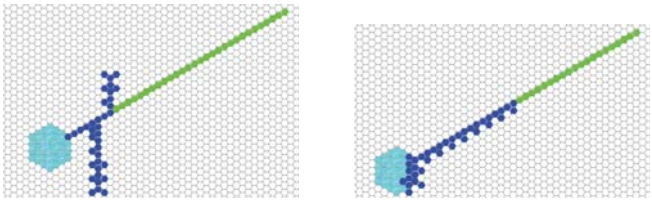


Fig. 6: examples of modules moving in BUTTERFLY algorithm (left) and ONE-DIR algorithm (right).

We simulated the execution of each algorithm on a set of different shapes, examples of which are shown in Fig. 7, with an increasing number of modules in each experimental run.

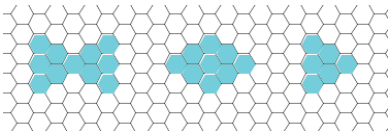


Fig. 7: Examples of bowtie, diamond and wedge configurations.

We simulated both the BUTTERFLY and ONE-DIR algorithms and the results are shown in Figs. 8, 9, and 10. There were slight differences in the time and number of moves for a particular size and shape of $\mathcal{I}$, depending on the intersection point and the direction of tilt of the goal chain. The results shown in the graphs represent an average for number of timesteps and moves with alternate intersections and directions of $\mathcal{G}$.

Algorithm ONE-DIR outperforms the BUTTERFLY algorithm both in number of timesteps and the combined number of module moves. The advantage of the BUTTERFLY algorithm is that it works for a larger number of initial configurations. Both algorithms require modules to know only a single goal cell coordinate and the direction of the goal chain, so both are scalable.

Correctness of the ONE-DIR algorithm requires that there is no module collision or deadlock and that all modules initially

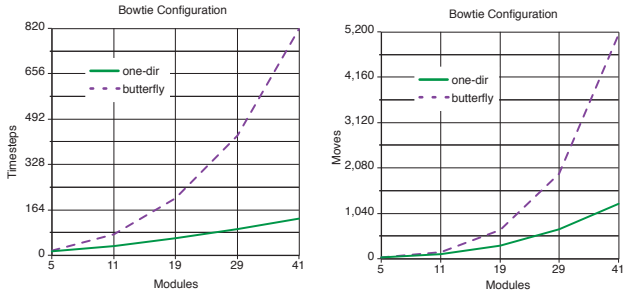


Fig. 8: Comparison of ONE-DIR and BUTTERFLY on bowtie shape.

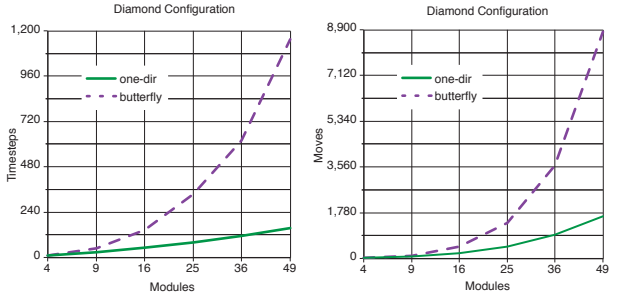


Fig. 9: Comparison of ONE-DIR and BUTTERFLY on diamond shape.

in $\mathcal{I}$ stop in a cell in $\mathcal{G}$. Collision occurs when two modules attempt to move into the same cell from different directions. Deadlock occurs when the rules for movement never apply to a module's local configuration, such as is the case when one module is adjacent to a moving module.

Module collision is not possible at the start of reconfiguration with ONE-DIR because modules choose direction based on the direction of the goal chain. Because of the required smoothness of admissible initial configuration surfaces and the direction of the chain $\mathcal{G}$, at most one module will move at the start of execution and other modules will follow with one-cell spacing between moving modules. Modules can distinguish adjacent moving modules from initial neighbors, so no module will start moving when it is adjacent to a moving module. Module collision in unoccupied goal cells is not possible due to the orderly fashion in which modules start to move, combined with the common direction chosen by all modules based on the angle at which the goal chain meets the initial configuration. Since modules move toward the obtuse angle intersection of $\mathcal{I}$ and $\mathcal{G}$, moving modules will not come into contact in the angle formed by the $\mathcal{I}$ and $\mathcal{G}$ intersection.

Moving modules will not be trapped against modules in $\mathcal{I}$ that have not yet begun to move because of the required smoothness of $\mathcal{I}$ and the second admissibility requirement, requiring every module in $\mathcal{I}$ except those in the west-most column to have neighboring modules to the NW, SW, or both NW and SW. Therefore, modules will not come into contact

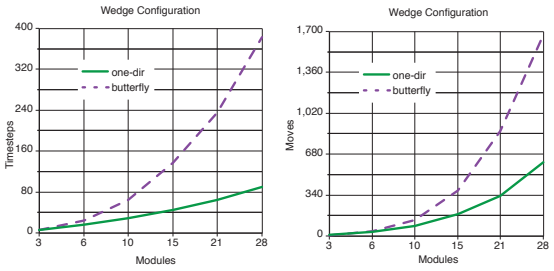


Fig. 10: Comparison of ONE-DIR and BUTTERFLY on wedge shape.

when moving over the surface of $\mathcal{I}$ nor when moving over the surface of modules already in $\mathcal{G}$.

6. Conclusions and future work

We have presented an algorithm that uses no preprocessing and only local contact and sensor information at each module to move modules from a set of admissible arbitrary initial configurations into straight chain goal configurations. Our algorithm is collision- and deadlock-free when run on an admissible initial configuration. This algorithm is another step toward the realization of a complete, deterministic planner for the reconfiguration of hexagonal metamorphic robots, using no message passing between modules, and the algorithm is scalable.

We are currently developing deterministic algorithms that use either preprocessing, sensory information, or both, to reconfigure an arbitrary initial configuration into a straight chain goal configuration. We aim for these algorithms to work on a larger subset of initial configurations by relaxing the assumptions on the admissibility of $\mathcal{I}$.

Acknowlegements

We thank Vassar College students Jonathan Gorman for early work on the algorithms and George Whiteside for his help running experiments on our discrete event simulator.

References

[1] S. Wong and J. Walter, "Deterministic distributed algorithm for self-reconfiguration of modular robots from arbitrary to straight chain configurations," in *Proc. IEEE Intl. Conf. Robotics & Automation*, 2013, pp. 537–543.

[2] G. Chirikjian, "Kinematics of a metamorphic robotic system," in *Proc. IEEE Intl. Conf. Robotics & Automation*, 1994, pp. 449–455.

[3] Y. Miao, G. Yan, and Z. Lin, "A distributed reconfiguration strategy for target enveloping with hexagonal metamorphic modules," in *Proc. IEEE Intl. Conf. Robotics & Automation*, 2011, pp. 4804–4809.

[4] M. Kanellos, "An ikea for solar?" *greentech solar*, 2009.

[5] H. Ali, "The power flower," *Photovoltaic Markets and Technology: Special Issue on Applications and Installations*, vol. 5, pp. 136–140, 2011.

[6] M. Karagozler, B. Kirby, W. Lee, E. Marinelli, T. C. Ng, M. Weller, and S. Goldstein, "Ultralight modular robotic building blocks for the rapid deployment of planetary outposts," in *Revolutionary Aerospace Systems Concepts Academic Linkage Forum*, 2006, pp. 1–15.

[7] B. Kirby, B. Aksak, J. Campbell, J. Hoburg, T. Mowry, P. Pillai, and S. Goldstein, "A modular robotic system using magnetic force effectors," in *Proc. IEEE Intl. Conf. Int. Rob. & Syst.*, 2007, pp. 1–7.

[8] J. Walter, J. Welch, and N. Amato, "Distributed reconfiguration of metamorphic robot chains," *Springer-Verlag Journal on Distributed Computing*, vol. 17, pp. 171–189, 2004.

[9] J. Walter, B. Tsai, and N. Amato, "Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 621–631, 2005.

[10] P. Ivanov and J. Walter, "Layering algorithm for collision-free traversal using hexagonal self-reconfigurable metamorphic robots," in *Proc. IEEE Intl. Conf. Intell. Rob. & Systs.*, 2010, pp. 521–528.

[11] S. Matysik and J. Walter, "Using a pocket-filling strategy for distributed reconfiguration of a system of hexagonal metamorphic robots in an obstacle-cluttered environment," in *Proc. IEEE Intl. Conf. Robotics & Automation*, 2009, pp. 4265–4272.

[12] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for a class of self-reconfigurable robots," in *Proc. IEEE Intl. Conf. on Robotics & Automation*, 2002, pp. 809–816.

[13] T. Larkworthy and S. Ramamoorthy, "An efficient algorithm for self-reconfiguration planning in a modular robot," in *Proc. IEEE Intl. Conf. Robotics & Automation*, 2010, pp. 5139–5146.

[14] R. Ghrist, "Shape complexes for metamorphic robot systems," in *Workshop in Algorithmic Foundations of Robotics*, 2002.

[15] A. Nguyen, L. J. Guibas, and M. Yim, "Controlled module density helps reconfiguration planning," in *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, 2000, pp. 23–36.

[16] M. Lakhlef, H. Mabed, and J. Bourgeois, "An energy and memory-efficient distributed self-reconfiguration for mems microrobots," in *22nd Euromicro Intl. Conf. on Parallel, Distributed and network-based Processing*, 2014, pp. 154–161.

[17] F. Zada, H. El-Deen, and Y. Dahab, "A navigation approach for configuring distributed hexagonal metamorphic autonomous mobile robots," *CiiT International Journal of Artificial Intelligent Systems and Machine Learning*, vol. 6, no. 07, pp. 256–262, 2014.

[18] J. Walter, J. Welch, and N. Amato, "Concurrent metamorphosis of hexagonal robot chains into simple connected configurations," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 6, pp. 945–956, 2002.

[19] D. Little and J. Walter, "Using hexagonal metamorphic robots to form temporary bridges," in *Proc. IEEE Intl. Conf. Intell. Rob. & Systs.*, 2005, pp. 2652–2657.

Applications of Petri Nets in Distributed Processing: a Scoping Study

S. J. Bachega[1], and D. M. Tavares[2]
[1]Production Engineering Department, Federal University of Goiás, UFG, Catalão, Goiás, Brazil
[2]Computer Science Department, Federal University of Goiás, UFG, Catalão, Goiás, Brazil

Abstract - *Petri Nets has been applied in several contexts since its creation. Among the classical applications are communication protocols, manufacturing systems, control system, software development, parallel algorithms, and robotic systems. The aim of this paper is to initiate a scoping study to collect and compare existing practical applications of Petri nets in distributed processing. We found the use of a great variety of Petri Nets applied to distributed processing, such as airport simulation, language definition, manufacturing simulation, and some derivative works which use some of the established theories presented. This review does establish a baseline, which we wish will be used by other researchers.*

Keywords: Petri nets; distributed processing; application; scoping review

1 Introduction

Petri nets give a uniform environment for the design of discrete event systems, modeling and formal analysis, considering its uses as a mathematical and graphical tool [1]. They have weight, label, directed graphs, and tokens that can have dynamic properties [2]. A formal definition of Petri net was shown in [3], and is replicated in TABLE I.

TABLE I. FORMAL DEFINITION OF PETRI NET [X3]

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:

$P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,
$T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
$W: F \rightarrow \{1,2,3,\ldots\}$ is a weight function,
$M_0: P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking,
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by N.

A Petri net with the given initial marking is denoted by (N, M_0).

There are several advantages to Petri nets, and a major one is that the same model can be used for performance evaluation, analysis of behavioral properties, systematic creation of discrete event simulators and controllers [1]. From the original Petri net definition, other studies were proposed and changed some of its features. These modifications were done to make the use of Petri nets more suitable to a wide range of problems with other challenges. Emerging modified Petri nets include Time Petri Nets, Stochastic Petri Nets, Colored Petri Nets and Fuzzy Petri Nets.

Since its publication by Carl Adam Petri in 1962, Petri nets are applied in many contexts such as communication protocols [4], manufacturing systems [5], control system [6], software development [7], parallel algorithms [8], robotic systems [9], etc. In this paper, we focus in distributed processing applications.

To define a distributed process, we first must understand the context of process creation in an operating system. The creation of a process in an operating system is usually an indivisible task involving a system call and the resources of a single computer. When we consider the features involved in distributed processes, the design of the process' creation mechanism needs to take into account the use of several computers; and hence, the process' support infrastructure is separated in several system tasks. These tasks can be separated in two independent aspects: the choice of the destination host (e.g. chosen amongst the nodes in a cluster, which act as computer servers); and in the creation of the execution environment for a distributed process (and an initial thread inside it) [10].

In order to map the current knowledge in the field, the purpose of this paper is to initiate a scoping study to collect and compare existing practical applications of Petri nets in distributed processing.

A scoping study, also known as scoping review, is a widely used approach to reviewing health research evidences [11]. Although the initial use was intended to health research, other areas started to use this approach to observe their importance and promising results. Examples of uses in related fields of computer science can be found in [12-16].

Even though there is no universal definition of scoping study [17], according to [18], the scoping study comprises another type of literature review and generally aims to map relevant literature in a given field of interest. The common reasons to do scoping researches include to investigate the nature, to extend the range of a research activity; to resume and propagate research findings; to verify the potential of undertaking a complete systematic review; and to find research gaps in the existing literature [18]. This approach inclines to address wider topics where several different study designs might be applicable, and does not care about very specific research questions or to assess the quality of the included studies. These issues differs scoping study from systematic literature review [18].

This paper is organized as follows: Section 2 describes the method, section 3 presents the results (answers to the research questions) and section 4 shows the conclusions.

2 Method

This paper has been undertaken as a scoping study based on the guidelines proposed by [17, 18]. The main steps followed are described below [18].

2.1 Identifying Research Question (Stage 1)

In this stage, it is necessary to start the process with the definition of the research question. Research questions guide the subsequent phases of the research and should be clearly defined. They have a broad nature [17, 18].

The research question of the present study is: What is known from the research results communications between 2000 and 2015 about practical applications of Petri nets in distributed processing?

2.2 Identifying Relevant Studies (Stage 2)

The identification of primary studies (published or unpublished) and reviews should be as comprehensive as possible. The searching strategy for the research evidence should cover different sources [18].

In order to gain a broad perspective, we searched in electronic sources, as suggested by [19]. The benefits of searching databases is highlighted by [20]. The following databases were covered: IEEE Xplore, Compendex, ScienceDirect, ACM Digital Library, Springer LNCS and digital proceedings sites. As confirmed by [21, 22], these databases cover the most important journals, conferences proceedings and workshops in computer science.

It is necessary to justify the decisions to limit the scope and recognize the boundaries of the study [17]. The scope of this research is to identify the formal communications of research results that contribute to check the research directions in Petri nets applied in distributed processing in the last fifteen years. The reason of this scoping study is to summarize and disseminate research findings in the proposed theme.

The search covered the period from 2000 to 2015 to ensure that the most relevant research within the field would be included. The keywords used were: Petri nets, distributed processing, parallel processing, application, scoping study. These key words were used with possible combination and alone.

2.3 Study Selection (Stage 3)

Stage 3 involves defining *post hoc* inclusion and exclusion criteria to select the papers used in this study. The *post hoc* feature ensures greater familiarity with the literature improving the selection based on relevance [18].

The following inclusion criteria was applied:

- Primary studies published in proceedings of conferences, workshops and journals, representing applications of Petri nets in distributed processing.
- The discussion results and analysis must be focused only on practical applications of Petri nets in distributed processing.

The defined exclusion criteria were:

- Secondary Literature Reviews.
- Abstracts.
- Overlapped papers issued by searches in the selected sources.
- Duplicate publications of the same study.
- Works in progress, technical reports, and some workshop reports ('grey' literature).

We excluded 'grey' literature from analysis because the quality of these literatures is more difficult to assess and the volume of the studies included in the first searches would have grown unreasonably [21].

In addition to this, we include only the most complete version of a study when several reports of the study exist in different journals (duplicate publications of the same study). Only studies written in English are included [21]. These decisions had to be made due to time span and budget constraints. Two reviewers applied the exclusion and inclusion criteria to identified citations.

2.4 Charting the Data (Stage 4)

Charting the data involves the extraction of data from each study as a narrative review [18, 23].

The data collected from each citation were [18]:

- Author(s), year of publication, study location.
- Aims of the study.
- Application context.
- Method.
- Main results.

The data-charting was conducted by one researcher and verified by another. We discussed the subjects when there were a point of disagreement. Distribution was based on the time availability of each researcher. This method is supported by [24, 25]. The data was tabulated to show the items presented in section 3.

2.5 Collating, summarising and reporting the results (Stage 5)

This stage, in scoping study, pursues to show an overview of all citations reviewed. Decisions about the best way to introduce this material were critical. There are two ways to present these findings: basic numerical analysis of the nature, extent and distribution of the material included in the study; and organizing the literature thematically [18].

3 Results

This section summarizes the results of this research.

3.1 Categorization of Selected Articles

We selected 13 papers that apply Petri Nets (original or modified) in distributed processing according to the definition of [10]. TABLE II shows the Publication ID, authors, date and source of the publications.

TABLE II. LIST OF SELECTED STUDIES

ID	Author	Date	Source
P1	Bertrand, Carle and Choppy [26]	2009	International ICST Conference on Simulation Tools and Techniques
P2	Boufaied, Subias and Combacau [27]	2002	International Conference on Systems
P3	Fang, Xu and Yin [28]	2007	International Conference on Natural Computation
P4	Iwaniak and Khadzhynov [29]	2014	Communications in Computer and Information Science
P5	Martiník [30]	2013	International Conference on Informatics and Applications
P6	Mazouzi, Mbarek, Kallel and Abid. [31]	2012	International Conference on Parallel and Distributed Processing Techniques and Applications
P7	Spiegel et al. [32]	2009	Procedia Earth and Planetary Science
P8	Wolfmann and Giusti [33]	2013	International Conference on Parallel and Distributed Processing Techniques and Applications
P9	Wolfmann and Giusti [34]	2014	International Conference on Parallel and Distributed Processing Techniques and Applications
P10	Yasuda [35]	2008	International Conference on Automation and Logistics
P11	Yasuda [36]	2009	ICCAS-SICE 2009 - ICROS-SICE International Joint Conference
P12	Yasuda [37]	2010	Lecture Notes in Computer Science
P13	Yasuda [38]	2011	SICE Annual Conference

The earliest study actually included was published in 2002. Fig. 1 presents the frequency of papers per year. As we can see, most of the papers in this issue were published in 2009 (three studies). Note that as of 2007, there was at least one article published in the issue at hand. It is noteworthy that in 2015, until the completion period of this article, no research had been published with the application of Petri nets in distributed processing.

The frequency of papers per source of publication is exposed in Fig. 2. Note that the highest frequency of publications in the theme of this research was at the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). In addition to this, most of the publications identified were published in conference proceedings.

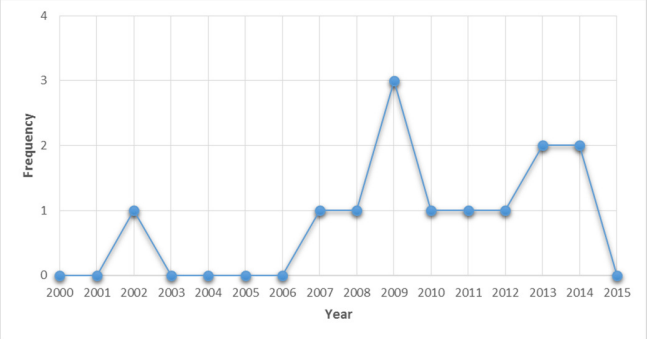


Fig. 1. Frequency of Papers per Year

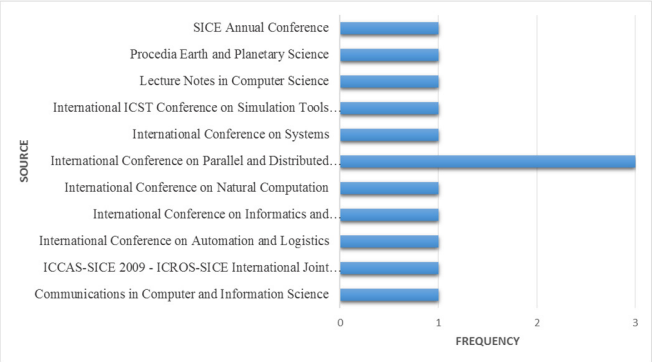


Fig. 2. Frequency of Papers per Source

It was found that approximately 31% (4 papers) of the analyzed papers, was elaborated by a researcher of the Nagasaki Institute of Applied Science in Japan (TABLE III). Two Argentinian researchers, one from Universidad Nacional de Córdoba and the other from Universidad Nacional de La Plata, jointly developed two papers with applications of Petri nets in distributed processing.

TABLE III. INSTITUTIONAL AFFILIATION OF RESEARCH

ID	Study Location	Researcher's Institution
P1	France	Université Paris XIII
P2	France	Université Paul Sabatier / Institut National des Sciences Appliquées de Toulouse
P3	China	Anhui University of Science and Technology / Chuzhou University
P4	Poland	Technical University of Koszalin
P5	Czech Republic	VŠB-Technical University of Ostrava
P6	Tunisia	Ecole Nationale d´Ingénieurs de Sfax / National School of Engineering of Tunis
P7	Germany	Universität Duisburg-Essen
P8	Argentina	Universidad Nacional de Córdoba / Universidad Nacional de La Plata
P9	Argentina	Universidad Nacional de Córdoba / Universidad Nacional de La Plata
P10	Japan	Nagasaki Institute of Applied Science
P11	Japan	Nagasaki Institute of Applied Science
P12	Japan	Nagasaki Institute of Applied Science
P13	Japan	Nagasaki Institute of Applied Science

3.2 Identified Applications of Petri Nets in Distributed Processing

The most used modification of the Petri net original concept was Coloured Petri nets (approximately 38% of the papers – 5 papers). The Petri Net Based Multitask Processing was the second most cited (Fig.3). The type of Petri nets used in each paper and its aims are in TABLE IV.

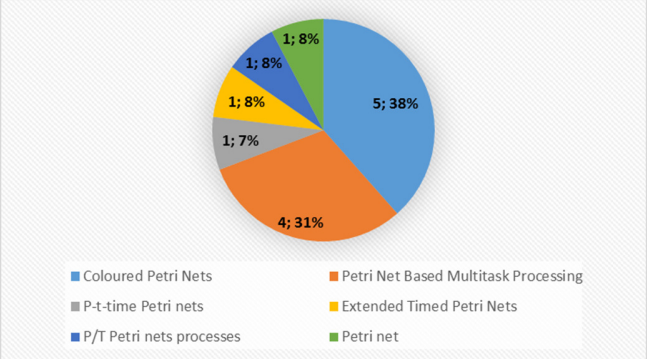


Fig. 3. Frequency and Percentage of Citation Type of Petri Nets

TABLE IV. AIMS OF THE STUDIES AND TYPES OF PETRI NETS

ID	Aim of the study	Type of Petri Nets
P1	To design a modeling and simulation infrastructure for the acquisition and analysis of airport operational concepts and equipment.	Coloured Petri nets
P2	To deal with the problem of detecting failure situations in several monitoring sites.	P-t-time Petri nets
P3	To present an improved mapping method based on existing Petri nets in order to achieve a better representation of the parallel or distribution simulation mechanism applied to the design and analysis of concurrent discrete event systems.	Extended Timed Petri Nets
P4	To develop a modeling framework, which could allow the system designer to create a conceptual model of the problem being solved and then transform it into a Petri Net model for a more detailed analysis.	Coloured Petri Nets
P5	To generalize the Property-preserving Petri net process algebras (PPPA) for the special class of the P/T Petri nets processes, and to define additional PPPA composition operator SYNC-CALL and prove its chosen properties.	P/T Petri nets processes
P6	To propose and model a switched fabric CAN network Architecture based on CAN Controllers and switched fabric by the use of timed colored Petri nets (CPNTools).	Coloured Petri Nets
P7	To introduce a realization concept for a CR which forms the basis of a hardware/firmware demonstrator developed by the authors.	Petri net
P8	To introduce Petri Nets as a tool for simplifying the modeling and execution of parallel asynchronous versions of this kind of algorithms, while using an efficient dynamic task scheduling implementation.	Coloured Petri Nets and Token Petri Net
P9	To introduce an asynchronous Parallel Execution Model based on Petri Nets and the process to go from a high level model to an executable parallel program.	Coloured Petri Nets and Token Petri Net
P10	To present a methodology using Petri nets for hierarchical and distributed control.	Petri Net Based Multitask Processing
P11	To present the methods for modeling discrete event robotic manufacturing systems using Petri nets.	Petri Net Based Multitask Processing
P12	To present a methodology of decomposition and coordination for hierarchical and distributed control.	Petri Net Based Multitask Processing
P13	To present a methodology of the net model decomposition and coordination for a hierarchical and distributed control.	Petri Net Based Multitask Processing

The main results presented in TABLE V are all related in some level to the presented definition of distributed process or, in essence to the parallelization of execution. It demonstrates the use of a great variety of Petri Nets in the most diverse areas (ex. airport simulation (P1), language definition (P2), manufacturing simulation (P10) etc.) and some derivative works which use some of the established theories presented (P2, P11- P13).

TABLE V. MAIN RESULTS OF RESEARCHES

ID	Main result
P1	This work thus provides elements to model complex chronicles recognition, and to define a semantics of the chronicle language in terms of the used subset of coloured nets.
P2	Chronicles are represented by p-t-time Petri nets combining t-time Petri nets representing window admissibly constraints and p-time Petri nets representing interval constraints.
P3	An improved partitioning algorithm regarding the process-processor mapping, which reduces error rate and promote the performance of the parallel simulation was discussed.
P4	Transformation of CPN slices into abstract software components which will be used in further work on proposing a new modeling framework for distributed systems.
P5	The successful application of P/T Petri net process (PTP) and Property-preserving Petri net process algebras (PPPA) in the area of the bi-relational P/T Petri nets and the sequential object Petri nets that represent an interesting modification of conventional Petri nets and that can be applied at design, modeling, analysis and verification of generally distributed multithreading object-oriented programming systems.
P6	Demonstration that CAN based Networks using crossbar Switched fabric are still very robust when compared to the new sophisticated buses.

P7	A methodology was designed for Cognitive Radios which is entirely based on Petri Nets and which is a very efficient way to describe concurrent processes. Furthermore, an iterative frequency sensing method was described, which is a basic prerequisite for cognitive operation. The algorithm could be implemented in an all parallel version or sequential version.
P8	Important improvements in performance can be obtained with respect to static scheduler algorithms, using a dynamic scheduler based on Petri Nets, which is easy to implement. The model is adaptable to different numbers of processors and data block partitions.
P9	The parallel execution environment developed was able to reach a real utilization of the processors very close to its theoretical limit. Also, modeling an algorithm with Colored Petri Nets (CPN) allows analyzing its parallel capabilities and brings information about its possibilities and limitations in the search for a better parallel performance. Any parallel algorithm designed following a well formed CPN can be executed with a high level of performance by only changing the incidence matrix and tuning its virtual processors.
P10	The demonstrations show that the proposed system can be used as an effective tool for consistent modeling and control of large and complex manufacturing systems.
P11	The cooperation of each controller was implemented so that the behavior of the overall system was the same as the centralized control system and the task specification was completely satisfied.
P12	Modeling, simulation and control of large and complex manufacturing systems can be performed consistently using Petri nets.
P13	The overall control structure of a two-level robotic system was implemented using a high-speed communication network of PLCs with shared global information.

4 Conclusions

In this paper, a scoping study was initiated to collect and compare existing practical applications of Petri nets in distributed processing.

We analyzed several applications of Petri nets in distributed processing (this will be indexed according to TABLE II). Among the examples presented, there is a failure detection language (P2), which is further used to create a distributed simulation for the analysis of several airport configurations (P1). In (P3), it is possible to see the creation of a parallel or distributed simulation mechanism inspired in existing Petri Nets (TTPN, ETTPN) and their correlation by the use of Lookahead computation.

In (P4) it was presented the development results for the proposal of a new modeling framework for distributed systems. In (P5) a modification of PPPA Petri Nets is proposed in order to allow its execution on distributed or parallel environments. In P(6), we have an application in the CAN industrial bus, where timed colored Petri nets are used to model a switched fabric CAN network Architecture and a switched-fabric bus. A switched-fabric bus is unique in that it allows all CAN Controllers on a bus to logically interconnect with each other. Therefore, we can envision the parallel distribution of the messages in the bus and its impacts during the simulation.

In (P7) we can see an application of Petri nets in the dynamic modification of device drivers for cognitive radios and the possibility of parallelization of the algorithm, which may imply in distributed processing. The development of a parallel programming model starting with Colored Petri Nets (CPN), unfolding to a Token Petri Net (TPN) and executed by a set of distributed processors is presented in (P8).

In (P9) we have a derivative work that continues the research presented in (P8), by applying the same algorithm model to verify the real utilization of the processors in a parallel execution environment. In (P10), the authors create an extended definition of Petri net and use it as an effective tool for consistent modeling and control of large and complex manufacturing systems. This research resulted in several derivative works (P11- P13) that uses the extended Petri net defined at (P10) in a variety of manufacturing scenarios (modeling discrete event robotic manufacturing systems, constructing hierarchical and distributed control systems, applying it to the control structure of a two-level robotic system using a high-speed communication network of PLCs).

This review does establish a baseline, which we wish will be used by other researchers. It contributes to the initial steps involved in the execution of a larger research aimed at conducting a systematic literature review on practical applications of Petri nets in distributed processing. Distributed processing was chosen as a candidate technique for a future application in the simulation of distributed systems applied to the processing of medical databases. We expect that Petri nets will perform an invaluable role as the simulation method for the proposed architecture.

Acknowledgment

The authors would like to thank the government bodies DECIT/SCTIE/MS/CNPq for the invaluable funding supporting the development of the current work, in the form of resources obtained in response to the public call number 12/2013, process number 201410267000311.

5 References

[1] R. Zurawski and M. Zhou, Petri Nets and Industrial Applications: A Tutorial, IEEE Transactions on Industrial Electronics, 41 (6) (1994) 567-583.

[2] B. Yilmaz, Applications of Petri Nets, A Thesis Submitted to the Graduate School of Engineering and Sciences of Izmir Institute of Technology in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mathematics, 2008. 61 p.

[3] T. Murata, Petri Nets: Properties, Analysis and Applications, in Proceedings of the IEEE, 77 (4), pp. 541 – 580, 1989.

[4] S. Singh, G. Singh, V. L. Narasimhan, and H. S. Pabla, Petri net modelling and analysis of Mobile Communication Protocols UMTS, LTE, GPRS and MANET, in 2014 International Conference on Computer Communication and Informatics (ICCCI - 2014), p. 1-9, 2014.

[5] Y. Liu, J-J. Hao and Z-G. Fang, Petri net model for performance evaluation of manufacturing system under uncertain information, Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS, 20 (5), (2014), p. 1237-1245.

[6] R. A. Siqueira and J. A. Jardini, "Using Petri nets to model and analysis the start and stop sequence control of an electric power generating unit", in 2013 IEEE PES Conference on Innovative Smart Grid Technologies, p. 1-8, 2013.

[7] D. Karunakaran and G. S. V. Rao, "A petri net simulation of software development lifecycle towards green IT", in: 2013 IEEE Conference on Open Systems (ICOS 2013), p 58-62, 2013.

[8] M. Westergaard, "Verifying Parallel Algorithms and Programs Using Coloured Petri Nets", in: Transactions on Petri Nets and Other Models of Concurrency VI Lecture Notes in Computer Science Vol. 7400, 2012, pp 146-168.

[9] G. Yasuda, Implementation of real-time distributed control for discrete event robotic systems using Petri nets, Artificial Life and Robotics, 16 (4), (2012), pp. 537-541.

[10] G. Coulouris, J. Dollimore and T. Kindberg (2007). Distributed Systems: Concepts and Design. Ed. Bookman, 4ed. ISBN 9788560031498. pp. 207.

[11] K. Davis, N. Drey and D. Gould, What are scoping studies? A review of the nursing literature, Int J Nurs Stud, 46 (2009), p. 1386-1400.

[12] S. M. A. Shah, M. Morisio and M. Torchiano, "An overview of software defect density: a scoping study", in 19th Asia-Pacific Software Engineering Conference, pp. 406-415, 2012.

[13] H. Abukwaik, D. Taibi and D. Rombach, Interoperability-related architectural problems and solutions in Information Systems: a scoping study, Software Architecture, Lecture Notes in Computer Science, vol. 8627, 2014, pp. 308-323.

[14] V. H. S. Durelli et al., A scoping study on the 25 year of research into software testing in Brazil and an outlook on the future of the area, The Journal of Systems and Software, vol. 86, 2013, 934-950.

[15] S. S. Askool, K. Nakata, Scoping study to identify factors influencing the acceptance of social CRM, in Proceedings of the 2010 IEEE ICMIT, pp. 1055 – 1060, 2010.

[16] A. Meredith and Z. Hussain, Online gaming: a scoping study of massively multi-player online role playing games, Electronic Commerce Research, 9 (1-2) (2009) pp. 3-26.

[17] D. Levac, H. Colquhoun and K. K. O'Brien, Scoping studies: advancing the methodology, Implementation Science, 5 (69) (2010) 1-9.

[18] H. Arksey and L. O'Malley, Scoping studies: towards a methodological framework, International Journal of Social Research Methodology, 8 (1) (2005) 19-32.

[19] B. Kitchenham, Procedures for Performing Systematic Reviews, Joint Technical Report, Department of Computer Science, Keele University (TR/SE-0401) and National ICT Australia Ltd. (0400011T.1), 2004.

[20] O. Dieste and A. G. Padua, "Developing Search Strategies for Detecting Relevant Experiments for Systematic Reviews", in 1st International Symposium on Empirical Software Engineering and Measurement, pp. 215-224, 2007.

[21] E. Engström, P. Runeson and M. Skoglund, A Systematic Review on Regression Test Selection Techniques, Information and Software Technology, 52 (1) (2010) 14-30, doi: 10.1016/j.infsof.2009.07.001

[22] T. Dybå, T. Dingsøyr and G. K. Hanssen, "Applying Systematic Reviews to Diverse Study Types: An Experience Report", in 1st International Symposium on Empirical Software Engineering and Measurement, pp. 225-234, 2007.

[23] R. Pawson, Evidence-based policy: in search of a method. Evaluation, 8 (2) (2002) 157-181.

[24] O. P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner and M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of Systems and Software, 80 (4) (2007) 571-583.

[25] B. A. Kitchenham et al., Systematic literature reviews in software engineering – A systematic literature review, Information and Software Technology, 51 (2009) 7-15.

[26] O. Bertrand, P. Carle and C. Choppy, "Modelling chronicle recognition for distributed simulation processing with coloured petri nets", in: 2nd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2009), p. 1-2, 2009.

[27] A. Boufaied, A. Subias and M. Combacau, "Chronicle modeling by Petri nets for distributed detection of process failures", in: Proceedings of the IEEE

International Conference on Systems, Man and Cybernetics, v. 4, p 245-250, 2002.

[28] X. Fang, Z. Xu and Z. Yin, "Distributed processing based on Timed Petri Nets", in: Proceedings - Third International Conference on Natural Computation (ICNC 2007), v. 5, p. 287-291, 2007.

[29] M. Iwaniak and W. Khadzhynov, Colored Petri Net Model of X/Open Distributed Transaction Processing Environment with Single Application Program, Communications in Computer and Information Science, v. 424, p. 20-29, 2014.

[30] I. Martinik, "Modelling of Distributed Programming Systems with Using of Property-Preserving Petri Net Process Algebras and P/T Petri Net Processes", in: 2nd International Conference on Informatics and Applications (ICIA 2013), p. 258-263, 2013.

[31] M. Mazouzi, I. B. Mbarek, O. Kallel and M. Abid, "Using SCPN for Modelling a Crossbar Switched Fabric CAN Network", in: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12), p. 981-987, 2012.

[32] Spiegel et al., A petri nets based design of cognitive radios using distributed signal processing, Procedia Earth and Planetary Science, 1 (1), p. 1474-1479, 2009.

[33] G.Wolfmann and A. Giusti, Parallel Asynchronous Modelization and Execution of Cholesky Algorithm using Petri Nets, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13), p. 52-58, 2013.

[34] G.Wolfmann and A. Giusti, Petri Net Based Algorithm Modelization and Parallel Execution on Symmetric Multiprocessors, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'14), p. 347 – 354, 2014.

[35] G.Yasuda, Hierarchical and Distributed Control of Robotic Manufacturing Processes Based on Petri Nets, in: Proceedings of the IEEE International Conference on Automation and Logistics (ICAL 2008), p 221-226, 2008.

[36] G.Yasuda, Distributed Control of Industrial Robot Systems using Petri Net Based Multitask Processing, in: Proceedingsof ICCAS-SICE 2009 - ICROS-SICE International Joint Conference 2009, p. 5633-5638, 2009.

[37] G.Yasuda, Distributed Cooperative Control of Industrial Robotic Systems Using Petri Net Based Multitask Processing, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), v 6425 LNAI, n PART 2, p 32-43, 2010.

[38] G.Yasuda, Modeling, Simulation and Distributed Control of Robotic Systems Using Petri Net Based Multitask Processing, in: Proceedings of the SICE Annual Conference (SICE 2011), p 1944-1949, 2011.

Real-Time Image Processing Applications on Multicore CPUs and GPGPU

R. Samet[1], **O.F. Bay**[2], **S. Aydın**[2], **S. Tural**[1], **A. Bayram**[1]
[1]Computer Engineering, Ankara University, Ankara, Turkey
[2]Education of Computer and Electronics, Gazi University, Ankara, Turkey

Abstract *- This paper presents real-time image processing applications using multicore and multiprocessing technologies. To this end, parallel image segmentation was performed on many images covering the entire surface of the same metallic and cylindrical moving objects. Experiments with multicore CPUs showed that by increasing the chunk size, the execution time decreases approximately four times in comparison with serial computing. The same experiments were implemented on GPGPU using four methods: 1) Single-Image Transmission with Single-Pixel Processing; 2) Single-Image Transmission with Multiple-Pixel Processing; 3) Multiple-Image Transmission with Single-Pixel Processing; 4) Multiple-Image Transmission with Multiple-Pixel Processing. All methods were implemented on GeForce and Tesla. Tesla gave the best results of 23 (for the first method), 20 (for the second method), 42 (for the third method), and 58 (for the fourth method) times improvements in comparison with serial computing.*

Keywords: Parallel computing, real-time image processing, image segmentation, thresholding, multicore programming, GPU programming

1 Introduction

On the one hand image processing requires long time. On the other hand time is usually limited in the real-time applications. So, serial image processing does not satisfy real time conditions. In order to solve this problem, parallel computing techniques, especially multicore and multiprocessing technologies, should be used.

Segmentation is one of the steps in image processing. Thresholding is widely used for this aim. In real-time applications, multicore CPUs and GPGPU should be used to execute thresholding on many images covering the whole surface of the same metallic and cylindrical moving object to satisfy real-time conditions.

A multicore CPU is a single computing component with two or more independent actual central processing units (called "cores"). Pthreads, OpenMP (Open Multi-Processing), TBB (Threading Building Blocks), and Cilk are application programming interfaces (API) to efficiently use the capacity of a multicore CPU. In this study, a general purpose and platform-independent OpenMP that supports shared-memory for multi-processing programming in C, C++, and FORTRAN will be used.

A graphic processing unit (GPU) is a single instruction and multiple data stream (SIMD) architecture where the same instruction is performed on all data elements in parallel. At the same time, the pixels of an image can be considered as separate data elements. So, GPU is a suitable architecture to process data elements of an image in parallel [1]. General-purpose computing on graphics processing units (GPGPU) is a tool to increase the utilization of GPU. There are many platforms to efficiently use the capacity of GPGPU, such as CUDA, DirectCompute, and OpenCL. The CUDA platform, which is the most common one, will be used in this study [2].

Multicore CPU and GPGPU technologies are widely used for non-real-time and real-time image processing applications. A short literature review related to these technologies is given below.

There are many studies reported in the literature related to non-real-time image segmentation using the threshold method [3, 4]. The performance was and still remains an urgent issue to be solved in real-time image processing applications. To this end, different algorithms and methods have been developed for serial computing [5, 6, 7]. Despite some performance improvements in these works, it is very difficult to satisfy real-time conditions by serial computing. Researchers have looked into alternative solutions and found the multicore CPU and GPGPU technologies to solve this issue. At the same time, in order to efficiently use these technologies, different platforms, such as OpenMP, and CUDA, have been developed and widely used. For example, OpenMP programming has been used in multithread image processing and image segmentation applications with multicore computing [8, 9]. CUDA programming has been used for parallel image segmentation by region growing, watershed, and Otsu binarization algorithms on GPU [10, 11, 12, 13]. The reduction sweep algorithm was used for image segmentation on both CPU and GPU [14]. In [15], several methods for image segmentation were implemented using CUDA and GPU and processing time was accelerated about 20 times. The authors of [16] present the results of image segmentation on a video with a frame rate of 30 Hz using CUDA and GPU.

This review shows that more efficient algorithms and methods still need to be developed to improve the performance of real-time image processing applications. One of the aims of this study was to make a contribution to this area using OpenMP and CUDA. To this end, bi-level

thresholding was implemented on the images covering the entire surface of the same metallic and cylindrical moving object in parallel with the following methods. One method is related to CPU programming with the OpenMP platform. In this context, shared memory multicore programming with OpenMP, scheduling threads on cores with different parameters, and performance related to the execution time were analyzed. The other four methods are related to GPU programming with the CUDA platform: 1) Single-Image Transmission with Single-Pixel Processing (SISP) (Namely, images are transmitted from CPU to GPU one by one and the pixels of the images are processed one pixel per core of GPU); 2) Single-Image Transmission with Multiple-Pixel Processing (SIMP) (Namely, images are transmitted from CPU to GPU one by one and the pixels of the images are processed multi pixels per core of GPU); 3) Multiple-Image Transmission with Single-Pixel Processing (MISP) (Namely, multiple images are transmitted from CPU to GPU together and the pixels of the images are processed one pixel per core of GPU); 4) Multiple-Image Transmission with Multiple-Pixel Processing (MIMP) (Namely, multiple images are transmitted from CPU to GPU together and the pixels of the images are processed multi pixels per core of GPU). Performance analysis related to execution time was performed by comparison of the results obtained by these methods with serial computing. The method with multicore CPU showed that, by increasing the chunk size, the execution time decreases approximately four times. All methods with GPU were implemented on GeForce and Tesla. Tesla gave best results of 23 (for SISP method), 20 (for SIMP method), 42 (for MISP method), and 58 (for MIMP method) times improvements in comparison with serial computing.

2 Proposed real-time image processing methods

Real-time applications of this study are related to the inspection of certain defects on the entire surface of metallic and cylindrical objects moving at a rate of 5 units per second or 1 unit per 200 milliseconds. Data sets or images used in this study are the images taken from the entire surface of the same metallic and cylindrical moving object. Images were used to inspect the defects in real-time. Defects are detected by image processing techniques. In order to detect certain defects of a single object, the image processing steps should be processed on K images covering its entire surface during 200 milliseconds. So, time is limited in given applications. In this study, only the first step of image processing related to image segmentation will be handled. Thresholding is the simplest and a fast way for image segmentation. Parallel programming techniques, such as multicore and multiprocessing technologies, were used to speed up the thresholding of the metallic and cylindrical object from images covering its entire surface.

Firstly, serial thresholding is described. Then, parallel thresholding on a multicore CPU with OpenMP is presented. Finally, parallel thresholding on GPU with CUDA is discussed. Four different algorithms and methods related to CUDA are proposed.

2.1 Serial thresholding

Image segmentation is the process of dividing the individual elements of an image into a set of groups so that all elements in a group have a common property. Segmentation allows visualization of the structures of interest, removing unnecessary information [17]. Thresholding is the simplest, most commonly used and the most popular technique for segmentation. Thresholding techniques can be classified into two categories: bi-level and multilevel. In this study, bi-level segmentation is used for the segmentation of objects and the background [4]. Thresholding is often used as a preprocessing step, followed by other post-processing methods [18]. Let us denote by $g(x,y)$ the segmented image obtained from $f(x,y)$. If we consider T as the threshold value, the resulting image will be given by

$$g(x,y) = \begin{cases} 255, & if\ f(x,y) \geq T \\ 0, & if\ f(x,y) < T \end{cases} \quad (1)$$

According to serial thresholding, equation (1) should be calculated on each pixel of (x,y) of an original image of $f(x,y)$, where $x = 1,2,\dots,N$ and $y = 1,2,\dots,M$. The performance or processing time of serial thresholding is defined as following:

$$t_{ST} = N * M * \Delta t \quad (2)$$

where t_{ST} is the processing time of serial thresholding and Δt is the processing time for thresholding on one pixel.

2.2 Parallel thresholding on a multicore CPU with OpenMP

In order to accelerate the thresholding process to satisfy the real-time conditions, the shared-memory multicore programming with OpenMP is proposed. An OpenMP platform always begins with a single thread of control, called the master thread, which exists during the run-time of the program. The master thread may encounter parallel regions, in which the master thread will fork the new threads, each with its own stack and execution context. At the end of the parallel region, the forked threads will be terminated and the master thread continues the program execution as shown in Fig.1.

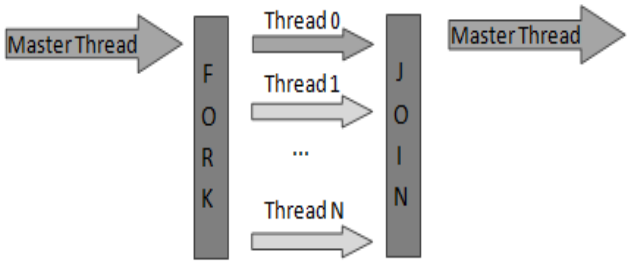


Fig.1. Thread organization with OpenMP

To achieve the optimal performance in multithread applications, different scheduling types and chunk sizes should be tested. With OpenMP, static, dynamic, and guided scheduling mechanisms can be specified. Static scheduling divides the loop into equal-sized chunks or as equal as possible in the case where the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size. Dynamic scheduling uses the internal work queue to give a chunk sized block of loop iterations to each thread. When a thread is finished, it retrieves the next block of loop iterations from the top of the work queue. By default, the chunk size for dynamic scheduling is 1. Guided is similar to dynamic scheduling, but the chunk size starts off large and decreases to better handle the load imbalance between iterations. The optional chunk parameter specifies the minimum chunk size to use. By default, the chunk size for guided scheduling is approximately calculated by:

$$ChunkSize = \frac{N_L}{N_T} \quad (3)$$

where N_L is a number of loop count and N_T is a number of threads. The performance or processing time of parallel thresholding with OpenMP is defined as follows:

$$t_{MP} = \frac{t_{ST}}{N_T} + t_O = \frac{N*M*\Delta t}{N_T} + t_O \quad (4)$$

where $t_{MP}$is a processing time of parallel thresholding with OpenMP and t_O is a processing time for fork and join of threads.

2.3 Parallel thresholding on a GPU with CUDA

In order to accelerate the thresholding process to satisfy the real-time conditions, GPU programming with CUDA is proposed. The CUDA programming model consists of functions, called kernels, which can be executed simultaneously by a large number of threads on the GPU. Threads are grouped into warps. A warp consists of 32 threads which are executed in SIMD fashion independently. Threads within a warp execute the same instruction on different data elements in parallel [19].

In order to parallelize the thresholding process, the kernel should be used. To organize kernels to work in parallel, streams are used (Fig.2).

```
1   cudaStream_t stream1, stream2, streamK;
2   cudaStreamCreate (&stream1);
…
3   cudaStreamCreate (&streamK);
4   cudaMemcpyAsync (dst, src, size, dir, stream1);
5   kernel<<<grid, block, 0, stream1>>>(…);
…
6   cudaMemcpyAsync (dst, src, size, dir, streamS);
7   kernel<<<grid, block, 0, streamK>>>(…);
8   cudaMemcpyAsync (dst, src, size, dir, stream1);
…
9   cudaMemcpyAsync (dst, src, size, dir, streamK).
```

Fig.2. Multi kernels organization by streams

Firstly, the K streams are defined (Line 1) and created (Lines 2, 3). K is a number of images. Then data (images) for created streams are transferred asynchronously from the CPU to the GPU (Lines 4, 6). After that, kernels execute the same instructions on K images asynchronously (Lines 5, 7). Finally, the results are transferred from the GPU to the CPU (Lines 8, 9).

Images can be sent from the CPU to the GPU one by one or in a combined data array. Images can be processed in the cores of the GPU as one pixel by one pixel or in multi pixels. Results can be returned from the GPU to the CPU one by one, or in a combined data array.

The algorithm for sending K images from the CPU to the GPU one by one, processing them in GPU and returning the results from the GPU to the CPU one by one is given in Fig.3.

Step 1: Send the 1st image from CPU to GPU;
Step 2: Execute the thresholding kernel on the 1st image;
Step 3: Send the 2nd image from CPU to GPU;
Step 4: Execute the thresholding kernel on the2nd image;
…
Step$2K-1$: Send the Kth image from CPU to GPU;
Step$2K$: Execute the thresholding kernel on the Kth image;
Step$2K+1$: Return the 1st result from GPU to CPU;
...
Step$2K+K$: Return the Kth result from GPU to CPU.

Fig.3. Algorithm for single-image transmission

The algorithm for sending Kimages from the CPU to the GPU in a combined data array, processing them in the GPU and returning the results from the GPU to the CPU in a combined data array is given in Fig.4.

Step 1: Combine K images in a data array;
Step 2: Send the combined data array from CPU to GPU;
Step 3: Execute the thresholding kernel on K images;
Step 4: Return the combined results from GPU to CPU;
Step 5: Separate the images from combined results.

Fig.4. The algorithm for multiple-image transmission

The algorithm for distributing and processing the images as one pixel per core of the GPU is given in Fig.5.

Step 1: Distribute pixels as one pixel per core of GPU;
Step 2: If pixel value >= T then the result is 255;
Step 3: If pixel value <T then the result is 0;
Step 4: Repeat Step 2 and Step 3 for all pixels.

Fig.5. The algorithm for single-pixel processing in the GPU

The algorithm for distributing and processing the images as P pixels per core of the GPU is given in Fig.6.

Step1: Distribute pixels as P pixels per core of GPU;
Step2: Take the ith pixel value;
Step 3: If pixel value >= Tthen the result is 255;
Step 4: If pixel value <Tthen the result is 0;
Step 5: Repeat Step 2, 3, 4 while i<= P;
Step 6: Repeat all steps for all pixels.

Fig.6. The algorithm for multi-pixel processing in the GPU

Four methods are proposed to execute thresholding on the GPU with CUDA: 1) SISP; 2) SIMP; 3) MISP, and 4) MIMP (Table 1).

Table 1. Proposed methods

Methods	Images and Results Transferring Algorithms	Image Distributing and Processing Algorithms
SISP	Fig.3	Fig.5
SIMP	Fig.3	Fig.6
MISP	Fig.4	Fig.5
MIMP	Fig.4	Fig.6

2.3.1 SISP method

In this method, the images are transmitted from the CPU to the GPU one by one and results are returned from the GPU to the CPU one by one using the proposed algorithm in Fig.3. Also, the pixels of the images are distributed and processed one pixel per core of the GPU using the proposed algorithm in Fig.5.

2.3.2 SIMP method

In this method, the images are transmitted from the CPU to the GPU one by one and the results are returned from the GPU to the CPU one by one using the proposed algorithm in Fig.3. Also, the pixels of the images are distributed and processed as multi pixels per GPU core using the proposed algorithm in Fig.6.

2.3.3 MISP method

In this method, the images are transmitted from the CPU to the GPU in a combined data array and the results are returned from the GPU to the CPU in a combined data array using the proposed algorithm in Fig.4. Also, the pixels of the images are distributed and processed one pixel per core of the GPU using the proposed algorithm in Fig.5.

2.3.4 MIMP method

In this method, the images are transmitted from the CPU to the GPU in a combined data array and the results are returned from the GPU to the CPU in a combined data array using the proposed algorithm in Fig.4. Also, the pixels of the images are distributed and processed as multi pixels per GPU core using the proposed algorithm in Fig.6.

3 Experiment results

Experiments were related to the real-time detection of standard defects on the surface of the military cases, such as scratches, dents, wrinkles, and crimps (Fig.7).

Eight images covering the entire 360-degree (8x45 degrees) surface of the same moving military cases were used to detect the defects (Fig.8).

A multicore CPU with OpenMP and GPGPU with CUDA were used to implement the parallel segmentation through the thresholding of the military cases and background.

Speed up value was used to evaluate segmentation techniques:

$$SpeedUp = t_{ST} / t_{PT} \quad (5)$$

where t_{PT} is the processing time of parallel thresholding.

Fig.7. 7.62 mm Military cases

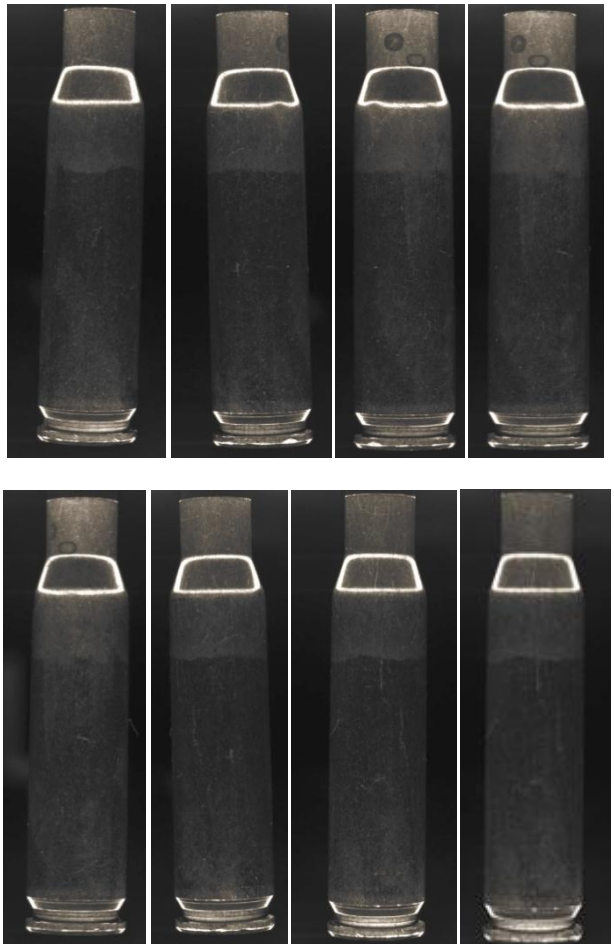


Fig.8. Images covering the entire 360-degree (8x45 degrees) surface of the same military case

The following platform was used: Intel Core i7-3630QM CPU with 4 cores and hyper threading technologies; 8 GB RAM; Windows 7. The codes were written in C++ using the Visual Studio 2012. Images with different resolutions (320x240, 640x480, and 1280x960) were used.

3.1 Parallel thresholding on a multicore CPU with OpenMP

Static, dynamic, and guided scheduling types with different chunk sizes were implemented to speed up the segmentation process (Table 2).

Table 2. Experiment results on a multicore CPU with OpenMP

Chunk Size	Speed up with		
	Static Scheduling	Dynamic Scheduling	Guided Scheduling
1	3,42	4,03	4,08
2	3,30	4,00	4,1
4	3,39	4,12	4,02
6	3,44	3,95	4,01
10	3,47	3,86	4,06
15	3,56	3,93	3,64
30	3,41	3,81	3,66
60	3,42	3,50	3,58
120	3,44	3,30	3,39
240	2,86	2,79	3,04
480	1,77	1,80	1,75

Table 2 presents the experiment results of the speed up of different scheduling types with different chunk sizes. As seen, the dynamic and guided scheduling types gave the best results. By increasing the chunk size, the speed up is decreased for all scheduling types. In summary, in order to obtain the best results by OpenMP, chunk sizes should be as small as possible and dynamic or guided scheduling types should be used.

3.2 Parallel thresholding on a GPU with CUDA

NVIDIA GeForce GT 635M with 96 cores and Tesla K20 with 2496 cores were used. The number of thread size was set to 1024. Four methods were implemented: 1) SISP; 2) SIMP; 3) MISP, and 4) MIMP.

3.2.1 SISP

In this method, eight images were sent and executed one by one. The pixels of the images were distributed as one pixel (or 8 bits) per GPU core (Table 3).

As seen, Tesla gave the best result of 23 times improvement in comparison with serial computing. Another point with Tesla was that by increasing the image resolution, speed up rate decreased. GeForce gave 10 times improvement, which is less than Tesla. This is due to the fewer number cores (96) in comparison with Tesla, which has 2496 cores. Also the speed up rate in GeForce was approximately the same for the different image resolutions. In summary, in order to obtain the best results by the SISP method, Tesla should be used and image resolution should be as small as possible.

Table 3. SISP results

GPU Type	Image Resolution	Speed up Rate on Kernel
GeForce GT 635M	320x240	8,03
	640x480	10,19
	1280x960	8,92
Tesla K20	320x240	23,02
	640x480	18,27
	1280x960	8,65

3.2.2 SIMP

In this method, eight images were sent and executed one by one. The pixels of the images were distributed as four pixels (or 32 bits) per GPU core (Table 4).

Table 4.SIMP results

GPU type	Image Resolution	Speed up Rate on Kernel
GeForce GT 635M	320x240	8,25
	640x480	9,25
	1280x960	9
Tesla K20	320x240	20.14
	640x480	20,41
	1280x960	10

As seen, Tesla gave the best result of 20 times improvement in comparison with serial computing. Another point with Tesla was that, for high image resolution, the speed up rate was decreased. GeForce gave 9 times improvement, which was less than Tesla. This was due to the fewer number cores (96) in comparison with Tesla, which has 2496 cores. Also, the speed up rate for GeForce was approximately the same for different image resolutions. In summary, in order to obtain the best results by the SIMP method, Tesla should be used and image resolution should be as small as possible.

3.2.3 MISP

In this method eight images were combined in a data array. This data array was sent and executed in a kernel. The pixels of the images were distributed as one pixel (or 8 bits) per GPU core (Table 5).

Table 5. MISP results

GPU type	Image Resolution	Serial Computing Time (ms)	Kernel Time (ms)	Speed up Rate on Kernel
GeForce GT 635M	320x240	5,01	0,88	5,98
	640x480	9,34	2,68	3,36
	1280x960	26,92	10.18	2,54
Tesla K20	320x240	8,45	0,44	18,92
	640x480	15,23	0,54	27,78
	1280x960	43,99	1,04	42,02

As seen, Tesla gave the best result of 42 times improvement in comparison with serial computing. Another point with Tesla was that, as the image resolution increased, speed up rate increased. GeForce gave 6 times improvement, which was less than Tesla. This is due to the fewer number cores (96) in comparison with Tesla, which has 2496 cores. Also, the speed up rate of GeForce was decreased by increasing the image resolutions. In summary, in order to obtain the best results by the MISP method, Tesla should be used and image resolution should be as large as possible.

3.2.4 MIMP

In this method, eight images were combined in a data array. This data were sent and executed in a kernel. The pixels of the images were distributed as four pixels (or 32 bits) per GPU core (Table 6).

Table 6. MIMP results

GPU type	Image Resolution	Serial Computing Time (ms)	Kernel Time (ms)	Speed up Rate on Kernel
GeForce GT 635M	320x240	5,01	0,69	7,26
	640x480	9,34	1,85	5,04
	1280x960	26,92	6,91	3,8
Tesla K20	320x240	8,38	0,43	19,09
	640x480	15,28	0,45	33,71
	1280x960	44,32	0,75	58,96

As seen, Tesla gave the best result of 58 times improvement in comparison with serial computing. Another point with Tesla was that, as the image resolution increased, the speed up rate increased. GeForce gave 7 times improvement, which is less than Tesla. This is due to the fewer number cores (96) in comparison with Tesla, which has 2496 cores. Another point with GeForce was that the speed up rate decreased by increasing image resolutions. In summary, in order to obtain the best results by the MIMP method, Tesla should be used and the image resolution should be as large as possible.

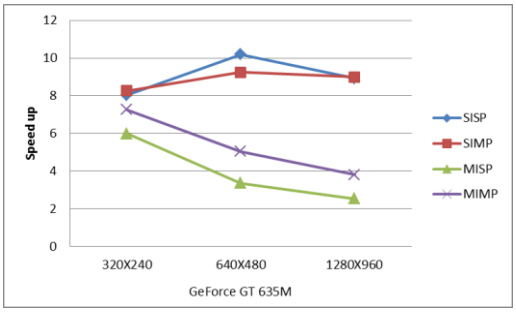
The comparison results of the proposed methods with CUDA in terms of speed up are given in Table 7 and Fig.9.

Table 7. Comparison results of the proposed methods

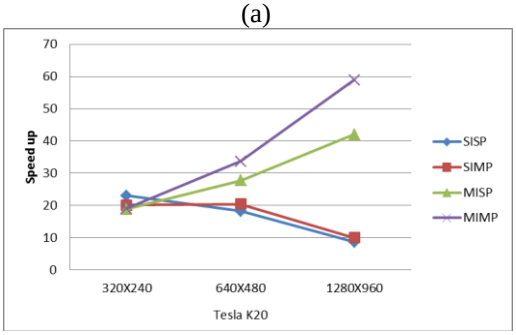
GPU type	Image Resolution	Speed up on Kernel by			
		SISP	SIMP	MISP	MIMP
GeForce GT 635M	320x240	8,03	8,25	5,98	7,26
	640x480	10,19	9,25	3,36	5,04
	1280x960	8,92	9	2,54	3,8
Tesla K20	320x240	23,02	20.14	18,92	19,09
	640x480	18,27	20,41	27,78	33,71
	1280x960	8,65	10	42,02	58,96

As seen, Tesla gave the best results for all methods. With Tesla, the speed up rates for MISP and MIMP methods were higher than those of the SISP and SIMP ones. Another point with Tesla was that, by increasing the image resolution, the speed up rate increased. In summary, in order to obtain the best results with CUDA, MISP and MIMP methods should be used.

An example for segmentation results with parallel thresholding is given in Fig.10.



(a)



(b)

Fig.9. Comparison results of the proposed methods: (a) with GeForce GT 635M; (b) with Tesla K20

(a)

(b)

Fig.10. Segmentation result: (a) the original image; (b) segmentation result by parallel thresholding.

4 Conclusions

This paper presented the image processing applications using multicore and multiprocessing technologies to satisfy real time conditions. To this end, algorithms and methods for the parallel image segmentation through thresholding on K images covering the entire surface of the same metallic and cylindrical moving objects were proposed. A multicore CPU with OpenMP and GPGPU with CUDA were used to implement the thresholding of military cases using eight real images covering their entire surface. Obtained implementation results were compared with the results of serial computing in terms of speed up values. Experiments showed that a GPU with CUDA has a huge capacity to increase the performance of real-time applications. For example, CUDA speeded up the real-time thresholding process 58 times in comparison with serial computing.

As future work, the time to transfer images from the CPU to the GPU and results from the GPU to the CPU will be analyzed and optimized. Another future work is that the proposed algorithms and methods will be implemented on a different multicore CPU and GPU.

Acknowledgement

This work was funded by the Ministry of Science, Industry and Technology of Turkey under San-Tez grant #0018.STZ.2013-1.

5 References

[1] E. Smistad, A.C. Elster, and F. Lindseth, "GPU accelerated segmentation and centerline extraction of tubular structures from medical images", Int J CARS, 9, 561–575, 2014.

[2] A. Brodtkorb, T.R. Hagen, and M.L. Saetra, "Graphics processing unit (GPU) programming strategies and trends in GPU computing", J. Parallel Distrib.Comput., 73, 4–13, 2013.

[3] S.S. Al-amri, N.V. Kalyankar, and S.D. Kamitkar, "Image Segmentation by Using Threshold Techniques", Journal of Computing, 2, 5, 83-86, 2010.

[4] V. Osuna-Enciso, E. Cuevas, and H. Sossa,"A comparison of nature inspired algorithms for multi-threshold image segmentation", Expert Systems with Applications, 40,1213–1219, 2013.

[5] S. Wei., Q. Hong, and M. Hou, "Automatic image segmentation based on PCNN with adaptive threshold time constant", Neurocomputing, 74, 1485–1491, 2011.

[6] S. Han, W. Tao, X. Wu, X. Tai, and T. Wang, "Fast image segmentation based on multilevel banded closed-form method", Pattern Recognition Letters, 31, 216–225, 2010.

[7] H.V.H. Ayala, F.M. Santos, and V.C. Mariani, "Image thresholding segmentation based on a novel beta differential evolution approach", Expert Systems with Applications, 42, 2136–2142, 2015.

[8] R. Wang, C. Li, J. Wang, X. Wei, Y. Li, Y. Zhu, and S. Zhang, "Threshold segmentation algorithm for automatic extraction of cerebral vessels from brain magnetic resonance angiography images, Journal of Neuroscience Methods, 241, 30–36, 2015.

[9] S. Patil, and A. Junnarkar, "Color Image Segmentation using Median Cut and Contourlet Transform:A Parallel Segmentation Approach", International Journal of Computer Science and Information Technologies (IJCSIT), 5, 6, 7353-7358, 2014.

[10] P.N. Happ, R.Q. Feitosa, C. Bentes, and R. Farias, "A parallel image segmentation algorithm on GPUs", Proceedings of the 4th GEOBIA, 580, 2012

[11] E. Smistad, A.C. Elster, and F. Lindseth, "GPU accelerated segmentation and centerline extraction of tubular structures from medical images", International journal of computer assisted radiology and surgery, 9, 561-575, 2014.

[12] A. Körbes, G.B. Vitor, R.A. Lotufo, and J.V. Ferreira, "Analysis of a step-based watershed algorithm using CUDA", International Journal of Current Research and Review, 1, 6-28, 2010.

[13] B.M. Singh, R. Sharma, A., Mittal, and D. Ghosh, "Parallel implementation of Otsu's binarization approach on GPU, International Journal of Computer Ap., 32, 16-21, 2011.

[14] R. Farias, R. Farias, R. Marroquim, and E. Clua, "Parallel Image Segmentation Using Reduction-Sweeps On Multicore Processors and GPUs", XXVI Conference on Graphics, Patterns and Images, 139-146, 2013.

[15] N. Prosser, "Medical image segmentation using GPU accelerated variational level set methods", Rochester Institute of Technology, Rochester, New York, 2010.

[16] A. Abramov, T. Kulvicius, F. Wörgötter, and B. Dellen, "Real-time image segmentation on a GPU", Facing the Multicore-Challenge Lecture Notes in Computer Science, 6310, 131-142, 2010.

[17] E. Smistad, T.L. Falch, M. Bozorgi, A.C. Elster, and F. Lindseth, "Medical image segmentation on GPUs– A comprehensive review", Medical Image Analysis, 20, 1–18, 2015.

[18] Y. Li, L. Jiao, R. Shang, and R. Stolkin, "Dynamic-context cooperative quantum-behaved particles warm optimization based on multi-level thresholding applied to medical image segmentation", Information Sciences, 294, 408–422, 2015.

[19] Z. Chen, X. Meng, L. Guo, and G. Liu, "GICUDA: A parallel program for 3D correlation imaging of large scale gravity and gravity gradiometry data on graphics processing units with CUDA", Computers&Geosciences, 46, 119–128, 2012.

The Path To Exascale Computing

R. Alshahrani
Department of Computer Science, Kent State University, Kent, OH, USA
Ministry of Higher Education, Riyadh, Saudi Arabia

Abstract— *Exascale supercomputers are the future of Cluster computing. In this paper we discuss the challenges of developing exascale supercomputers and provide suggestions on how to deliver the required performance from these new machines. The major point is that the current programming systems over valued the flops and ignore the data locality and data movement which becomes increasingly important. A cheaper innovative technologies are needed to improve the memory bandwidth and density for a better data management.*

Keywords: HPC, Exascale

1. Introduction

Advances in science led the scientists to face mountains of data that needs to be computed and stored. On the other hand, the notion of supercomputing and High Performance Computers (HPC) allows for more scientific breakthroughs. For example, Petascale supercomputers are able to reach performance of one petaflops which is used for advanced computation in diverse fields such as quantum chemistry, brain and weather simulations [1]. According to Moor's law, the data is doubling from year to year which urge the need for more advanced supercomputers with higher speed and advanced capabilities. Delivering a system with exaFLOP capability is significant for more scientific discoveries in different areas. Exascale computing is the next generation of supercomputing. It will be capable of performing at least one exaFLOPS which means 10^{18} operations per second, a thousandfold increase over its counterpart petascale supercomputer [2].

The advancement from petascale to exascale computing is not easy. Many companies are competing to present the first supercomputer with exaFLOPs capability. However, the development of such a powerful system is constrained by many factors such as power, memory and cost. Based on the improvement chart for 20 years, the Top500 expected the first exascale supercomputer to see the light not before 2020 and maybe zetascale supercomputer by 2029 [3].

Developing such a powerful system is possible but with some difficult challenges. Apparently, the current technologies are limiting the developers of exascale supercomputers, hence, new innovative technologies are needed to come up with the first exascale supercomputer. The leading design constraint is the power efficiency as discussed in section 3. Improving the performance while lowering down the energy consumption is a challenging task. The processor can consume upto 30% .Add to that the data movement which became a significant source of power consumption.

The challenges involve frequency improvement in future machines [4]. In the old days, clock frequency used to be the main constraints of performance improvement. However, with parallelism that has became not a big issue. the only way to increase performance is to increase parallelism. Parallelism is growing by an exponential rate within a chip. The next generation with supercomputing will keep on the same track of parallelism which is discussed in more details in section 2. In addition to the previous challenges, the memory constrains the performance of the future machines. The performance of the CPU is limited by the memory speed. therefore, adding more cores without improving the memory will not improve the performance. The memory technology improvements are slowing down [3], [4]. For example, DRAM technology have not changed for the last two decades [4].

More innovative solutions are needed to produce the next generation of supercomputing. The cost of data movement within the supercomputers became dominant since the cost of data movement is exceeding the cost of performing a floating point operation [5], [2]. The overall system should be optimized to decrease the data movement costs. Additionally, the energy cost for moving data is not improving in terms of the cost of the flop. Therefore, getting the applications more aware of the data locality will lower the cost of the data movement [5].

Since the HPC are extensively used in scientific communities, the characteristics of the scientific computations must drive the fundamentals of the exascale computing design [6]. The benefit of building such a powerful system is not limited to the scientific computing. Building effective exascale systems allowed for further advances such as in cellphone performance and voice recognition. The new technologies that are needed to develop exascale supercomputers will open the doors for new innovations. In order to do what we want do, the fundamental architecture should be different because it will be influenced by the workload and the power requirements of HPC. The power consumption should not exceed 20MW and the cost should be limited by 200M$ [7] to make it available for the users.

2. Architecture of Exascale Computing

The development of high-end systems such as supercomputing or High Performance Computing (HPC) that involves from millions to several million processors is highly challenging. It could have upto million processor cores, billions of threads, memory on the order of multiple petabyte. Increasing the clock rate by adding more transistors was the first approach to improve the performance of a computer. In high-end systems, the parallelism is the dominant factor to improve the performance. The parallelism within a chip has been increasing exponentially in two dimensional design. To fulfill the needs for the future computers, the parallelism is going to take another direction with three dimensional design[4]. Exascale supercomputers will still run x86 code like the current processors to ensure the compatibility with the current applications [3], [5].

Paying attention to a specific component of the system such as improving the performance in terms of FLOPs is not enough. Looking at the big picture of the system from different angles and how those components are interacting with each others is a key point. The overall performance in terms of FLOPS, memory speed, power consumption, data movement are all dependent. In other word, the processing speed is limited by the memory speed. To deliver this supercomputer to the market, a complete revolution is needed. In the following section, we provide an overview of the challenges to build the main architectural elements of the exascale supercomputers and what have been done so far.

2.1 CPU

Back in 80s, supercomputers used to be designed with specialized, custom-built processors which are very expensive. In early 90s, the research community has shifted into evolving more commodity components in the supercomputers with a better cost-to-performance ratio. In 2008, IBM revealed the Roadrunner machine which was the first supercomputer with hybrid processing scheme [8]. That brings the trend back to the specialized components as co-processors. The idea of hybrid processing was first invented for Sony PlayStation gaming console in 2008. The design of hybrid processors involved processing elements and specialized co-processing elements. Involving the specialized co-processing elements has improved the performance significantly. Other hybrid processing systems were built with (GPUs) as co-processors. Recently, the design of supercomputers with hybrid processing system became dominant.

2.2 Memory

To meet the performance requirement of exascale computing, the memory bandwidth have to increase, which in turns increases the power consumption. The more capabilities we put in the system the more the stress on the memory bandwidth. In order to meet the bandwidth needs stacking up more chips is not efficient due to data movement and power consumption. Additionally, increasing pins count for the sake of increasing the memory bandwidth will increase the cost without any performance improvement. That is because of the gap between CPU performance and the memory performance. Adding more cores to the system does not improve the memory bandwidth because adding pins is expensive. Nowadays, it is a matter of what we do about the power and the cost.

For the last 30 years, the memory technologies have not been changed. It is getting faster but basically it has the same architecture. We have to rethink the memory technology to make it more efficient. There are many thoughts on how the memory subsystem would look like in exascale computing. More innovative packaging and IO solutions are needed such as 3D stacking. For example, stacking the memory on top on the CPU which can be done by using new materials that can absorb the heat generated by the processor [9]. That will reduce the IO power consumption significantly while increasing the bandwidth.

A new technology is non-volatile memory technology that are beyond nano technology [9], [10]. However, the problem with this new technology is the limited lifetime which contradicts the reliability of this kind of computing system [10], [6]. The non-volatile memory does not consume energy while reading. However, it takes more energy than DRAM to write a bit. Memory technology have not changed for a long period of time. The design of the memory is probably will change permanently [4].

3. Power Consumption and Cost

Minimizing the power consumption while maximizing the performance is a key issue. The power consumption for the current high-end systems does not exceed 10MW. The future high-end supercomputers are expected to consume upto 20MW which is double the current consumption rate [4]. That would raise the cost of these super machines which contradict the design goals and requirements. A first step to find a solution for this problem is to figure out the sources of power consumption. In the current systems, the processor can consume upto 30% and the data movement became increasingly a source of high power consumption. Specifically, we should identify which part in the processor is consuming more power. Series of measurements and simulations have been done by Brooks and his colleagues to test the performance of the current processors and examine alternative designs [8]. Finding a balance between the performance and the clock frequency rate is complicated.

The memory is another resource of power consumption. Moving the data either horizontally or vertically has its own cost that could be higher than the cost of a FLOP [11]. All these factors combined urges the need for designing a new generation of chips and rethink the algorithms to compute efficiently. High power consumption will generate more heat,

hence, cooling will be needed. Finding the roots of the problem is a key point to develop a supercomputer with more efficient power management and control. A significant amount of research is needed to overcome this problematic issue.

4. Data Movement in Exascale Computing

While the FLOP used to be the most expensive component in the performance of HPC [7], the cost of data movement in a copper wire is not trivial and as important. The data movement could be vertical or horizontal. Vertical Data movement cost is the cost of moving the data from the memory into the processor and moving it back to the memory. Horizontal data movement includes moving the data between the interconnected [4]. For the exascale computing, the cost of the data movement could be more than the cost of a FLOP. Data movement management became increasingly important due to the cost in terms of power consumption and latency. That means, in average, every time we move a bit from the internal caches into the core we consume 10P of energy which is considerably high. For the future exascale computing, the cost could be even higher and it would cost upto 20P of energy. The consequence is that the cost for data movement costs more than the cost of a FLOP [11].

The increased cost of data movement is ignored in the current data programming systems. The current programming systems such as OpenMP values the flops and ignores the cost of data movement assuming it is free which is not the case [**?**], [9]. Developing applications that are more data locality aware could decrease the data movement significantly. For example, the old model of OpenMP describes how to parallelize loop iterations evenly among processors while ignoring where is the data located. A new programming system and algorithmic models should be more data-centric. These systems should describe how data is laid out in memory and the loop statements should operate locally on data similar to MapReduce [11].

From hardware point of view, the cost to move a bit is proportional to distance. The emerging hardware constraints are increasingly mismatched with the current programming paradigm [11]. Hardware/software co-design must consider better decisions about the future programming environment together with performance. Intel has already established co-design centers world wide to understand what the system developers need so they can develop a hardware that can be efficiently utilized by the system developers [11].

It is not only about the flops we count, it is about seeking a balance in terms of data movement and FLOPs. A better data movement management system and innovative solutions to minimize data movement across the system is extremely vital. We need to come up with HW/SW co-design for a better data movement management.

5. Algorithms

The advances in the architecture of super computers raise the need for changing the programming systems approach. The applications in supercomputing systems need to run more efficiently in terms of scalability, reliability and data movement. Dealing with parallelism and data locality for the exascale supercomputing is challenging. It is impossible for the programmer to manage a high-end system with several million processors in terms of load balancing, failures, and data locality. These machines should have the capability to manage the applications at the runtime [7], [4], [5].

Improving the algorithms could result in more efficient power management system. Self-aware systems is one approach toward solving this problem [5], [2]. In other words, the code should be able to exploit the efficiency of the machine. For example, shutting down the network while the machine is doing calculations to save the energy consumed by the network and turn the network on when data transfer is needed. That is essential for energy management. Traditionally, in computing environment that has always been handled by the operating system. Nowadays, having hardware that is able to implement event-based power management system is a necessity.

The programming models are increasingly mismatched with the reality of the underlying hardware architecture. Rewriting the current algorithms and applications can contribute to define the architecture of the future exascale computers through the co-design with hardware architects. Data structures and algorithms should be optimized to minimize data movement in the system [7], [4], [11]. The programmer can specify what the machines need to do to run the applications efficiently and the architect will build the hardware based on that [4], [11].

The software developers understand the requirements for exascale applications so they can provide feedback to the hardware architects and provide guidelines to build exascale HW and SW prototypes [4]. Because the programming model should be a reflection of the underlying machine architecture, the co-design is essential to seek balance between cost-to-performance ratio. Even if there is a new technology in the hardware, we have to change the software to use this technology efficiently. That can be achieved by understanding the performance consequences for the software running on that machine.

6. Reliability

Reliability and fault tolerance are essential for high-end computing. The reliability in a system with several million processors is a difficult challenge because of their scale and complexity [10]. When operating the transistors at a high level, the chances of failure increases which make the

reliability an essential factor in developing such a high-end system [3]. Failure in HPC is very costly and might result in the loss of a very large amount of computed work. Additionally, these systems involved long-running jobs that should finish in a timely manner. The applications and the system in general should be more resilient and able to recover from errors.

Traditionally, defining a checkpoint-restart was a solution. However, at this scale, checkpoints are problematic due to the overhead involved in terms of cost and energy [7], [9], [10]. It is used to be the hardware job, for the future HPC it became a hardware and software challenge [3]. Beside, Memory should have more significant and sufficient error correction codes should be implemented.

To develop a resilient system for future HPC with exaFLOP, the more transistors we have the more reliable the system should be. Many approaches have been proposed such as using state machine replication [7] which was adopted from the high-availability systems. Other studies [9] suggested improving the data locality in order to improve system reliability. However, adopting well-known fault tolerant techniques are not the solution. Data integrity and consistency in the event of failure is an important point [6]. Therefore, the entire software stack running on the system should be fault tolerant and aware. That's to ensure the integrity of the whole system and avoiding building a reliable system over unreliable one [10].

7. Conclusion

The current emphasis is on preserving the FLOPs. The real cost now are not FLOPs, it is data movement. That requires shifting to data locality centric programming paradigm and developing hardware features to support it. The programmers should focus on developing applications with significant parallelism capabilities. With the new technologies involved we can get the capacity and density we need. Better simulation tools are important to predict the performance of theses machines without the cost of building real machines.

References

[1] B. Hayes, "Built for speed: Designing exascale computers," 2014.
[2] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *High Performance Computing for Computational Science–VECPAR 2010*. Springer, 2011, pp. 1–25.
[3] K. Ferreira, J. Stearley, J. H. Laros III, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold, "Evaluating the viability of process replication reliability for exascale systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 44.
[4] J. Follows, "Seventh framework programme," 2012.
[5] M. Hall, R. Lethin, K. Pingali, D. Quinlan, V. Sarkar, J. Shalf, R. Lucas, K. Yelick, P. C. Diniz, A. Koniges, *et al.*, "Ascr programming challenges for exascale computing," 2011.
[6] T. Trader, "Doe exascale roadmap highlights big data."
[7] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, 2008.
[8] J. Shalf, "Computer architecture for the next decade," *International Journal of High Performance Computing Applications*, 2013.
[9] D. Zhao, D. Zhang, K. Wang, and I. Raicu, "Exploring reliability of exascale systems through simulations," in *Proceedings of the High Performance Computing Symposium*. Society for Computer Simulation International, 2013, p. 1.
[10] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *International Journal of High Performance Computing Applications*, 2009.
[11] S. Amarasinghe, D. Campbell, W. Carlson, A. Chien, W. Dally, E. Elnohazy, M. Hall, R. Harrison, W. Harrod, K. Hill, *et al.*, "Exascale software study: Software challenges in extreme scale systems," *DARPA IPTO, Air Force Research Labs, Tech. Rep*, 2009.

SESSION

COMMUNICATION TOPOLOGIES, INTERCONNECTION NETWORKS, AND RELATED ALGORITHMS

Chair(s)

TBA

Parallel Packet Processing on Multi-core and Many-core Processors

Andy Harvath, Hiroaki Nishi
Graduate School of Science and Technology, Keio University, Japan
harvath@west.sd.keio.ac.jp, west@sd.keio.ac.jp

***Abstract*—The Service-oriented Router (SoR), a highly functional router based on a novel router architecture, enables unprecedented web services traditional routers were unable to provide. The SoR performs Deep Packet Inspection (DPI) to analyze Layer 7 information, which is becoming increasingly difficult due to the substantial increase in Internet traffic. Meanwhile, multi-core processors and general-purpose many-core processors are increasing in popularity. These highly programmable many-core processors are suited for parallel packet processing and propose a parallelization method for packet inspection. The method is applied to NEGI, a software SoR simulator. The parallelized software is then implemented on a multi-core Xeon CPU to test for effectiveness and scalability. While the results confirm its scalability, we find that the throughput of the output process must be improved for SoRs to benefit from the proposed method.**

Keywords—Service-oriented router; multi-core; many-core; packet processing

I. INTRODUCTION

With the growing ubiquity of smartphones and other communication devices in the past decade, the Internet has become increasingly accessible. As a result, contents transferred over the Internet are expanding in both amount and diversity. For the Internet to provide high-quality services under these circumstances, it must not only transfer data from one end host to another at high speed, but also provide the user with valuable information efficiently.

Examples of an approach to improve delivery efficiency include Content-Centric Networking (CCN) [1]. CCN was introduced as an alternative networking paradigm based on named data rather than named hosts and is believed to enable efficient content distribution. The Service-oriented Router (SoR) [2] is a router architecture that takes a similar information-centric approach. The SoR is capable of observing traffic data stream, inspecting packet payloads, and storing data in databases. Content-based routing can be performed by an SoR router using the extracted data, sending information to where it is demanded. SoR also enables other unprecedented web services that traditional routers were unable to provide, such as a router-based Network Intrusion Detection System (NIDS).

One of the existing challenges to the SoR is to achieve wire-rate throughput. The rapidly growing Internet traffic demands high transfer speed for backbone routers, which means that SoRs must extract data from packets at an equally high processing speed.

Meanwhile, parallelization has become the most common approach for speedup, both at the hardware level and at the software level. We have entered the "era of higher processor parallelism" [3], in which superior microprocessor performance is gained from high parallelism and not from high clock speed. A majority of processing units that are currently in use implement multi-core architectures. In addition to these multi-core processors, many-core processors have been introduced for applications that make use of even higher parallelism. Many-core generally refers to computing devices that have exceptionally large numbers of processors on a single chip. Examples of many-core processors include Graphic Processing Units (GPU) and Intel's Many Integrated Core (MIC) processors.

Given these backgrounds, parallelizing the SoR's packet inspecting process is a natural approach to solving the throughput problem. Because a large number of streams flow through a router simultaneously, there should be sufficient concurrency to utilize the highly parallel hardware. Moreover, each TCP stream is entirely independent, which means that all can be processed concurrently without the need to lock resources.

Because one of the main ideas behind the SoR is to make it flexible and programmable, a hardware-based approach is not preferable. One of the most popular forms of many-core processors are GPUs, and a section of a software can be run on a GPU to increase its throughput. However, GPUs are optimized for floating-point Single Instruction Multiple Data (SIMD) instructions. The threads in a GPU are grouped into fixed sized batches called warps, and all threads in a warp has to execute the same instruction. This makes conditional branches in parallel codes very slow. While GPUs lack the the flexibility for complicated packet processing functions, the MIC architecture consists of 50 to 60 simple in-order x86 cores connected to a bidirectional ring bus and is known to be highly programmable. In addition, because MIC processors support the x86 instruction set, codes that are optimized for any x86 CPU will also run on and MIC chip. Although the theoretical speed for current implementations of MIC is known to be extremely difficult to achieve [4], we believe that the high parallelism and programmability of MICs are suited for parallel packet processing.

This study focuses on the parallelization of NEGI [5], a software implementation of SoR written in C/C++. We

propose a parallel method to reconstruct TCP streams, extract content information, and output results to a database from Internet packets. The proposed method was implemented on a 12-core, 48 thread Intel Xeon Processor to test scalability and prepare for its future implementation on Intel MIC processors.

The rest of this paper is structured as follows. In section II, several works related to parallel packet processing and many-core parallelization are introduced. Section III describes the NEGI application in detail. The parallelization method is proposed in Section IV, and its test results are presented in Section V. Finally, we conclude the paper in Section VI.

II. RELATED WORKS

Parallel packet processing has drawn a lot of interests recently. As more and more network processors implement multi-core architectures, the demand for parallel applications that exploit these architectures is growing. Vert Paxon et al. introduced an event-based framework for parallelizing Network Intrusion Prevention Systems (NIPS) on multi-core processors [6]. Yunchun Li et al. proposed a packet processing model and calculated the theoretical speedup of parallelization of DPI systems [7]. Most of the research in parallel packet processing target network processors, and there are no studies on the use of MIC processors for packet processing.

III. SoR SIMULATOR: NEGI

NEGI is a Layer 7 information extractor developed for simulating and evaluating SoR. NEGI is written in the C/C++ programming language, and uses the libpcap library to processes packets from either a Linux Ethernet device or a pcap file. The current version of NEGI loads a user-defined filter and applies it to the incoming packets before saving the resulting data in a SQLite database file, along with basic information such as source/destination IP addresses, source/destination port numbers, and the protocol number. Below is a more detailed description of how NEGI operates.

Figure 1 shows the architecture of the NEGI application, which can be divided into several function blocks that interact with each other. These modules include:

- Packet capture engine
- TCP reconstructor
- Layer 7 decoder
- String matching engine
- Database insertion engine
- TCP timeout manager

A. Packet capture engine

As noted, NEGI makes use of the libpcap library to monitor a Linux Ethernet device. The packets are written in NEGI's shared memory, and its pointers are added to a message queue. Since certain packets can be instantly discarded, the processing time for the packet capture engine is not uniform. The packet engine, therefore, works independently from the succeeding modules. The interface between the packet capture engine and the TCP reconstructor engine is provided by a message queue. By splitting the capture engine and the processing engines, NEGI conceals the unevenness in processing each packet.

B. TCP reconstructor

Figure 2 is a diagram that shows how the TCP reconstructor processes each packet using a context switch. A, B and C in the figure refers to different TCP packet streams. In addition, the units labeled A1, B1, etc. each represents a single packet. For example, B2 is a packet with sequence number 2 that belongs to TCP stream B. Finally, the packets stored in Stream Reconstruct Information are prefixed with asterisks to emphasize the fact that they are merely pointers that point to where the packets are stored.

In figure 2, packets A1, B1, A2, and C1 have already passed, and packet C2 is being processed. When a packet is sent to open a TCP connection, the TCP reconstructor acknowledges it, creates a new TCP stream information frame, and passes the information to the subsequent modules. After all the main modules process the packet, the processing states of each module is saved as Stream Reconstruct Information. This information is recalled when another packet from the same stream is loaded. In figure 2, C1 has created Relevant Information C and Intermediate State C, which are loaded by the TCP reconstructor to process C2. When the TCP reconstructor detects an end of a stream, the stream results are finalized and saved to the database. The stream information is then freed from memory.

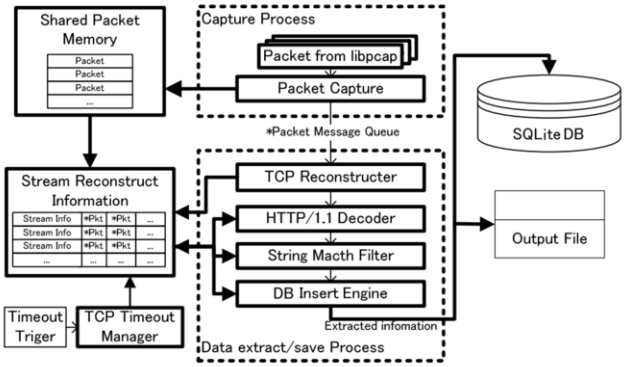


Fig. 1. The NEGI architecture

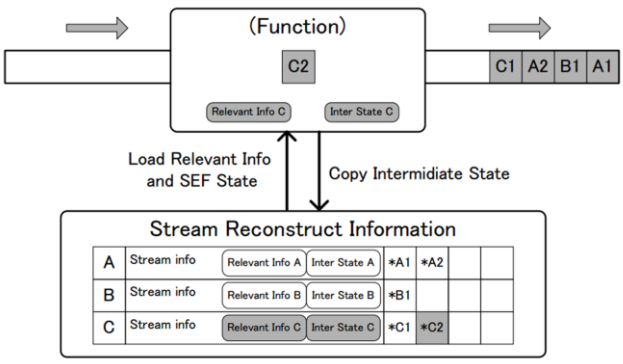


Fig. 2. TCP reconstruction in NEGI

The TCP reconstructor assigns packets to different streams by analyzing the TCP header. Namely, the source and destination IP addresses, source and destination Port Numbers, and the Protocol numbers are used to distinguish TCP connections. Stream information is created when the TCP reconstructor detects a new SYN packet, and destroyed when a FIN or a RST packet is captured.

C. Layer 7 decoder

The Layer 7 decoder uses the stream context information to decode various application protocols to enable string matching and other processes. The targets include HTTP/1.1's chunk encode and gzip encode. When the decoder reaches the end of a packet, intermediate states are saved in a format specific to the protocol type.

D. String matching engine

The string matching engine loads user-defined matching rules from a database and applies them to incoming packets. When matching strings or patterns are detected in the packet content, they are passed to the database insertion engine. When there is a matching pattern between multiple packets, the string matching engine will reach the end of a packet while it is matching a string. In such cases, the state is saved as stream context and recalled when a succeeding packet arrives. The version of NEGI used in this study does not support advanced regular expression matching.

E. Database insertion engine

If any defined strings or patterns are found in a TCP stream, the database insertion engine stores the following information in an database: an ID to identify the stream, timestamp of its arrival, destination IP address, source IP address, destination port number, header information, and an ID to label which rule was applied. If any strings were extracted, the database insertion engine also saves the strings to the database with corresponding TCP stream IDs. As previously noted, the version of NEGI that was used in the study uses a SQLite3 database, which comes in the form of a single file, to store the results.

F. TCP timeout manager

When dealing with real internet traffic, it is not guaranteed that all TCP connections close normally. If for some reason the closing packets are not detected, the context information of the stream could be stored in memory as long as the NEGI process is up and running. To avoid memory leaks in this situation, the TCP timeout manager monitors each stream. The TCP manager ahs two main functions. First, if no packets are received from a stream for duration, the TCP manager deems it as closed and frees all relevant information. Another function for the TCP timeout manager is to destroy stream data depending on available memory. If a situation of memory overuse is detected, the TCP timeout manager deletes the least active streams to fit the memory requirements. The timeout duration and the maximum available memory are user-defined and written in a configuration file.

IV. PARALLELIZATION METHOD

To keep pace with the substantial increase in internet traffic, we proposed a method to parallelize the process of NEGI and increase its throughput. The parallel version of NEGI (pNEGI) is written in the C language and is designed to be run on an x86-based multicore system and ultimately on an MIC processor. pNEGI uses the POSIX threads library to create threads and utilize the multiple cores provided by the hardware.

A. Limitations

The software architecture of NEGI is as previously shown in Figure 1. Out of the modules illustrated in this diagram, The packet capture engine cannot be parallelized at the software level. This is due to the libpcap library's single thread nature; the packets are inputted serially. Since NEGI operates on the SQLite3 library, the output is a single file, and hence, also serial. A parallel write to one file generally does not give performance because the file must be locked frequently. Therefore, NEGI can be assumed a single input, single output model.

On the other hand, multiple instances of other modules that operate in the main thread, namely the TCP reconstructor, the L7 decoder, and the string match engine can be created and run simultaneously. This allows multiple threads to handle different streams at the same time. However, it must be noted that packets have sequence numbers, and ones that belong to the same stream must be processed in order. In addition, for there to be no dependency between processing threads, the same stream has to be processed in the same thread.

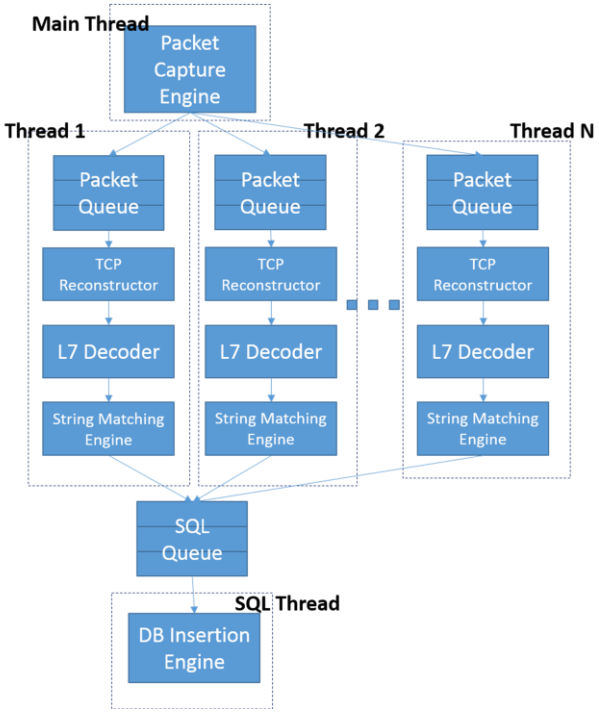


Fig. 3. The pNEGI architecture

B. Architecture

Figure 3 shows the architectural structure of pNEGI. Parallelizing a single-input, single-output software model like one of NEGI requires some kind of a fork-join structure. As shown in figure 3, pNEGI implements this by placing ring

buffer queues at the diverging and converging points. These queues not only allows for single-to-multiple and/or multiple-to-single message passing between threads, but also lets the threads work asynchronously. This is especially important for this architecture, because neither the number of packets assigned to each core nor the processing time for each packet is known.

C. Behavior

When pNEGI is started, the main thread performs the initializing tasks, which include creating new processing threads and various module instances. The number of threads is user-defined; it will be read from the configuration file. The main thread then opens either an Ethernet device or a pcap file and begins reading.

When the main thread captures a packet, the header and content data is distributed to the processing threads. Specifically, each thread owns a packet queue, and the main thread pushes the packets to them. As previously mentioned, the main thread cannot simply distribute the data evenly in a round robin fashion. Packets belonging to a certain stream depend on each other and have to be processed in a single thread. Both to assign each stream to the same thread and to balance the load, pNEGI uses a hashing technique to determine the receiving thread.

Cyclic Redundancy Check (CRC) is a method to detect accidental changes to data. CRC is mainly used in networking to check for unintended bit errors, but can also be used as a hash function in a non-security context. pNEGI inputs a concatenation of source IP address, destination IP address, source port number, and destination port number to a CRC function that outputs an 8-bit hash value. This hash value is divided by the number of threads and the remainder is used to determine the thread to assign packet data. CRC was implemented because of its simplicity and efficiency, and the output was set to 8bits (0-255) to fit the maximum number of threads on an MIC processer, which is currently 244.

Each processing thread receives a signal when a new packet is added to its queue. The TCP reconstructor responds to the signal by dequeuing the packet and processing it. The packet data is then transferred to the L7 decoder and the String match engine in the same manner as the single-thread NEGI. When a packet is done processing, its intermediate status is saved to the thread's unique stream information pool. Each processing thread has its own memory area to store information for stream reconstruction for increased independency among threads. For the same reason, each processing thread also has its own TCP timeout manager that monitors each stream.

If a processing thread has valuable data or information that has to be stored in the database, it generates a SQL command string to insert the data. The SQL string is enqueued to the SQL queue and eventually passed to the database insertion engine, which executes the given command to the SQLite3 database. The database insertion engine itself runs in an independent thread, which we call the SQL thread, to prevent multiple threads trying to open the database file, and to take away computation from the processing threads.

TABLE 1. INTEL XEON E5-2697 PROCESSOR SPECIFICATIONS [8]

# of Cores	12
# of Threads	24(48)
Instruction Set	64bit
Processor Base Frequency	2.7GHz
Max Memory Size	768GB
Max Memory Bandwidth	59.7GB/s

V. EXPERIMENTAL RESULTS

Both the NEGI and its parallelized version, pNEGI, were implemented on a 12-core Intel Xeon CPU to evaluate the proposed method. A pcap file was inputted to each software and the processing times were measured to calculate throughput. pNEGI, was also tested for different numbers of threads.

A. Expermental environment

All experiments were performed on an Intel Xeon E5-2697 CPU. Table 1 lists the basic specifications for the processor. As shown in table 1, the Xeon processor used in the study has 12 cores with 2 implemented hardware threads on each core. With Intel's Hyper-Threading Technology (HTT) [9], these cores can execute up to 4 threads concurrently with comparatively lower performance.

A 1.5MB pcap file was used as a controlled input. This file was dumped from our laboratory's gateway server in a five-minute interval, and included totals of 1,391,020 packets and 47971 TCP streams.

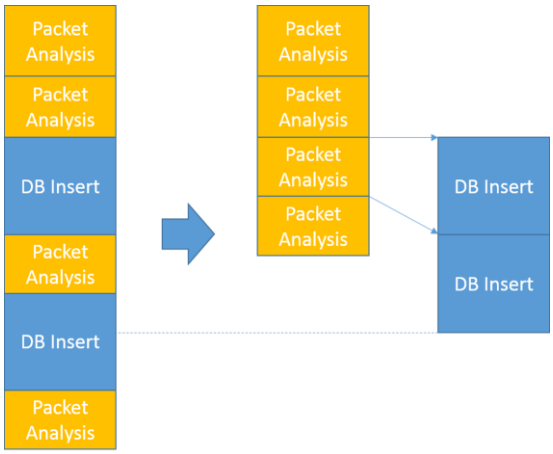


Fig. 4. Speedup from extraction of insertion process

B. Results: Database insertion engine enabled

The results are shown in Figure 5 and Table 2. As it can be seen from both figures, pNEGI recorded a throughput approximately 3Mbps higher than that of NEGI. However, it can easily be deducted that the speedup is not a result of the parallelization of packet processes. pNEGI exhibited the best performance with only 1 processing thread; distributing packets among multiple processing threads did not increase the performance, but rather reduced it.

TABLE 2. RESULTS WITH DB INSERTION ENABLED

	Max Throughput (Mbps)	# of Threads
NEGI	41.44	/
pNEGI	44.63	1

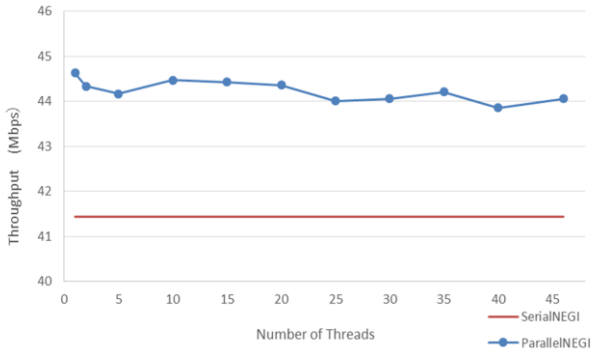


Fig. 5. Throughput of the NEGI applications with DB insertion enabled

TABLE 3. RESULTS WITH DB INSERTION DISABLED

	Max Throughput (Mbps)	# of Threads
NEGI	420.96	/
pNEGI	1025.97	40

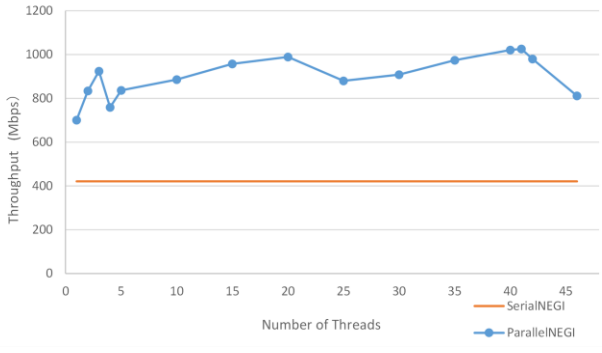


Fig. 6. Throughput of the NEGI applications with DB insertion disabled

It is easy to conclude that the 3Mbps speedup was due to the implementation of the SQL thread. Extracting the insertion process from the packet inspection process allows for a faster processing of packets, as shown in figure 4. The decrease in throughput at larger numbers of threads can be attributed to the resource costs of thread creation, and possibly to the more frequent locking and unlocking of the semaphore of the SQL queue to serialize its accesses. Finally, it can be presumed from the flat graph that the database insertion process was a bottleneck; parallelizing the packet inspection process had nearly no effect to the overall throughput because the database insertion engine could not process the SQL messages as quickly as the processing threads processed the packets.

C. Results: Database insertion engine disabled

Another experiment was performed to test the hypothesis that the output was a bottleneck. In this experiment, the database insertion engine received the SQL command strings, but did not execute the commands.

An overall increase in throughput can be observed from the results in figure 6 and table 3. As opposed to the previous results, increase in the number of threads generally resulted in high throughput. pNEGI scaled in what can be seen as a linear rate up to about 20 processing threads. This is because the Xeon processor has 24 hardware threads. For a bigger number of threads, the processer made use of Intel's HTT technology to execute them using limited resource. Finally, at about 46 processing threads (48 total running threads), the processor had simply reached the maximum number of threads that it could run, and the performance plummeted. The results clearly supported the hypothesis that database insertion was NEGI's bottleneck.

VI. Conclusion

In this study, we have proposed a parallelization model for SoR's packet inspection process and applied it to the software SoR simulator, NEGI. The parallelized software was implemented on an Intel Xeon CPU, where its scalability was confirmed. We also found that for the SoR to benefit from this method, a critical bottleneck has to be removed: it will have to speed up the database insertion process if any data has to be saved. Possible methods of this include the parallelization of the database insertion process, the use of on-memory databases, the use of no DBMSs at all, or a combination of these methods.

The results suggest that pNEGI will also scale on an MIC processor. The fact that parallelization was effective for processing a dump file from a low-scale private network indicates that it will be equally, if not more effective in real-world situations, where greater concurrency is expected. The application will definitely not exhibit the same rate of increase in performance because although MIC processors have a much larger number of cores, each core is substantially slower than today's CPU cores, especially for integer operations. Despite these limitations, we believe that the MIC architecture is suited for parallel packet processing.

Our ultimate goal is to implement pNEGI on an MIC processor and evaluate the effect of highly parallel hardware on parallel packet processing. The results of this study are essential because it ensures good scalability on the Xeon CPU and hence suggests scalability on MIC processors as well.

VII. Acknowledgement

This work was partially supported by the funds of SECOM Science and Technology Foundation, and by MEXT/JSPS KAKENHI Grant (B) Number 24360230 and 25280033.

VIII. References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. "Networking named content." Proc. Of the 5th international conference on emerging networking experiments and technologies, ser. CoNEXT '09

[2] K. Inoue, D. Akashi, M. Koibuchi, and H. Nishi. "Semantic router using data stream to enrich services." In 3rd International Conference on Future Internet (CFI), pp.20-23, June 2008.

[3] Jeffers, James, and James Reinders. Intel Xeon Phi coprocessor high-performance programming. Newnes, 2013. Juniper Networks. http://www.juniper.net.

[4] Ramachandran, Arunmoezhi, et al. "Performance evaluation of NAS parallel benchmarks on Intel Xeon Phi." Parallel Processing (ICPP), 2013 42nd International Conference on. IEEE, 2013

[5] Masuda, Kazuki, Shinichi Ishida, and Hiroaki Nishi. "Cross-site recommendation application based on the viewing time and contents of webpages captured by a Network Router." ICOMP, Las Vegas (2013).

[6] Paxson, Vern, Robin Sommer, and Nicholas Weaver. "An architecture for exploiting multi-core processors to parallelize network intrusion prevention." Sarnoff Symposium, 2007 IEEE. IEEE, 2007.

[7] Li, Yunchun, and Xinxin Qiao. "A parallel packet processing method on multi-core systems." Distributed Computing and Applications to Business, Engineering and Science (DCABES), 2011 Tenth International Symposium on. IEEE, 2011.

[8] Intel Xeon Processor E5-2697 v2.
http://ark.intel.com/products/75283/Intel-Xeon-Processor-E5-2697-v2-30M-Cache-2_70-GHz

[9] Tian, Yuan, Chuang Lin, and Kangqiao Hu. "The Performance Model of Hyper-Threading Technology in Intel Nehalem Microarchitecture." 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE). Vol. 3. 2010

A Data Communication Reliabilty and Trustability Study for Cluster Computing

Eduardo A. Colmenares[1], **Per Andersen**[2]
[1]Department of Computer Science, Texas Tech University, Lubbock, Texas, USA
[2]High Performance Computing Center (HPCC), Texas tech University, Lubbock, Texas, USA

Abstract - *In HPC most of the problems under study will be either embarrassingly parallel, or data dependent. Beyond the nature of the problem, scientists will be interested in either one or two additional characteristics. The first, performance, focuses in achieving an accurate solution in a fraction of the time of a sequential approach. The second is consecutive, accurate and steady time readings. In their quest for performance, some scientists forget that the chosen tool, in many cases a distributed-memory system, is not only a multi-user system, but also that its components are interconnected through a high-speed communications network to facilitate the interaction among processors. In this paper, we show why a cluster characterization is relevant, particularly for scientific kernels where multiple accurate and consecutive time readings are necessary to statistically validate a behavior. We provide the characterization of two clusters by using two variants of the ping pong test. One of the clusters is a multi-user research oriented cluster, while the second is a one-user cluster with older technology.*

Keywords: high performance computing, cluster, ping-pong, queue, normalization, average.

1 Introduction

High performance computing and parallel processing are now relevant to a variety of sciences, not only computer science. Nowadays, more researchers in different fields of science are considering new and cutting edge technologies capable of considerable computational power. A widely used technology comes in the form of distributed-memory systems also known as clusters, which combine the capabilities of multiple computational nodes via a high speed communication network. In most of the cases these distributed-memory systems will have a considerable number of users, which creates the need for scheduling policies capable of handling multiple job submissions from the same or different users, who may be competing for resources.

In this research, we acknowledge two potential characteristics for researchers interested in parallel processing. Performance and consistent time accuracy. For some researchers, achieving an accurate solution in a fraction of the time needed by a sequential approach is the ultimate goal; however, for some others the goal is not only the former, but also to build the foundations to statistically validate the behavior of a phenomena which requires steady, accurate, and consistent time readings over multiple executions of the kernel that represents it.

In order to identify if the scientists have a testing environment that will allow consistent and accurate time readings over consecutive executions of the kernel, we propose a reliability study through the characterization of two clusters of different nature, by executing on each one of them two different variants of the ping pong test.

2 Ping Pong Test

First, we introduce some terminology used throughout the remainder of the paper. As suggested by [8], the purpose of the ping pong test is to provide a measure of the end-to-end delay time associated with sending a message back and forth between processes in a cluster of workstations or any other parallel hardware system. Two different variants of ping-pong tests which we will refer to as Ping-Pong-A and Ping-Pong-B, were used in order to evaluate if the communication network was capable of providing a clean and steady communication time between all participant processes [2].

The purpose of Ping Pong-A test is to see if repeatedly sending a message between two workstations and their associated MPI processes results in reliable timings, while the purpose of Ping-Pong-B test is to see if alternating messages between workstations and their associated MPI processes results in any change in the latency timings. We expect both tests to generate the same results.

2.1 Ping-Pong-A

This first ping pong test will execute the ping pong core procedure 100 times between two workstations, where one workstation executes process P0 and the other workstation executes process P1. The test is then repeated 100 times between two different workstations, in this case the pair (P0, P2). The same procedure is repeated until all pairs of workstations (P0, Pi) have executed the Ping-Pong-A test. Each workstation is executing the SPMD ping pong application where P0 is defined as the master process, while a Pi process where i ranges from "1" to the "total number of participant process -1" will be considered a slave process

[2,8]. Typically each workstation executes only one MPI application as a process at any given time; therefore, P0 can be viewed as the process executing on workstation 0 and P1 as a process executing on workstation 1 and so on.

For each one of the executions of this version of the ping pong test, the communication time is recorded and saved into a spreadsheet, which will be used for posterior statistical analysis and validation. Figure 1 shows the test and its corresponding pseudo-code.

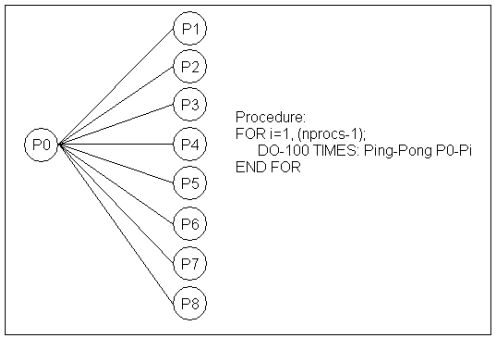


Figure 1: Diagram and Pseudo-Code for Ping-Pong-A.

2.2 Ping-Pong-B

The Ping-Pong-B test consists of two phases. In the first phase P0 will execute the ping pong core procedure with all participant processes: (P0, P1) → (P0, P2), .. → .., (P0, $P_{nprocs-1}$). The time required to complete the communication to each process on each workstation is registered into a spreadsheet for posterior analysis, as done in section 2.1.

The second phase of this ping pong test is to repeat the first phase until 100 sets of timings have been collected. Figure 2 shows the test and its corresponding pseudo-code.

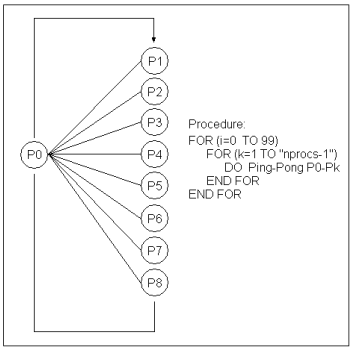


Figure 2: Diagram and Pseudo-Code for Ping-Pong-B.

3 Testing Environment

We used two different testing environments. The first testing environment corresponds to a community cluster, which provides access to high performance computing hardware and software for the local research community. For the remainder of this paper we will refer to it as the Community-Cluster. The second testing environment is a personal cluster with only one user and a maximum of 9 nodes, we will refer to this cluster as My-Cluster.

3.1 Community-Cluster

This cluster has 12TB of public shared Lustre storage and three groups of public and private nodes, all connected by SDR Infiniband and Gigabit Ethernet. The quad-core nodes have Infinihost III Lx (PCI-e) cards, and the older nodes have Infinihost (PCI-X) cards.

3.1.1 Public quad-core (512 cpu, 4.77 TF).

64 nodes with dual quad-core Intel 5345 processors (2.33 GHz) and 12GB of memory each. Designated compute-1-x, 2-x.

3.1.2 Public single-core (128 cpu, 0.82 TF).

64 nodes with dual single-core Intel "Irwindale" processors (3.2 GHz) and 4 GB of memory each. Designated compute-3-x, 4-x, 5-x.

3.1.3 Public AMD dual-core (8 cpu, .04 TF).

1 node with quad dual-core AMD 8218 processors (2.60 GHz) and 64GB of memory. Designated compute-8-1.

3.2 My-Cluster

Each one of the nodes in this cluster has the following characteristics: one Intel(R) Pentium(R) 4 CPU at 1.70GHz, one 3Com PCI 3c905C Tornado network card. All the nodes in this cluster are interconnected via a 3Com® Super Stack® 3 Switch 3300 12-Port. Table 1 summarizes the major hardware differences between nodes.

Table 1: Major hardware differences among nodes for My-Cluster

NODE	MEMORY (MB)	HARD DISK
1	511.46	40020 MB-T340016A, ATA
2	1023.4	40020 MB-T340016A, ATA
3	1023.4	20547 MB-MAXTOR 6L020J1, ATA
4	511.46	40020 MB-T340016A, ATA
5	1023.4	20547 MB-MAXTOR 6L020J1, ATA
6	1023.4	20547 MB-MAXTOR 6L020J1, ATA
7	1023.4	40020 MB-WDC WD400BB-75DEA0, ATA
8	1023.4	40027 MB-MAXTOR 6L040J2, ATA
9	1023.4	40027 MB-MAXTOR 6L040J2, ATA

4 Experimental Results

The purpose of both Ping-Pong tests (A,B) is to measure the end-to-end delay sending a message back and forth between processes. The size of the message was 8 bytes for all testing environments, and both versions of the Ping Pong test.

4.1 Ping-Pong-A in the Community-Cluster

The Community-Cluster is a multi-user cluster, which has several queues where jobs can be submitted. Following a recommendation from the personnel in charge of managing

this cluster, three public queues (2Wkpar, 48Hquadpar and 48Hpar) were used for our testing purposes. In addition, a script and configuration files were used to force a single process per node or workstation. Otherwise, according to [2, 5] given the multi-processor nature of each workstation, more than one process would execute in each node, which would skew the communication timings collected.

4.1.1 2WKpar Queue

Table 2 summarizes the results for the Ping-Pong-A test for the 2WKpar queue at the Community-Cluster. Each one of the rows in this table presents the statics derived from the 100 executions of this test.

After obtaining the results of this test, it was possible to observe two important events. The first is that when communication is established for the very first time between a pair of processors, the corresponding initialization cost is high. The second event is that for consecutive interactions between the same pair of processors, the cost of such interaction was considerably smaller than the first interaction.

Table 2: Ping-Pong-A- Community-Cluster -2WKpar-All Samples

Community-Cluster - 2WKpar - 9P – 1st Sample Included				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	1218.29	12091.28	8	120922
P0 - P2	751.8	7424.667	8	74256
P0 - P3	598.89	5895.264	8	58962
P0 - P4	496.8	4885.778	7	48866
P0 - P5	599.54	5911.764	7	59126
P0 - P6	600.08	5905.447	8	59064
P0 - P7	597.5	5893.182	7	58940
P0 - P8	599.12	5908.675	7	59095

The high price to pay for the very first interaction between a pair processors is a one-time event; this can be observed by simple comparison of Tables 2 and 3. The only difference between these two tables, is that table 3 does not include the first sample or time reading. The notorious and drastic difference between the tables clearly identifies the first time reading as an outlier and justifies its removal from all subsequent analysis.

For simplicity, but without lack of generality, none of the remaining tables associated with any of the clusters, queues, and ping pong tests will include the first sample or time reading.

Figures 3, 4, and 5 show the behavior for all 100 samples in all three queues of the Community-Cluster. Each one of these curves has been normalized by computing the timings as a ratio of the average. The idea behind this normalization is to allow an easy but effective comparison of the curves. In the figures the curves show all pairs of processes P0-Pi, where i ranges from 1 to 8 for a total of 9 participating parallel processes [2].

Table 3: Ping-Pong-A Community-Cluster -2WKpar-1st Sample Removed

Community-Cluster - 2WKpar - 9P – 1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	9.161616	1.861024	8	25
P0 - P2	9.333333	1.8681	8	25
P0 - P3	9.363636	1.548235	8	21
P0 - P4	8.222222	1.644754	7	18
P0 - P5	8.363636	1.36617	7	16
P0 - P6	9.535354	2.149101	8	25
P0 - P7	8.181818	1.312138	7	17
P0 - P8	8.252525	1.592991	7	20

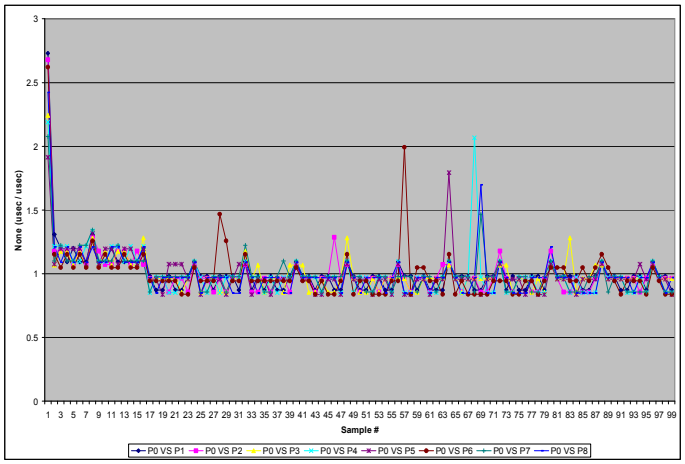


Figure 3: Normalized Curves for Ping-Pong-A Community-Cluster-2WKpar-9P.

4.1.2 48Hpar Queue

Table 4: Ping-Pong-A- Community-Cluster -48Hpar-1st Sample Removed

Community-Cluster – 48Hpar - 9P – 1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	12.80808	2.961235	10	27
P0 - P2	13.86869	2.448017	12	22
P0 - P3	13.49495	2.800779	11	26
P0 - P4	12.31313	2.648243	10	24
P0 - P5	13.88889	2.754917	11	27
P0 - P6	13.48485	2.708127	11	26
P0 - P7	13.52525	2.749187	11	25
P0 - P8	13.50505	2.588726	11	24

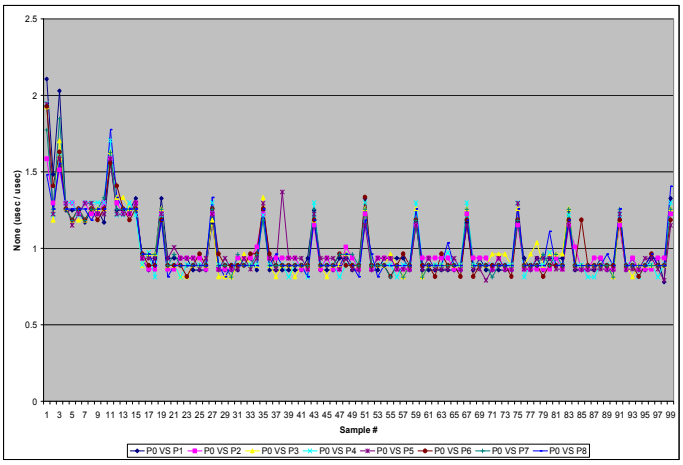


Figure 4: Normalized Curves for Ping-Pong A- Community-Cluster (48Hpar)-9P.

4.1.3 48Hquadpar Queue

Table 5: Ping-Pong-A- Community-Cluster -48Hquadpar-1st Sample Removed

Community-Cluster - 48Hquadpar-9P-1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	10.38384	2.141462	8	19
P0 - P2	10.25253	1.745802	8	21
P0 - P3	10.54545	1.540426	9	16
P0 - P4	9.373737	1.7237	7	18
P0 - P5	16.66667	53.26733	8	526
P0 - P6	10.14141	1.51866	8	18
P0 - P7	9.282828	1.450126	8	15
P0 - P8	9.363636	1.548235	8	16

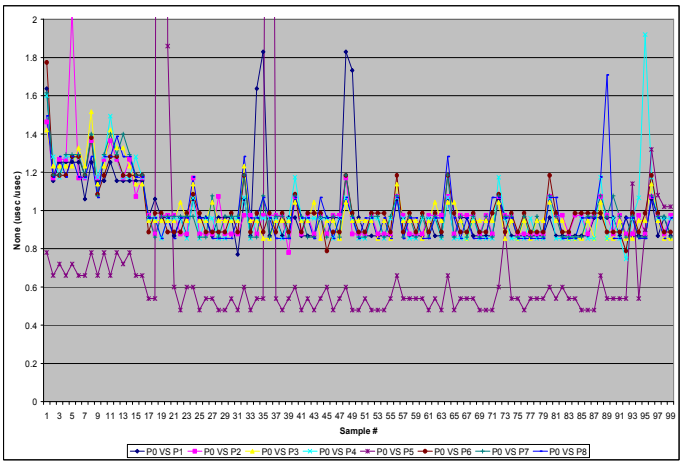


Figure 5: Normalized Curves for Ping-Pong-A- Community-Cluster (48Hquadpar)-9P.

Comparing Figures 3, 4, and 5, it is possible to notice that all three queues at the Community-Cluster exhibit a data pulsing behavior. It is relevant to mention that this cluster is accessible to a considerable number of users in a research community, and it is more than likely that more than one user will submit jobs for processing at the same time; this requires the use of additional processing layers such as scheduling, which may contribute to the pulsing nature of this shared cluster [2]. Because of the pulsing behavior of this cluster, any time reading will be more susceptible to a higher margin of error than a time reading in cluster with a non-pulsing behavior [1, 4]. It is important to be aware that using a cluster where this type of error may be present only affects the accuracy of time readings -not the accuracy of the computations being carried out.

4.2 Ping-Pong-A in My-Cluster

This cluster has a different architecture, operating system, network switch and communication capabilities than the Community-Cluster as indicated in section 3.2. The data collected in this cluster during the execution of this test are shown in table 6. Once again, the very first sample takes a considerable amount of time when compared to consecutive samples; as a consequence, it is possible to treat the first time reading as an outlier.

Table 6 presents the corresponding statistics after the removal of first time reading, the reasons that justify this approach are explained in section 4.1.1.

Table 6: Ping-Pong-A-My-Cluster-1st Sample Removed

My-Cluster - 9P - 1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	134.1111	19.91507	127	273
P0 - P2	144.202	49.94302	126	477
P0 - P3	140.9091	170.2679	114	1676
P0 - P4	138.0202	100.3862	115	821
P0 - P5	162.7172	185.8668	118	1495
P0 - P6	145.1515	163.1716	117	1575
P0 - P7	154.8485	152.9347	117	1086
P0 - P8	138.0101	123.824	116	1277

Comparing Figure 6 against Figures 3, 4, and 5, it is possible to conclude that My-Cluster does not have the same pulsing nature observed in all three queues of the Community-Cluster, (2WKpar, 48Hpar, 48Hquadpar).

Figure 6 also shows some other outliers, but it is easily observable that the communication provided by My-Cluster is less susceptible to pulses. It is also possible to observe regions where each one of the curves is almost flat for multiple consecutive samples. The previous observation clearly indicates a reduced number of disturbances and interferences during the test.

The previous conclusions and findings are supported by Figure 7, where a comparison of 100 consecutive executions of the Ping-Pong-A test for the pair of processors (P0, P1) in My-Cluster and the 48Hpar queue of the Community-Cluster are shown.

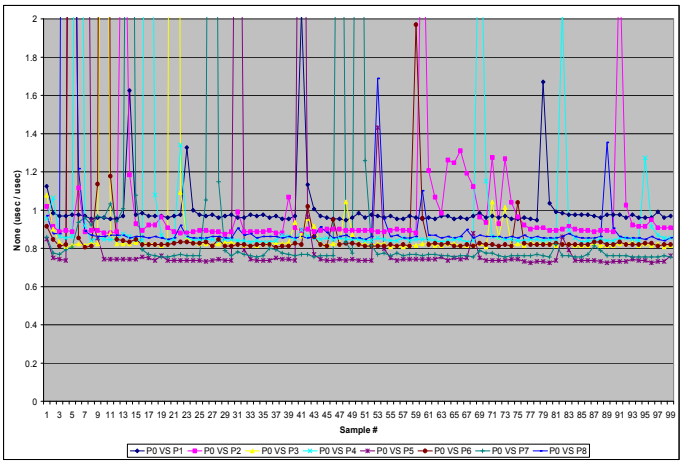


Figure 6: Normalized Curves for Ping-Pong-A- My-Cluster-9P.

It is relevant to mention that My-Cluster, was running in X-windows mode, which requires refreshing cycles; in addition, the system was also subject to mouse movements during the execution of the test, which may have affected the time readings at a lesser degree; however, these interruptions will certainly impact the results; therefore, must be avoided during data collection.

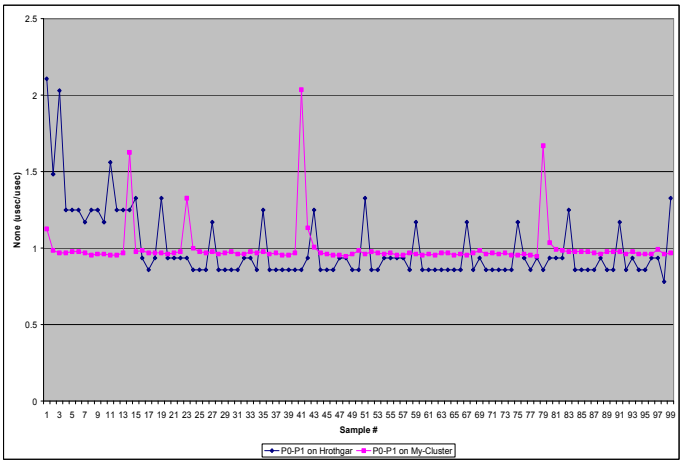


Figure 7: Normalized Curves Only for (P0, P1)-Overlapping Ping-Pong-A in Community-Cluster (48Hpar) and My-Cluster.

4.3 Ping-Pong-B in the Community Cluster

For this variant of the ping pong test, we considered the same three queues used during the study of the Ping-Pong-A test, which are 2Wkpar, 48Hquadpar and 48Hpar.

4.3.1 2WKpar Queue

Table 11 summarizes the results for the Ping-Pong-B test for the "2WKpar" queue at the Community-Cluster. Each one of the rows in this table presents the statics derived from the 100 executions of this test.

Table 7: Ping-Pong-B-Community-Cluster-2WKpar-1st Sample Removed

Community-Cluster - 2WKpar - 9P – 1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	9.585859	1.628885314	8	18
P0 - P2	9.292929	1.56656732	8	20
P0 - P3	11.54545	1.960178705	10	28
P0 - P4	10.25253	1.547502493	9	22
P0 - P5	13.0404	5.396369134	9	40
P0 - P6	9.282828	1.229239055	8	18
P0 - P7	25.40404	89.00159586	9	888
P0 - P8	10.26263	0.932254877	9	13

Three different figures illustrates the behavior of the ping-pong-B test for all 100 samples for each one of the queues. These figures are figures 8, 9, and 10. Each one of the curves on these figures has been normalized by computing the timings as a ratio of the average. This normalization was done to allow a better comparison of the curves. In the figures, the curves show all pairs of processes P0-Pi, where i goes from 0 to 8 for a total of 9 participating parallel processes

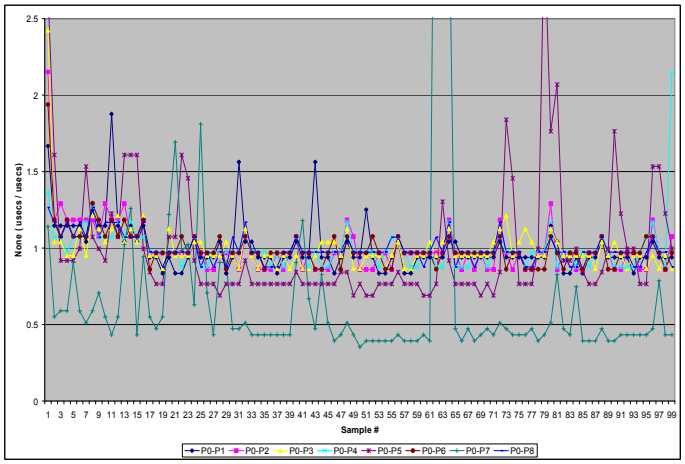


Figure 8: Normalized Curves-Ping-Pong-B-Community-Cluster-2WKpar-9P.

4.3.2 48Hpar Queue

Table 8: Ping-Pong-B-Community Cluster-48Hpar-1st Sample Removed

Community-Cluster – 48Hpar - 9P – 1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 – P1	12.19192	2.414567894	10	21
P0 – P2	12.9697	2.50083474	11	22
P0 - P3	11.89899	2.384023993	10	20
P0 - P4	12.93939	2.376836269	11	22
P0 - P5	12.94949	2.480017023	11	22
P0 - P6	14.77778	2.593380725	13	24
P0 - P7	12.89899	2.296825447	11	21
P0 - P8	13.27273	3.24766221	11	35

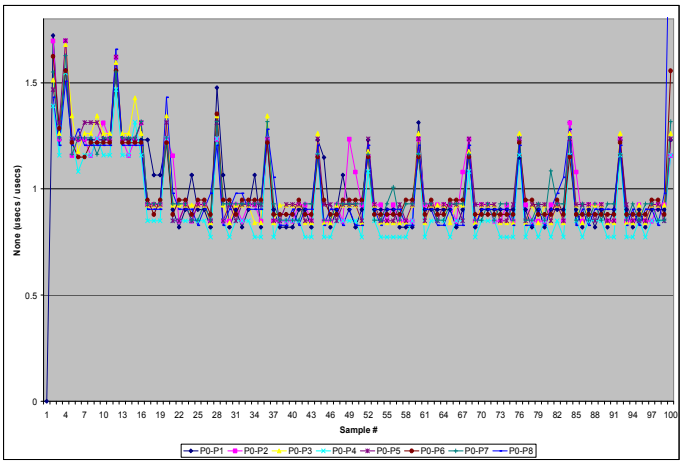


Figure 9: Normalized Curves-Ping-Pong-B-Community-Cluster-(48Hpar)-9P.

4.3.3 48Hquadpar Queue

Table 9: Ping-Pong B-Community CLuster-48Hquadpar -1st Sample Removed

Community-Cluster – 48Hquadpar - 9P – 1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	9.343434	1.691283875	8	19
P0 - P2	9.464646	1.547302665	8	21
P0 - P3	10.90909	2.321553443	9	27
P0 - P4	10.36364	1.438923531	9	22
P0 - P5	10.61616	1.838960835	9	22
P0 - P6	10.47475	1.053114277	9	15
P0 - P7	9.353535	0.872635362	8	13
P0 - P8	9.676768	2.668135032	8	28

The results and behaviors observed through the execution and corresponding analysis of the Ping-Pong-B test reinforce the conclusion that all three queues of the Community-Cluster exhibit a data pulsing behavior. Again, it is relevant to mention that due to its pulsing nature the Community-Cluster will provide time reading with a higher margin of error than a cluster with a non-pulsing behavior.

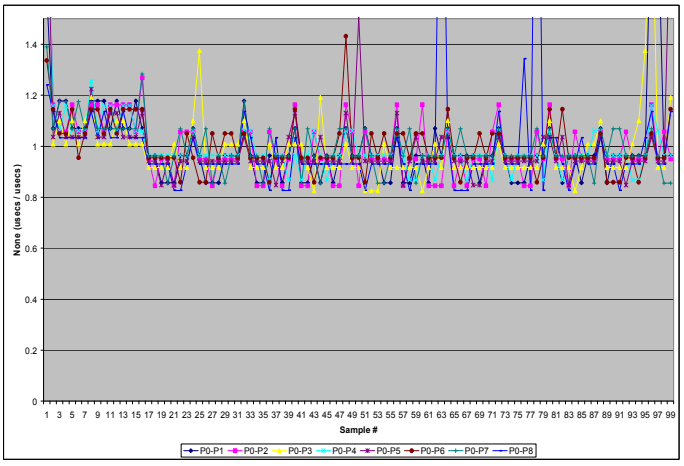


Figure 10: Normalized Curves-Ping-Pong-B-Community-Cluster-48Hquadpar-9P.

4.4 Ping-Pong-B in My-Cluster

This second variant of the ping-pong test was executed in a one-user cluster, referred to as My-Cluster, and the statistics can be observed on table 10.

Table 10: Ping-Pong-B-My-Cluster-1st Sample Removed

My-Cluster - 9P – 1st Sample Removed				
Proc-Pair	Average (usecs)	Stdv (usecs)	Min (usecs)	Max (usecs)
P0 - P1	155.5657	54.73240312	133	583
P0 - P2	199.4545	410.5913921	138	4221
P0 - P3	214.9293	788.5313299	122	7978
P0 - P4	261.9697	1216.547821	122	12236
P0 - P5	174.8283	272.2759345	124	2767
P0 - P6	161.3636	233.9981011	122	2449
P0 - P7	202.4646	64.69146762	121	358
P0 - P8	129.4747	19.3086337	119	274

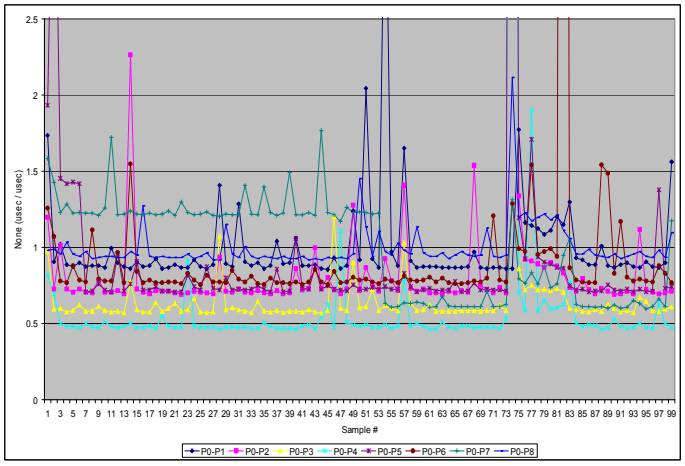


Figure 11: Normalized Curves for Ping-Pong-B- My-Cluster Using 9P.

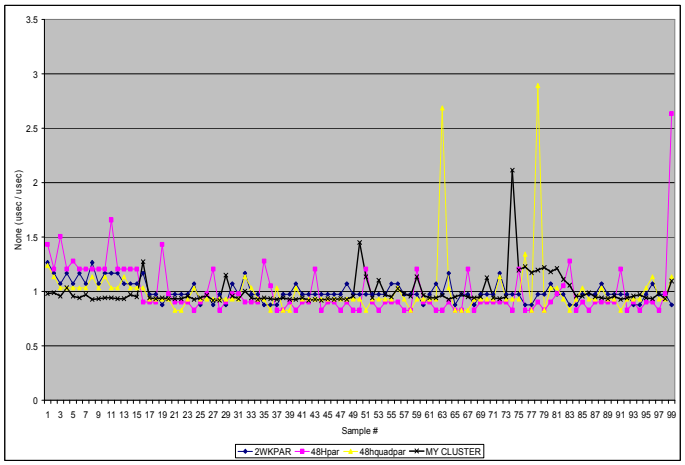


Figure 12: Normalized Ping-Pong B Curves for the pair (P0, P8)-Community-Cluster (all queues) vs My-Cluster

Comparing Figure 12 against Figures 9, 10 and 11, it is possible to conclude that My-Cluster does not exhibit the pulsing behavior observed in all three queues of the

Community-Cluster. After the comparison of these four figures, it is not only possible to conclude that My-Cluster presents a more steady behavior in terms of communication, but also that this type of characteristic will reduce the source of error due to the pulsing nature present on the Community-Cluster.

Since the pulsing behavior of the Community-Cluster has been observed in both ping pong tests, it can be speculated that some sort of background monitoring process for the cluster is being executed at regularly scheduled intervals, which may be causing the graphed timings to appear to pulse. Figure 12 still shows some outliers; however, it is possible to conclude that the communication pattern experienced by My-Cluster, is less susceptible to pulses. It is possible to observe regions where each one of the curves is almost flat for a considerable number of consecutive samples; this constitutes a clear indicator of fewer disturbances and interferences during the test. These conclusions and findings are supported and reinforced by Figure 12, where a comparison of the ping-Pong-B test between the pair of processes P0-P8 for My-Cluster and all queues of the Community-Cluster is illustrated.

5 Conclusions

All three queues of the Community-Cluster are prone to pulses of some nature that can negatively impact high performance applications where consecutive steady and accurate time readings are needed to statistically validate a phenomena under study.

The one-user cluster, referred to as My-Cluster, presents a much better behavior in terms of disturbances during communication between processes; the communication between processes is clearly more stable and steady for a considerable number of consecutive samples; as a consequence, it constitutes a more appropriate choice for time sensitive analysis.

The third and most important conclusion is derived from both ping pong tests, and it is associated with the fact that only the first time that a pair of processes establish a communication, a one-time high fee to pay in terms of time will exist. Consequently, it is possible to think that some sort of communication initialization takes place during this first message, and that such situation does not occur for all subsequent communications. This finding is supported by both ping pong tests, ping-pong-A and ping-pong-B, and occurs in both clusters even though they have different operating systems, hardware architecture and communication capabilities.

6 References

[1] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. "Optimizing Bandwith Limited Problems Using One-sided Communication and Overlap". *Parallel and Distributed Processing Symposium*, Apr. 29, 2006.

[2] E. Colmenares, P. Andersen, Y. Zhuang, "Overlapping Communication and Computation with MPI-2 for Floyd's Algorithm", MSCS Thesis, Texas Tech University, 2008.

[3] G. Glass, K. Ables, *Unix for Programmers and Users,* Pearson Prentice Hall, 2003.

[4] W. Groop and E. Lusk, *Reproducible Measurements of MPI Performance Characteristics,* Argonne National Laboratory, 1999.

[5] J. Hein, S. Booth and M. Bull, *Exchanging Multiple Messages via MPI*, HPCx Consortium, EPCC University of Edinburgh, 2003.

[6] F. Junior, *Configurando um Cluster No Fedora Core 4 com MPICH2*, Ministério Da Ciéncia e Tecnologia, Instituto Nacional De Pesquisas Espaciais. Sao Jośe dos Campos, Brasil, 2008.

[7] M. Machado, and M. Hofman, *Sistemas Distribuídos*, Pontifica Universidade Católica do Rio de Janeiro, Brasil, 2005.

[8] N. Nupairoj and Lionel M. Ni. (1994). Performance Evaluation of some MPI Implementations on workstation clusters. Available: http://citeseer.ist.psu.edu/7635.html

[9] Rice University - Information Technology. "Advanced Unix Scripts", Sep. 2003. Available: http://www-teaching.physics.ox.ac.uk/Unix+Prog/rice/pdf/unix18.pdf

Application-Aware Routing Policy based on application pattern traffic

Joe Carrión, Daniel Franco and Emilio Luque
Computer Architecture and
Operative Systems Department
Universitat Autónoma de Barcelona,
08193, Bellaterra, Spain
Email: joe.carrion@caos.uab.es, daniel.franco@uab.es, emilio.luque@uab.es

***Abstract*—Search engines are deployed over large datacenters and they support queries of thousands of users about a set of heterogeneous content, a typical configuration of a search engine includes three main components, a Front Service (FS) for user requests, Cache Service which is a subset of the most frequently used data and the full data set called Index Service (IS). Queries generate thousands of messages between nodes. The message delivery process should be performed efficiently. This paper is focused on improving the efficiency for allocating network resources. It is proposed the analysis of communication patterns of the system and the work balance of the network. The traffic pattern defines the behavior of each service and the workload on each router. The proposed mechanism makes decisions based on the historic behavior of the IS, FS, CS and the application pattern. The experimental results are evaluated using simulations techniques. Workloads from a real search engine are injected to the simulator and finally the results are compared with conventional routing mechanisms.**

Keywords*—Routing algorithm, interconnection networks, application-aware network.

I. Introduction

Search engines can be oriented for users all over the world and the content provided can be heterogeneous (general purpose search engine) we call them as Horizontal Search Engine (HSE). HSE are supported for a set of Vertical Search Engine (VSE). The VSE is a component of a HSE and they can be focused on specific content (commercial, scientific, cultural, etc.) for users from a delimited geographic area. Each VSE processes the requests of limited subset of users. Using VSE is an approach to balance the workload of the HSE.

On environments of VSE, the datacenter and network design have the capacity to scale from low to high periods of workload. The frequency of queries submitted is unpredictable and it changes from hundreds to thousands very quickly. This behavior throws an unexpected traffic to the network resources, hosts and routers. Therefore an efficient network resources allocation policy is desired.

Network protocols, devices and services have been designed with a huge set of configurable features, this approach allows extend the range of supported applications on expenses of performance and cost. However for datacenters focused on a reduced set of applications a specially designed datacenter and network is required.

Regarding to datacenter in [1], some best practices related to power efficiency are proposed. In [2] some guidelines to datacenter designs from the industry are proposed. Secondly, when the datacenter hosts a delimited set of applications, the network design should include an exhaustive traffic analysis to support the specific application hosted. The output should be an application-aware network (AAN). AAN is a mechanism "for boosting utilization of network resources based on customer demand" [3]. Since this point of view network design should be based on the hosted applications. The goal is a network based on the profile application. There are different parameters to draw the profile of an application. From literature we can mention [11], which includes number of terminals, latency, message size, traffic pattern and others. This information can be captured on runtime using monitoring standard protocols like sFlow and Netflow. They use a sampling mechanism [3] applied to the traffic network. The application monitoring process returns the delay of the messages of clients, servers and network devices, response time, number of new connections, bytes and packets submitted, packet lost and latency. Then the monitoring application generates very useful information for allocating network resources on demand.

In this paper we introduce a routing policy based on the application pattern analysis for a specific application using trace files from a real VSE. This policy computes the workload of the applications. A runtime monitoring process allows us balance the traffic and allocate network resources fairly. We evaluate the results with two standard network metrics: latency and throughput.

II. Related work

Application pattern analysis involves collecting traffic and data analysis. [4] analyzes the Intra-Rack and Extra-Rack traffic, link utilization and hot-spot behavior to create a profile of an application. [7] proposes a static mechanism based on a traffic analysis to identify the best routes between couples of nodes, after, they combine the best routes to create a set of new optimal routes. Furthermore network resources management techniques have been proposed, for instance [5] describes Generic-Adaptive-Resource-Control (GARC) as a control mechanism of connectivity to reduce the overall performance, GARC is a mediator between applications and network. Most specific techniques like routing policies allow load balance, [6] introduces Application-Specific Routing Algorithm (APSRA) to model the application using graphs, and

the output is a static routing table. The application pattern has impact on the network status [11], so adaptative routing algorithms aim to make decisions based on network status. [8] introduces Distributed Routing Balancing (DRB) which uses an algorithm based on maximize the use of resources by creating new paths between nodes and minimizing the path-length, the output is a fairly message distribution. In [9] PR-DRB (Predictive DRB) extends to real applications by monitoring the best paths and storing them to make routing decisions based on alternative paths.

III. Application Pattern Analysis

A. Background

We conduct our research over a VSE. The main software components of a VSE are: Front Service (FS), Cache Service (CS) and Index Service (IS) deployed on a Fat-tree topology [13].

The services of the VSE are deployed on a large cluster of computers. The nodes are arranged on arrays of P x D, where P define level of data partitioning and D the level of replication of the data, a full description of the architecture is published in [13], we show a basic overview on figure 1 and we remark the following terminology useful for this paper: FS as Front Service, IS as Index Service, ISR as Index Service Replica, CS as Cache Service and CSR as Cache Service Replica.

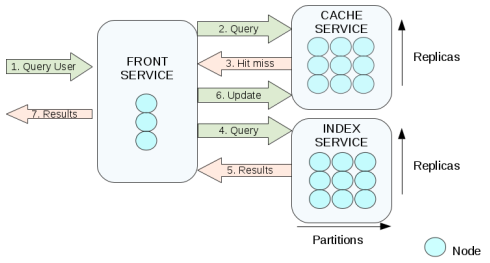


Fig. 1. Overview of traffic flow between the main components of the VSE.

B. Communication pattern

VSE communication pattern is defined for the messages between nodes. The load of the system depends on two elements, the volume of user requests and the size of the content stored in the system. The volume of users submitting queries in a period of time is unpredictable. This rate of input triggers an unstable communication pattern if we compare it with traditional synthetic traffic patterns used to evaluate network performance.

Let N a Fattree topology network with 64 nodes. Axis Y represents the range of sources and destinations and axis X is used for time. We can see the trend of source and destination and the distribution of each couple. The traffic pattern is defined by a couple of Sender and Receiver (S-R).

On figures 2 and 3, we depict a set of samples of synthetic traffic from literature [11]. Synthetic traffic patterns used are Uniform (UTP) and Shuffle (STP). For synthetic traffic we are using 64 nodes to represent plainly instead of 128 nodes of the real VSE network.

Using UTP the Sender in predictable through the time, in contrast Receiver is randomly selected. Figure 2 shows that traffic is distributed in a large set of nodes as source (from node 0 to 64) and the destination goes in the same interval. The result is a distributed traffic over all network. Although the pairs S-R are unpredictable, the traffic goes through a big range of network resources. Then we conclude traffic is distributed between a big set of resources.

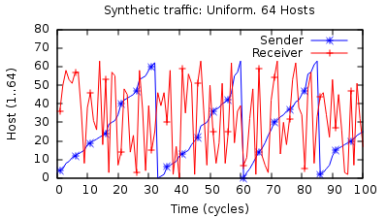


Fig. 2. Uniform Traffic Pattern.

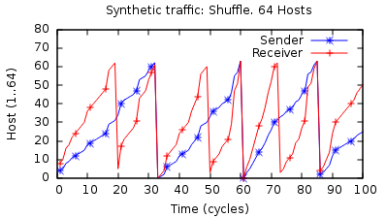


Fig. 3. Shuffle Traffic Pattern.

With STP the S-R pairs are predictable because R is computed from S. The traffic pattern goes from lower nodes to the highest. The result is a traffic distributed on specific part of the network. The used region of the network moves uniformly.

We represent the traffic of a VSE using a trace file from a VSE on figure 4. We call this traffic VSETP (VSE traffic pattern). Although the couples S-R are unpredictable through the time, they move over a specific set of nodes. VSETP is distributed around a reduced set of nodes through the time. For instance on figure 5, axis X shows the time, interval is 25 to 35, senders are less than 10 and receivers are from 30 to 50. The same pattern appears on range 55 to 65 and 115 to 125. The range of nodes define locality and time define frequency (or repetition). Figure 6 shows the repetition and locality for receiver. Periods of time are defined from 92 to 102, 112 to 122 and 145 to 165. Figure 7 shows the trend for using a configuration of 256 nodes. The locality is defined for senders from 0 to 50 and receivers from 150 to 200 when time goes from 20 to 40. This trend is repeated on period 90 to 110.

The traffic of those periods of time can be managed by a specific policy taking into account the repeatability and locality.

C. Traffic Flow

User queries are accepted by the FS, it submits the queries to CS. If the query has been performed previously the query has been cached by the CS (based on a cache policy), thereby CS checks these conditions and returns the output of the query (cache hit) otherwise when the output is not cached, the output is a cache failure. FS redirects the unsuccessfully queries to

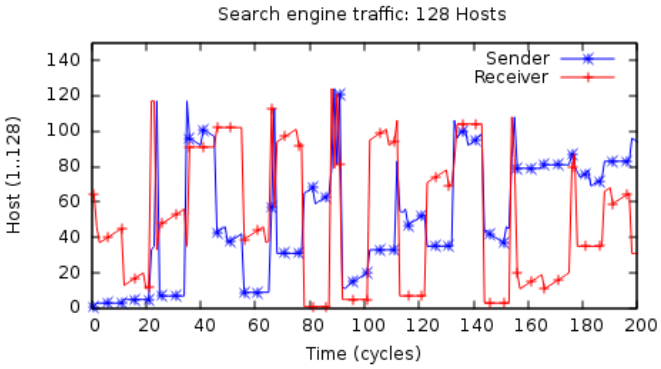


Fig. 4. VSE traffic pattern. Trend of each S-R pair with 128 nodes.

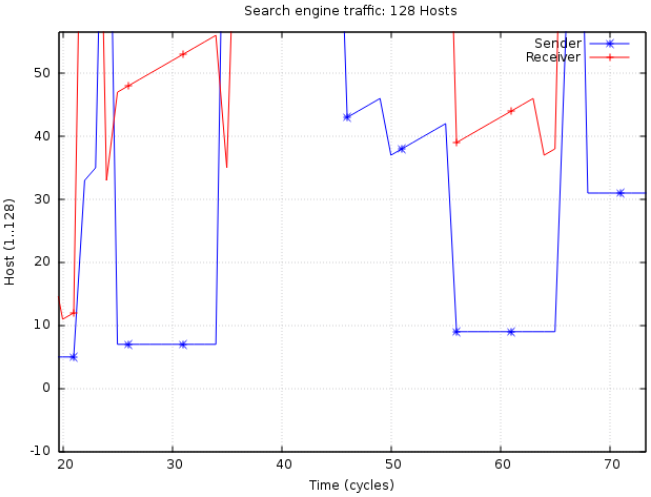


Fig. 5. Trend of repetition and locality for Sender with 128 nodes.

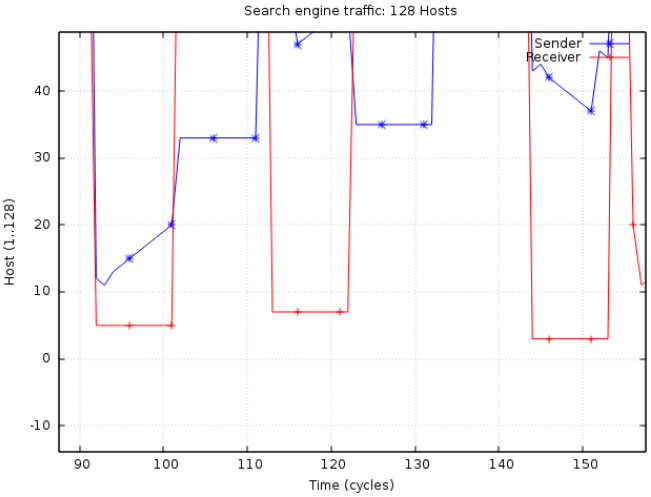


Fig. 6. Trend of repetition and locality for Receiver pair with 128 nodes.

IS, which will create a top-k results to send back to FS. The detailed business rules about a VSE are out of the scope of this paper, a full description is published in [13].

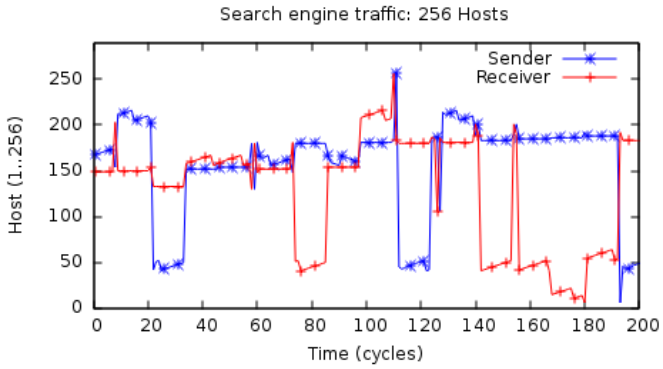


Fig. 7. VSE traffic pattern. Trend of each S-R pair with 256 nodes.

D. Workload analysis

The workload analysis needs a real trace file. We take two networks with 128 and 256 nodes. Firstly we evaluate the workload for the whole system. To depict the load of the network we take the traffic generated by the FS as source (sender) and the destination (receiver) the remaining components (IS, ISR, CS and CSR). The trace file is generated after processing 100000 queries.

On figure 8 axis Y shows the number of messages. We show the workload of each service thought the time in axis X. We can see the load is predominated by ISR, a closer view shows that IS traffic is proportional to traffic ISR; also CS and CSR keep almost constant. Figure 9 shows the workload for a network with 256 node and we can see that ISR traffic behavior prevails.

Fig. 8. Workload with 128 hosts.

There are periods of time with more messages in the network and there are periods with a low number of messages; for example in figure 10, we show the 128-nodes network to the period of time from 50 to 60. The number of messages by unit time is more than 6000 messages. So the injection rate is very high compared to the period of time from 65 to 70.

The variance of the number of messages causes a variance of the injection rate; the application traffic pattern requires proportional allocation resources accordingly to the traffic of each service. There are more messages from ISR compared to other services.

We use this behavior to allocate network resources using a routing policy to balance the workload. The first approach is focused on allocate routers buffers according to the service.

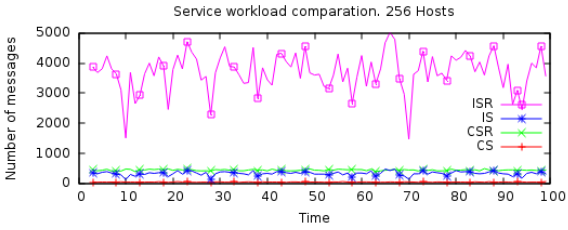


Fig. 9. Workload with 256 hosts

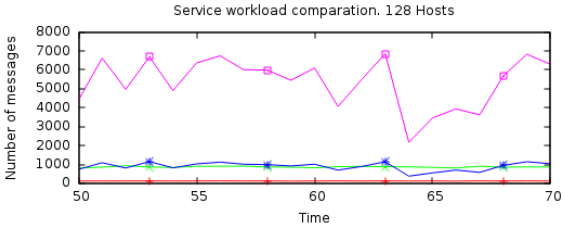


Fig. 10. Changes of the workload by kind of service of the VSE traffic.

E. Buffer Occupancy Analysis

We analyze the workload over the network; this paper is focused on the buffer of input channels (B). Taking into account each message is divided in packets to go from a source to a destination. When a packet (m) arrives to a router (R), it is allocated into an input buffer to wait for an output channel.

Router buffers have a size (BS) and the size should be computed accordingly to workload application (WL). Buffers contain a set of packets, the number of packets define the buffer occupancy (BO). BO goes from 0 to BS. We define 3 thresholds for BO. If BO goes from 0 to 25% of BS we call BO low (BOL), Medium BO (BOM) when BO goes greater than 25% to 50% and High BO (BOH) for BO higher than 50%. The higher BO the more congested is the channel.

An overview of BO for a typical network configuration using UTP is showed in Figure 11. The network configuration used is a Fat-tree topology, with k=10 and n=3, then we have routers arranged in 3 levels (Level 0, 1 and 2, Level 0 is the root of the tree). There are 300 routers, and we have more than 1000 B. Axis X shows the buffer routers. There are 3 groups of bars, one for each level of routers.

Figure 12 zooms only from the 900 to 1100 BO. The size of the bars represents the number of events that BO was higher than LBO. Therefore for UTP the workload is distributed over all routers. The most congested B belongs to routers of Level 2, and the less congested B belongs to Level 0. This distributed traffic is desired for real traffic patterns. However this tendency does not appear on the pattern generated for a VSE.

We apply the same analysis for VSETP and we can see an unbalanced behavior. Figure 13 draws BO for the same threshold LBO. BO is higher than 25% in most of the routers, but there are some buffers without occupancy. Also there are events when BO is higher than 50%. Figure 14 show the behavior for HBO.

Figure 13, shows that workload distribution is not balanced, on Level 2, the occupancy in concentrated in a reduced set of

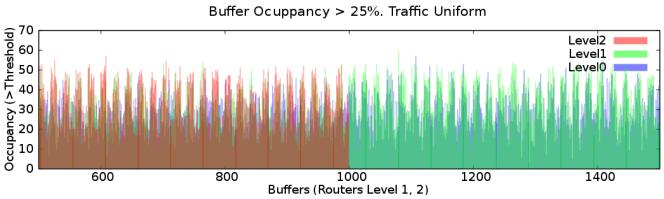


Fig. 11. Full buffer occupancy using Uniform synthetic traffic.

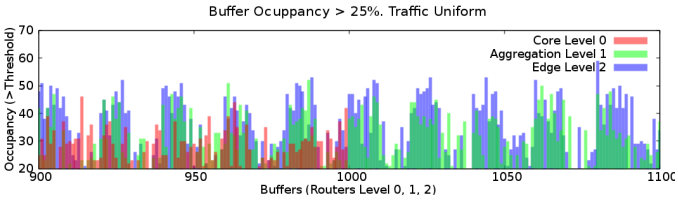


Fig. 12. Buffer occupancy using Uniform synthetic traffic.

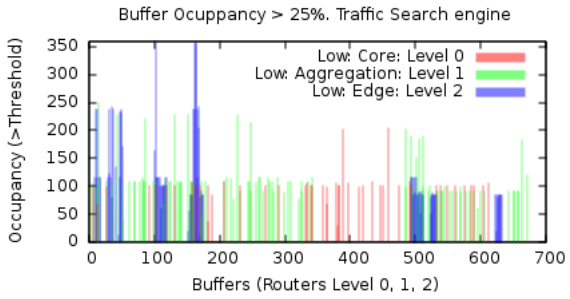


Fig. 13. Buffer occupancy using traffic in the VS with Threshold 25%.

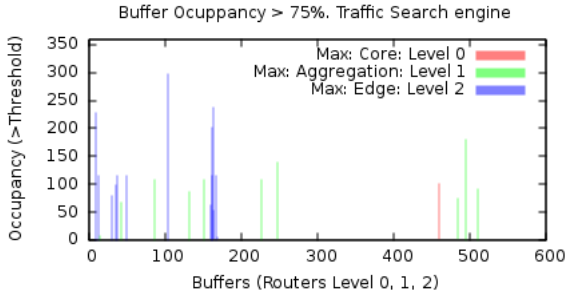


Fig. 14. Buffer occupancy using traffic in the VS with Threshold 75%.

routers. Some routers do not get the threshold for LBO. Level 1 also has the same unbalanced behavior although, it is less than Level 1 and finally on Level 0 the unfair workload prevails.

A routing policy should allocate buffers accordingly to the load of an application. We know the workload application for each service, then we can allocate routers based on each service. In order to detect the kind of service it is necessary identify the sender and receiver; the routing policy should redirect the messages based on workload and BO.

F. Host-Destination Analysis

This analysis is based on then traffic flow described on Section III-C. Each node only keeps contact with a delimited set of services, for instance a CS node submits messages to a

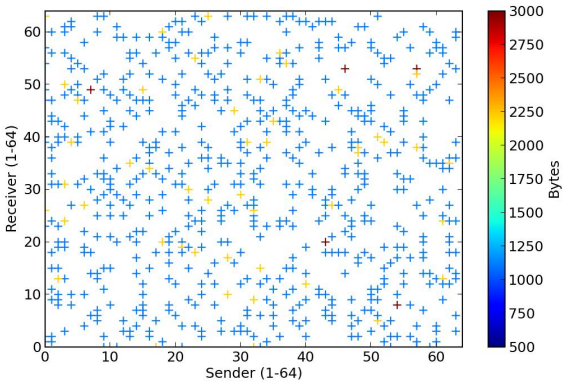


Fig. 15. Pairs SR using Uniform Synthetic traffic.

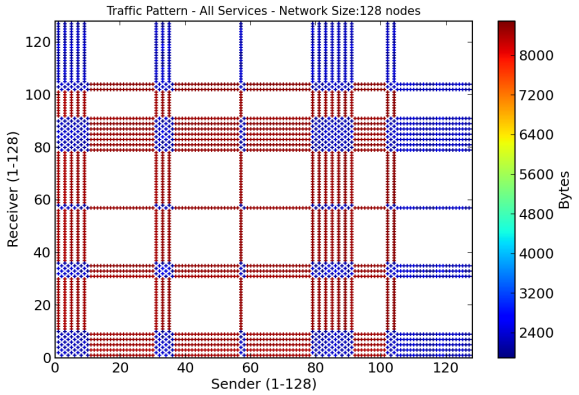


Fig. 16. SR using Flow-traffic conditions of the VSE.

FS node, and a IS node submits messages only to a FS node. With algorithms to generate synthetic traffic the behavior can be predictable, for random generation the set of couples is unpredictable, see figure 15. Each pair S-R appears randomly distributed on space. However using the Flow-traffic conditions of the VSE the set of couples is deterministic, figure 16 shows that each pair S-R is created with a specific set of nodes. For instance there is not traffic from node source 20 to node destination 20 and 40 (source) to 40 (destination). We have a well-defined empty area.

The tendency reduces the set of couples and it creates an unbalanced traffic. Areas with traffic are much defined and areas without traffic can result in network resources unused. On one hand if there is resources unused the network could be reduced, on the other hand there is buffers overloaded.

IV. Application-aware Routing Policy

Based on analysis presented in previous section, we present a routing policy called Application-Aware Routing (AAR) which allocates network resources based on application profile.

AAR can be outlined as follows. Three main components are introduced into router architecture. First one is the Buffer occupancy monitor (BOM), the second one is the Deep Packet Inspection (DPI) mechanism and finally a Decision Maker (DM). Figure 17 shows an overview of the components.

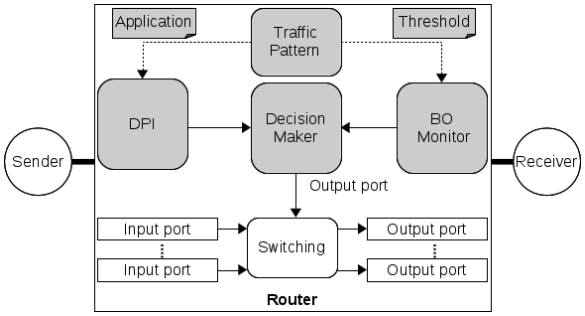


Fig. 17. Overview of the AAR.

BOM keeps historical information about buffer occupancy. DPI identifies if a packet belongs to a specific service. The DM based on information of BOM and DPI redirects the traffic by allocating output channels. Packets belonging to IS or ISR are redirected to output ports with less BO. This approach allow us allocate resources proportionally to service demand. The workload is distributed toward less used network area.

A. Buffer occupancy monitor

BO tracks the buffer occupancy on runtime. This tracking process is based on three thresholds. High, medium and low occupancy (BOH, BOM and BOL respectively). When a packet arrives it is allocated a buffer. There is a BO counter for each buffer, which is updated when it reaches a threshold. These three levels allow us tune the policy accordingly to the profile application. For instance in Section III-D we conclude that VSETP is predominated by the ISR. We configure the routing policy with BOH to apply the policy to redistribute the traffic of the service with higher workload. The next step involves a detection service action.

B. Deep Packet Inspection and Decision Maker

This component takes as input two parameters, a mapping of hosts-services and application profile based on the workload. When a packet arrives to the router the policy merges the mapping table and the workload. The output is a candidate packet to be redirected. This packet is passed to the DM. The remaining packets are redirected applying the default routing policy of the system.

Taking as input the BOM and the candidate packet, the packet is redirected based on the BOM. BOM sends the packet to the output port with lower occupancy.

V. Experimentation

We evaluate AAR using a modified version of Booksim simulator published in [10]. We present the results comparing AAR with conventional Nearest Common Ancestor (NCA) introduced in [11]. The network is configured with the same characteristics of the real VSE. So we use 128 and 256 nodes arranged in a Fat-tree topology of tree levels. The result are analysed using two standard metrics for networks: Average Network latency (ANL) and Accepted Packet Rate (APR).

Firstly we focus on the overall AAR and NCA comparison with VSE traffic pattern. It takes the whole period of time of simulation.

On figure 18 we depict the ANL. Axis X is the time, (in K-cycles of simulation). Axis Y is the number cycles need to deliver a packet. The first overview shows the curve of AAR under the curve of NCA almost in the full period of time. This tendency exemplifies a decrease of the network latency using AAR.

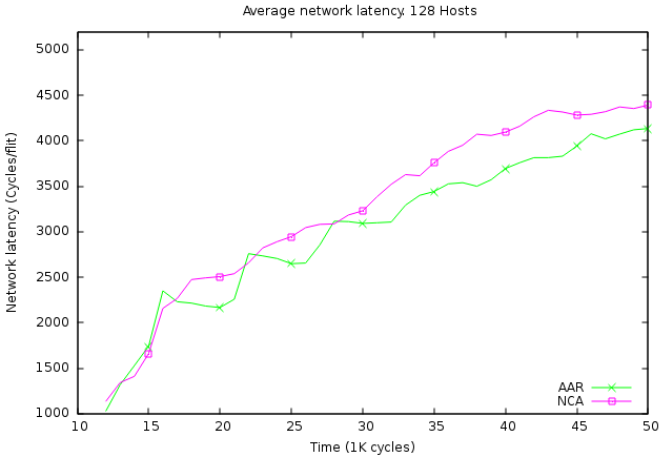


Fig. 18. AAR and NCA using 128 nodes and traffic pattern of VSE

In table I we show that network latency (NL) is 3339 K-cycles using NCA, we take this value as baseline and we compare the result of AAR. AAR gets 3119 Kcycles. Then AAR reduces the network latency in 7.26%.

TABLE I. COMPARISON BETWEEN AAR AND NCA. NODES:128. TRAFFIC PATTERN: VSE

Metric	AAR	NCA
Average network latency	3119	3339
Percentage of decrease	92.73%	100%

Regarding to the analysis of APR, on figure 19 we can see the curves of APR using two routing algorithms. Axis X is the time and axis Y is the number of messages delivered on each unit time. The AAR curve is over the NCA. It shows that AAR delivers more packets on each unit time against NCA.

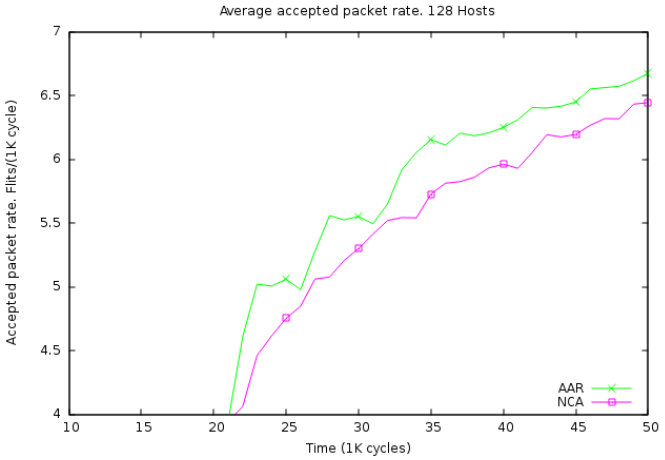


Fig. 19. AAR and NCA with 128 nodes for traffic pattern of VSE

Table II shows APR, using NCA is 4.91 packets accepted by each K-cycle. Using AAR the APR grows up to 5.12. This increase represents the 4.4%.

TABLE II. SUMMARY OF COMPARISON BETWEEN AAR AND NCA. NODES: 128. TRAFFIC PATTERN: VSE

Metric	AAR	NCA
Average accepted packet rate	5.12	4.91
Percentage of increase	104.4%	100%

Now we analyze the behavior through the time. If we divide vertically the curves of ANL and APR in two areas, the part of the right shows that AAR and NCA are overlapped when the time is lower than 25. Also, the ANL is less than 3000K cycles. However, this tendency change later if we see the left part, when the time is higher than 25. The difference between two curves increases and then we compare the routing algorithms after a period of time.

At the beginning there is not information about the buffer occupancy. Then AAR performance is the same as NCA. When AAR has enough information about buffer occupancy, its performance improves against NCA. NCA allocates network resources in a deterministic way. On the other hand AAR takes advantage of the historic information.

On table III we show the results when AAR have information about BO. We compare the performance when the latency is higher than 3000 Kcycles. After this period of time AAR has collected information about buffer occupancy then the policy is applied for more packets compared to the beginning of network operation.

The final results show that AAR improves latency on 7.93% and the accepted packet rate increases in 4.78%.

TABLE III. SUMMARY OF COMPARISON BETWEEN AAR AND NCA. NODES: 128. TRAFFIC PATTERN: VSE. AFTER WARM UP PHASE

Metric	AAR	NCA
Average network latency	3662	3978
Percentage of decrease	92.06%	100%

TABLE IV. SUMMARY OF COMPARISON BETWEEN AAR AND NCA. NODES: 128. TRAFFIC PATTERN: VSE. AFTER WARM UP PHASE

Metric	AAR	NCA
Average accepted packet rate	6.21	5.92
Percentage of increase	104.78%	100%

Next set of experiments were carry out using a configuration for 256 nodes. The workload is a VSETP. Figure 20 shows the latency. The AAR curves goes over the NCA curve.

As we can see in table V network latency (NL) is 5213 Kcycles using NCA and we compare the result of AAR. AAR gets 4996 Kcycles. Then AAR reduces the network latency in 4.16%.

TABLE V. SUMMARY OF COMPARISON BETWEEN AAR AND NCA. NODES: 256. TRAFFIC PATTERN: VSE.

Metric	AAR	NCA
Average network latency	4996	5213
Percentage of decrease	95.83%	100%

The APR is showed on figure 21. The period of time less than 10 show the warm-up period when there is not traffic.

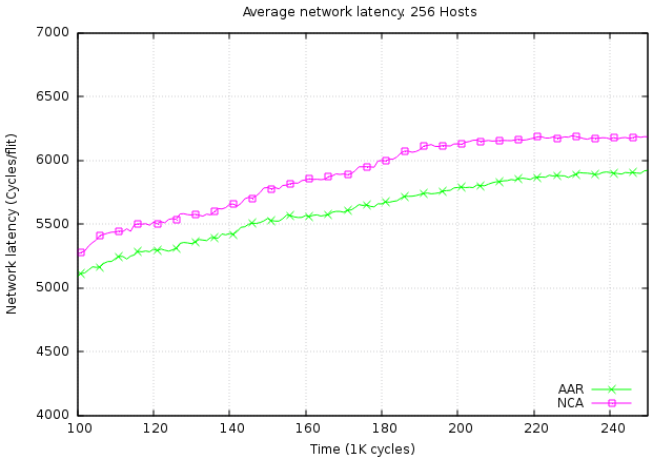


Fig. 20. AAR and NCA with 128 nodes for traffic pattern of VSE

From period 10 to 50, NCA have AAR have almost the same performance. AAR needs information about buffer occupancy to distribute the packets, after collecting information the performance provided by AAR increases.

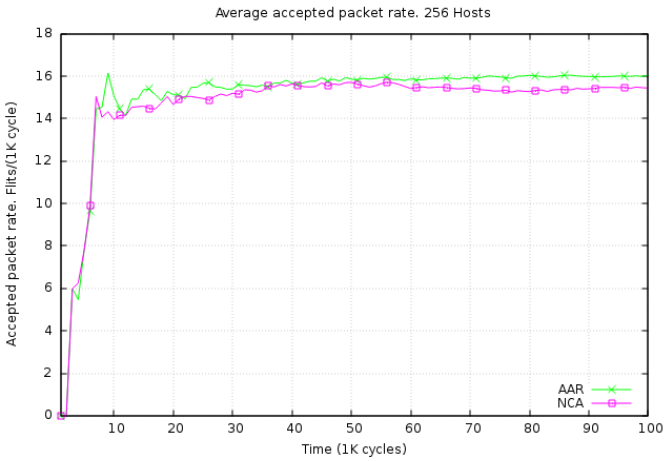


Fig. 21. AAR and NCA with 256 nodes for traffic pattern of VSE

Table V shows that APR is 14.96 packets using NCA, and the result using AAR is 15.54 packets. Then AAR improves the throughput in 3.92%.

TABLE VI. SUMMARY OF AAR AND NCA COMPARISON. NODES: 256. TRAFFIC PATTERN: VSE

Metric	AAR	NCA
Average accepted packet rate	15.54	14.96
Percentage of increase	103.92%	100%

The set of experiments demonstrates that AAR delivers more packets while it reduces the network latency. The performance prevails for networks with 128 and 256 nodes.

VI. CONCLUSION

We have proposed the Application-Aware Routing policy, AAR. This policy is based on the analysis of an application pattern of a vertical search engine. The algorithm keeps historical information about buffer occupancy and the application profile. The network resources are allocated on application demand. The performance of AAR is analyzed with two standard network metrics: latency and throughput. Experiments show AAR improves the network performance by reducing the latency and delivering more packets in the same period of time. Future work is oriented to extend the analysis to other network components and test our AAR with different topologies.

ACKNOWLEDGMENT

This research has been supported by : MINECO (MICINN) Spain under contract TIN2011-24384, SENESCYT[1] Ecuador government under contract 2013-AR7L335.

Authors would like to thank to Veronica Gil-Costa, Mauricio Marin and Yahoo! Research Latin America.

REFERENCES

[1] Greenberg, et al,*Best Practices for Data Centers: Lessons Learned from Benchmarking 22 Data Centers*, Summer Study on Energy Efficiency in Buildings, ACEEE 2006, pp. 7687.

[2] Cisco Systems, Inc. 2007, Cisco Data Center Infrastructure 2.5 Design Guide. http://www.cisco.com

[3] MRV,*Application-Aware Networking at A Glance*, White Paper 2013, http://www.mrv.com.

[4] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. *Network traffic characteristics of data centers in the wild*. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10). ACM, New York, NY, USA, 267-280. DOI=10.1145/1879141.1879175 http://doi.acm.org/10.1145/1879141.1879175

[5] J. Mueller, T. Magedanz. *Towards a Generic Application Aware Network Resource Control Function for Next-Generation-Networks and Beyond*. In *IEEE* Proceedings of the International Symposium on Communications and Information Technologies (ISCIT 2012), pp. 777882.

[6] Palesi, M., Holsmark, R., Kumar, S. Catania, V. *Application Specific Routing Algorithms for Networks on Chip*. Parallel and Distributed Systems, IEEE Transactions on 20, 316-330 (2009).

[7] N. Michael, M. Nikolov, A. Tang, G. E. Suh, C. Batten, Proceedings of the Fifth ACM/IEEE International Symposium , 9-16 (2011).

[8] D. Franco, I. Garcés, and E. Luque. 1999. *A new method to make communication latency uniform: distributed routing balancing*, ICS '99, ACM, New York, NY, USA, 210-219, http://doi.acm.org/10.1145/305138.305195

[9] Carlos Nunez Castillo, Diego Lugones, Daniel Franco, Emilio Luque, Martin Collier: 2013. *Predictive and Distributed Routing Balancing, an Application-Aware Approach*, ICCS 2013, 179-188.

[10] Nan Jiang Becker, D.U. ; Michelogiannakis, G. ; Balfour, J. ; Towles, B. ; Shaw, D.E. ; Kim, J. ; Dally, W.J.. 2013. *A detailed and flexible cycle-accurate Network-on-Chip simulator*. ISPASS 2013: 86-96.

[11] William Dally, Bryan Twles. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, ELSEVIER, San Francisco, p.550.

[12] George Michelogiannakis, Daniel Becker, Brian Towles and William J. Dally. N Jiang. 2013. *BookSim 2.0 User's Guide*. https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim.

[13] Jair Lobos, Veronica Gil-Costa, and Mauricio Marin, Alonso Inostrosa-Psijas. 2012. *Capacity Planning for Vertical Search Engines: An Approach Based on Coloured Petri Nets. Yahoo! Research Latin America.*

[14] Alexander Loukissas, Amin Vahdat Mohammad Al-Fares. 2008. *A Scalable, Commodity Data Center Network Architecture*. SIGCOMM'08, August 17-22, p. 12

[1]SENESCYT: Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación, http://www.senescyt.gob.ec/

An Algorithm for Node-to-Node Disjoint Paths Problem in a Möbius Cube

David Kocík†, Yuki Hirai‡, and Keiichi Kaneko‡
† Faculty of Information Technology, Czech Technical University in Prague, Thákurova, Prague, Czech Republic
‡ Institute of Engineering, Tokyo University of Agriculture and Technology, Koganei-shi, Tokyo, Japan

Abstract— *In this paper, we propose an algorithm that solves the node-to-node disjoint paths problem in n-möbius cubes in polynomial-order time of n. We also give a proof of its correctness as well as the estimates of time complexity $O(n^3)$ and the maximum path length $O(n^2)$. We conducted a computer experiment for $n = 4$ to 31 to measure the average performance of our algorithm. The results show that the average time complexity is $O(n^{1.5})$ and the maximum path lengths on average are linear to n.*

Keywords: container problem, hypercube, multicomputer, interconnection network, parallel processing

1. Introduction

Recently, researches on parallel processing, especially massively parallel systems are enthusiastically performed. A lot of nodes are connected in a massively parallel system. To interconnect them efficiently, many topologies for interconnection networks have been proposed [1], [9] and studied [2], [3], [4], [5], [13], [14], [15], [24] to replace simple interconnection networks such as a ring, a mesh, a torus, and a hypercube [25]. A möbius cube [10] is one such new topology. It has attracted much attention because it can connect the same number of nodes as a hypercube while keeping its diameter about half of that of the hypercube [12], [20], [27], [28], [29].

The unsolved problems in möbius cubes include the node-to-node disjoint paths problem: given a source node $\boldsymbol{s}$ and a destination node $\boldsymbol{d}$ in a k-connected graph $G = (V, E)$, find k paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ that are node-disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The node-to-node disjoint paths problem is an important issue in parallel and distributed computation [11], [19], [23], [26] as well as the node-to-set disjoint paths problem [7], [17], [22], [21] and the set-to-set disjoint paths problem [6], [8], [16], [18].

For a graph $G(V, E)$, by using the maximum flow algorithm, the node-to-node disjoint paths can be obtained in polynomial-order time of $|V|$ in general. However, the complexity of the algorithm is too large for an n-dimensional möbius cube or an M_n in short because the number of nodes in it is equal to 2^n. In this paper, we propose an algorithm called N2N (node-to-node) which has a polynomial-order time of n instead of 2^n. Algorithm N2N consists of four cases according to the relative positions of the source node and the destination node. The algorithm obtains n disjoint paths from the source node $\boldsymbol{s}$ to the destination node $\boldsymbol{d}$ where n is equal to the connectivity of an M_n. We also present the results of an average performance evaluation by a computer experiment.

The rest of this paper is organized as follows. Section 2 introduces the definition of möbius cubes as well as other requisite definitions. Section 3 explains our algorithm N2N in detail. Section 4 describes a proof of correctness and the theoretical complexities of N2N. Average performance of N2N is reported in Section 5. We conclude and give future works in Section 6.

2. Preliminaries

In this section, we first introduce a definition of a möbius cube followed by several lemmas.

Definition *1:* An n-dimensional möbius cube M_n has 2^n nodes. Each node has a unique n-bit address. For two nodes $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{y}$, they are connected if and only if one of the following conditions are satisfied:

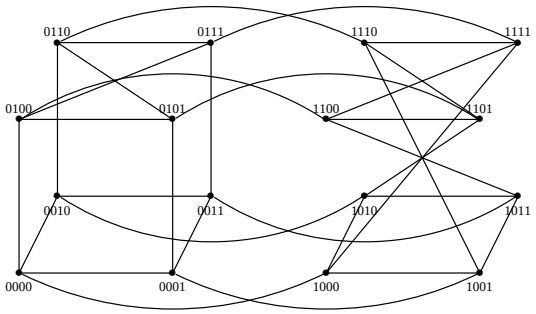
$$\boldsymbol{y} = \begin{cases} (x_1, x_2, \ldots, x_{i-1}, \bar{x}_i, x_{i+1}, \ldots, x_n) & \text{if } x_{i-1} = 0, \\ (x_1, x_2, \ldots, x_{i-1}, \bar{x}_i, \bar{x}_{i+1}, \ldots, \bar{x}_n) & \text{if } x_{i-1} = 1. \end{cases}$$

where we can assume that $x_0 = 0$ or $x_0 = 1$. For the former case, we call it a 0-M_n while for the latter a 1-M_n.

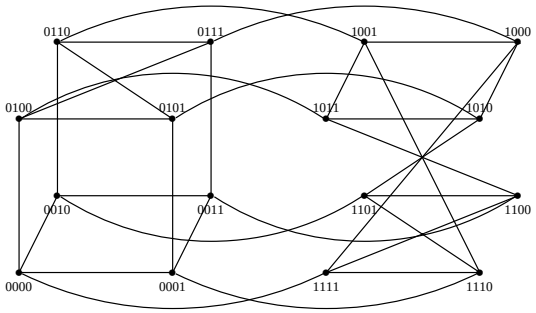
If two nodes $\boldsymbol{x}$ and $\boldsymbol{y}$ are connected by one of the conditions in Definition 1, we say that $\boldsymbol{x}$ and $\boldsymbol{y}$ are connected by an edge of i-th dimension, and $\boldsymbol{y}$ is denoted by $\boldsymbol{x}^{(i)}$ or $\boldsymbol{x}$ is denoted by $\boldsymbol{y}^{(i)}$.

Figure 1 shows examples of a 0-M_4 and a 1-M_4. An M_n consists of two disjoint subgraphs M^0 and M^1 where M^i is derived from the set of nodes $\{\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \mid x_1 = i\}$. Note that an M^0 and an M^1 are isomorphic to a 0-M_{n-1} and 1-M_{n-1}, respectively.

Table 1 shows a comparison of characteristics of an n-dimensional 0-möbius cube, 0-M_n, and an n-dimensional 1-möbius cube, 1-M_n, with an n-dimensional hypercube, H_n and an n-dimensional twisted hypercube, T_n. With respect to the diameter, a T_n is a slightly better than 0-M_n. However, a T_n is much inferior to a 0-M_n and a 1-M_n with respect to the average distance.



(a) 0-M_4



(b) 1-M_4

Fig. 1: Examples of a 0-M_4 and a 1-M_4.

Table 1: Comparison of a 0-M_n and a 1-M_n with other topologies.

	#nodes	degree	diameter	average distance
0-M_n	2^n	n	$\lceil (n+2)/2 \rceil$	†
1-M_n	2^n	n	$\lceil (n+1)/2 \rceil$	†
H_n	2^n	n	n	$n/2$
T_n	2^n	n	$\lfloor (n+1)/2 \rfloor$	$\rightarrow 3n/8$ $(n \rightarrow \infty)$

†: $\leq n/3 + [1-(-1/2)^n]/9 + 1$ [10]

There is a shortest-path routing algorithm for an M_n and it takes $O(n)$ time [10]. In the rest of this paper, we refer the algorithm `spr`.

We assume that each node is stored in a machine word, and construction of an edge by obtaining $\boldsymbol{a}^{(i)}$ for any node $\boldsymbol{a}$ requires $O(1)$ time. On the other hand, `spr` takes $O(n)$ execution time to construct a shortest path whose length is at most $\lceil (n+2)/2 \rceil$ [10].

Lemma *1:* For an arbitrary node $\boldsymbol{a}$ in a 0-M_n, we can construct $(n-1)$ paths Q_i: $\boldsymbol{a}^{(i)} \rightsquigarrow \boldsymbol{a}^{(1)}$ $(2 \leq i \leq n)$ that are disjoint except for $\boldsymbol{a}^{(1)}$. The time complexity for construction of these paths is $O(n)$. The lengths of the paths are at most $\lceil n/2 \rceil + 2$.

(Proof) Let us consider the following four cases.

Case 1 ($3 \leq i \leq n$, $a_{i-1} = 0$) Construct a path Q_i: $\boldsymbol{a}^{(i)} = (a_1, a_2, \ldots, a_{i-1}, \bar{a}_i, a_{i+1}, \ldots, a_n) \rightarrow \boldsymbol{a}^{(i,1)} = (\bar{a}_1, a_2, \ldots, a_{i-1}, \bar{a}_i, a_{i+1}, \ldots, a_n) = \boldsymbol{a}^{(1,i)} \rightarrow \boldsymbol{a}^{(1)}$.

Case 2 ($3 \leq i \leq n$, $a_{i-1} = 1$) Construct a path Q_i: $\boldsymbol{a}^{(i)} = (a_1, a_2, \ldots, a_{i-1}, \bar{a}_i, \bar{a}_{i+1}, \ldots, \bar{a}_n) \rightarrow \boldsymbol{a}^{(i,1)} = (\bar{a}_1, a_2, \ldots, a_{i-1}, \bar{a}_i, \bar{a}_{i+1}, \ldots, \bar{a}_n) = \boldsymbol{a}^{(1,i)} \rightarrow \boldsymbol{a}^{(1)}$.

Case 3 ($i = 2$, $a_1 = 0$) Select an edge $\boldsymbol{a}^{(2)} = (a_1, \bar{a}_2, a_3, \ldots, a_n) \rightarrow \boldsymbol{a}^{(2,1)} = (\bar{a}_1, \bar{a}_2, a_3, \ldots, a_n)$. In the $\bar{a}_2$-M_{n-2}, construct a shortest path $\boldsymbol{a}^{(2,1)} \rightsquigarrow \boldsymbol{a}^{(1,2)} = (\bar{a}_1, \bar{a}_2, \bar{a}_3, \ldots, \bar{a}_n)$ by `spr`. Select an edge $\boldsymbol{a}^{(1,2)} \rightarrow \boldsymbol{a}^{(1)}$.

Case 4 ($i = 2$, $a_1 = 1$) Select an edge $\boldsymbol{a}^{(2)} = (a_1, \bar{a}_2, \bar{a}_3, \ldots, \bar{a}_n) \rightarrow \boldsymbol{a}^{(2,1)} = (\bar{a}_1, \bar{a}_2, \bar{a}_3, \ldots, \bar{a}_n)$. In the $\bar{a}_2$-M_{n-2}, construct a shortest path $\boldsymbol{a}^{(2,1)} \rightsquigarrow \boldsymbol{a}^{(1,2)} = (\bar{a}_1, \bar{a}_2, a_3, \ldots, a_n)$ by `spr`. Select an edge $\boldsymbol{a}^{(1,2)} \rightarrow \boldsymbol{a}^{(1)}$.

Then, the $(n-2)$ paths constructed in Case 1 or Case 2 are Q_i: $\boldsymbol{a}^{(i)} \rightarrow \boldsymbol{a}^{(1,i)} \rightarrow \boldsymbol{a}^{(1)}$ $(3 \leq i \leq n)$. Hence, they are disjoint except for $\boldsymbol{a}^{(1)}$. The path constructed in Case 3 or Case 4 is Q_2: $\boldsymbol{a}^{(2)} \rightarrow \boldsymbol{a}^{(2,1)} \rightsquigarrow \boldsymbol{a}^{(1,2)} \rightarrow \boldsymbol{a}^{(1)}$. Since its sub path $\boldsymbol{a}^{(2,1)} \rightsquigarrow \boldsymbol{a}^{(1,2)}$ is included in the $\bar{a}_2$-M_{n-2} and the internal nodes of Q_2 are all in the form of $(\bar{a}_1, \bar{a}_2, \ldots)$, Q_2 is disjoint with other paths Q_i $(3 \leq i \leq n)$.

It takes $O(n)$ time to construct the $(n-2)$ paths of length 2 in Case 1 or Case 2. Also, it takes $O(n)$ to construct the path in Case 3 or Case 4 since `spr` takes $O(n)$ time to construct a path. In total, the $(n-1)$ paths are constructed in $O(n)$ time.

The lengths of the paths constructed in Case 1 or Case 2 are all 2. The path length is at most $1 + \lceil n/2 \rceil + 1 = \lceil n/2 \rceil + 2$. See Figure 2. □

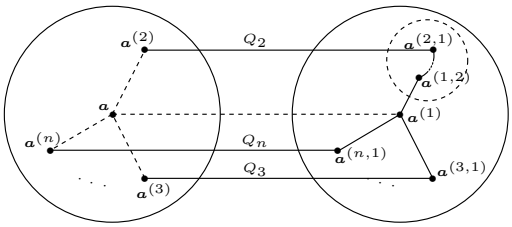


Fig. 2: Disjoint paths from $\boldsymbol{a}^{(i)}$ to $\boldsymbol{a}^{(1)}$ $(2 \leq i \leq n)$ in a 0-M_n

Lemma *2:* For an arbitrary node $\boldsymbol{a}$ in a 1-M_n, we can construct $(n-1)$ paths Q_i: $\boldsymbol{a}^{(i)} \rightsquigarrow \boldsymbol{a}^{(1)}$ $(2 \leq i \leq n)$ that are disjoint except for $\boldsymbol{a}^{(1)}$. The time complexity for construction of these paths is $O(n^2)$. The lengths of the paths are at most $\lceil n/2 \rceil + 2$.

(Proof) Let us consider the following two cases.

Case 1 ($2 \leq i \leq n$, $a_{i-1} = 0$) Select an edge $\boldsymbol{a}^{(i)} = (a_1, a_2, \ldots, a_{i-1}, \bar{a}_i, a_{i+1}, \ldots, a_n) \rightarrow \boldsymbol{a}^{(i,1)} = (\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_{i-1}, a_i, \bar{a}_{i+1}, \ldots, \bar{a}_n)$. In the a_i-M_{n-i}, construct a shortest path $\boldsymbol{a}^{(i,1)} \rightsquigarrow \boldsymbol{a}^{(1,i)} = (\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_{i-1}, a_i, \ldots, a_n)$ by `spr`. Select an edge $\boldsymbol{a}^{(1,i)} \rightarrow \boldsymbol{a}^{(1)}$.

Case 2 ($2 \leq i \leq n$, $a_{i-1} = 1$) Select an edge $\boldsymbol{a}^{(i)} = (a_1, a_2, \ldots, a_{i-1}, \bar{a}_i, \bar{a}_{i+1}, \ldots, \bar{a}_n) \rightarrow \boldsymbol{a}^{(i,1)} = (\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_{i-1}, a_i, a_{i+1}, \ldots, a_n)$. In the a_i-M_{n-i}, construct a shortest path $\boldsymbol{a}^{(i,1)} \rightsquigarrow \boldsymbol{a}^{(1,i)} = (\bar{a}_1, \bar{a}_2,$

$\ldots, \bar{a}_{i-1}, a_i, \bar{a}_{i+1} \ldots, \bar{a}_n)$ by `spr`. Select an edge $\boldsymbol{a}^{(1,i)} \rightarrow \boldsymbol{a}^{(1)}$.

Then, the $(n-1)$ paths constructed in Case 1 or Case 2 are Q_i: $\boldsymbol{a}^{(i)} \rightarrow \boldsymbol{a}^{(i,1)} \rightsquigarrow \boldsymbol{a}^{(1,i)} \rightarrow \boldsymbol{a}^{(1)}$ $(2 \leq i \leq n)$. Since each sub path $\boldsymbol{a}^{(i,1)} \rightsquigarrow \boldsymbol{a}^{(1,i)}$ is included in the a_i-M_{n-i} and the internal nodes of Q_i are all in the form of $(\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_{i-1}, a_i, \ldots)$, Q_i's $(2 \leq i \leq n)$ are disjoint except for $\boldsymbol{a}^{(1)}$ with each other.

It takes $O(n)$ to construct one of the paths in Case 1 or Case 2 since `spr` takes $O(n)$ time to construct a path. In total, the $(n-1)$ paths are constructed in $O(n^2)$ time.

The lengths of the paths are at most $1 + \lceil n/2 \rceil + 1 = \lceil n/2 \rceil + 2$. See Figure 3. □

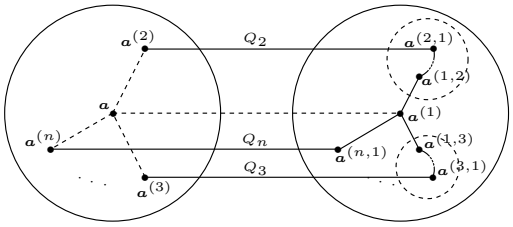


Fig. 3: Disjoint paths from $\boldsymbol{a}^{(i)}$ to $\boldsymbol{a}^{(1)}$ $(2 \leq i \leq n)$ in a 1-M_n

Lemma *3:* For an arbitrary node $\boldsymbol{a}$ in an M_n, we can construct $(n-1)$ paths Q_i: $\boldsymbol{a}^{(i)} \rightsquigarrow \boldsymbol{a}^{(1)}$ $(2 \leq i \leq n)$ that are disjoint except for $\boldsymbol{a}^{(1)}$. The time complexity for construction of these paths is $O(n^2)$. The lengths of the paths are at most $\lceil n/2 \rceil + 2$.

(Proof) It is trivial from Lemmas 1 and 2. □

3. Algorithm N2N

In this section, for a source node $\boldsymbol{s}$ and a destination node $\boldsymbol{d}$ in an M_n, we show an algorithm N2N that finds n paths from $\boldsymbol{s}$ to $\boldsymbol{d}$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$.

3.1 Procedure 1

In case that the source node and the destination node are included in the same M^j $(j \in \{0,1\})$, that is, $\boldsymbol{s}, \boldsymbol{d} \in M^j$, we construct n paths from $\boldsymbol{s}$ to $\boldsymbol{d}$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$ by the following Procedure 1.

Step 1 In the M^j, apply the algorithm recursively to construct $(n-1)$ paths from $\boldsymbol{s}$ to $\boldsymbol{d}$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$.

Step 2 Select edges $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(1)}$ and $\boldsymbol{d} \rightarrow \boldsymbol{d}^{(1)}$.

Step 3 In the $M^{\bar{j}}$, construct a path $\boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{d}^{(1)}$ by using the algorithm `spr`. See Figure 4.

3.2 Procedure 2

In case that the source node is included in M^j $(\boldsymbol{s} \in M^j)$ and the destination node is included in $M^{\bar{j}}$ $(\boldsymbol{d} \in M^{\bar{j}})$, and $\boldsymbol{s} = \boldsymbol{d}^{(1)}$, we construct n paths from $\boldsymbol{s}$ to $\boldsymbol{d}$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$ by the following Procedure 2.

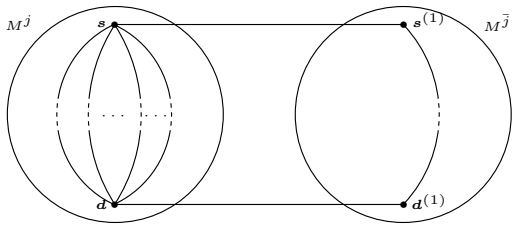


Fig. 4: After Step 3 in Procedure 1

Step 1 Select n edges $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(i)}$ $(1 \leq i \leq n)$.

Step 2 Construct $(n-1)$ paths Q_i: $\boldsymbol{s}^{(i)} \rightsquigarrow \boldsymbol{d}$ $(2 \leq i \leq n)$ from Lemma 3. Finally, n paths R_i $(1 \leq i \leq n)$ are obtained:

$$R_i : \begin{cases} \boldsymbol{s} \rightarrow \boldsymbol{d} & \text{if } i = 1, \\ \boldsymbol{s} \rightarrow \boldsymbol{s}^{(i)} \rightsquigarrow \boldsymbol{d}^{(i)} \rightarrow \boldsymbol{d} & \text{if } 2 \leq i \leq n. \end{cases}$$

See Figure 5.

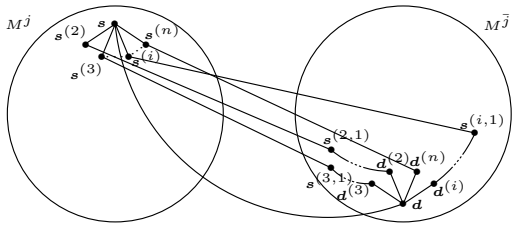


Fig. 5: After Step 2 in Procedure 2

3.3 Procedure 3

In case that the source node is included in M^j $(\boldsymbol{s} \in M^j)$ and the destination node is included in $M^{\bar{j}}$ $(\boldsymbol{d} \in M^{\bar{j}})$, and $\boldsymbol{s}$ and $\boldsymbol{d}^{(1)}$ are adjacent, we construct n paths from $\boldsymbol{s}$ to $\boldsymbol{d}$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$ by the following Procedure 3.

Step 1 In the M^j, apply the algorithm recursively to construct $(n-1)$ paths P_i: $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}^{(1,i)} \rightarrow \boldsymbol{d}^{(1)}$ $(2 \leq i \leq n)$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}^{(1)}$.

Step 2 Construct $(n-1)$ paths Q_i: $\boldsymbol{d}^{(1,i)} \rightsquigarrow \boldsymbol{d}$ from Lemma 3.

Step 3 Assume that $\boldsymbol{s} = \boldsymbol{d}^{(1,h)}$. Discard edges $\boldsymbol{d}^{(1,i)} \rightarrow \boldsymbol{d}^{(i)}$ $(2 \leq i \neq h \leq n)$. Select an edge $\boldsymbol{d}^{(1)} \rightarrow \boldsymbol{d}$. Finally, n paths R_i $(1 \leq i \leq n)$ are obtained:

$$R_i : \begin{cases} \boldsymbol{s} \rightarrow \boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{d}^{(h)} \rightarrow \boldsymbol{d} & \text{if } i = 1, \\ \boldsymbol{s} \rightsquigarrow \boldsymbol{d}^{(1,i)} \rightarrow \boldsymbol{d}^{(1,i,1)} \rightsquigarrow \boldsymbol{d}^{(i)} \rightarrow \boldsymbol{d} & \\ & \text{if } 2 \leq i \neq h \leq n, \\ \boldsymbol{s} \rightarrow \boldsymbol{d}^{(1)} \rightarrow \boldsymbol{d} & \text{if } i = h. \end{cases}$$

See Figure 6.

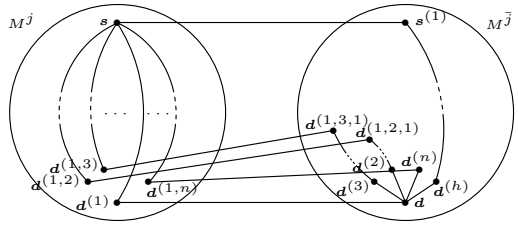


Fig. 6: After Step 3 in Procedure 3

3.4 Procedure 4

In case that the source node is included in M^j ($\boldsymbol{s} \in M^j$) and the destination node is included in $M^{\bar{j}}$ ($\boldsymbol{d} \in M^{\bar{j}}$), and the distance between $\boldsymbol{s}$ and $\boldsymbol{d}^{(1)}$ is more than 1, we construct n paths from $\boldsymbol{s}$ to $\boldsymbol{d}$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$ by the following Procedure 4.

Step 1 In the M^j, apply the algorithm recursively to construct $(n-1)$ paths P_i: $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}^{(1,i)} \rightarrow \boldsymbol{d}^{(1)}$ $(2 \leq i \leq n)$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}^{(1)}$. See Figure 7.

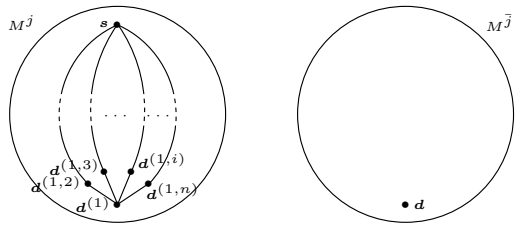


Fig. 7: After Step 1 in Procedure 4

Step 2 Construct $(n-1)$ paths Q_i: $\boldsymbol{d}^{(1,i)} \rightsquigarrow \boldsymbol{d}$ from Lemma 3. See Figure 8.

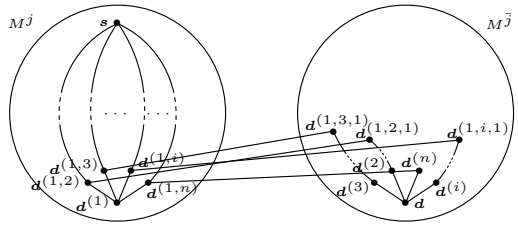


Fig. 8: After Step 2 in Procedure 4

Step 3 Select edges $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(1)}$ and $\boldsymbol{d}^{(1)} \rightarrow \boldsymbol{d}$.

Step 4 In the $M^{\bar{j}}$, construct a path $\boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{d}$ by using the algorithm `spr`.

Step 5 The path constructed in Step 4 includes at least one node on the paths constructed in Step 2, let $\boldsymbol{u}$ be the node on the path, say Q_h, that is closest to $\boldsymbol{s}^{(1)}$ along the path constructed in Step 4. See Figure 9.

Step 6 Discard the sub path $\boldsymbol{d}^{(1,h)} \rightarrow \boldsymbol{d}^{(1,h,1)} \rightsquigarrow \boldsymbol{u}$ and the $(n-2)$ edges $\boldsymbol{d}^{(1,i)} \rightarrow \boldsymbol{d}^{(1)}$ included in P_i $(2 \leq i \neq h \leq n)$.

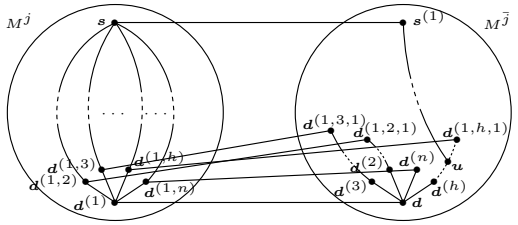


Fig. 9: After Step 5 in Procedure 4

Finally, n paths R_i $(1 \leq i \leq n)$ are obtained:

$$R_i : \begin{cases} \boldsymbol{s} \rightarrow \boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{u} \rightsquigarrow \boldsymbol{d}^{(h)} \rightarrow \boldsymbol{d} & \text{if } i = 1, \\ \boldsymbol{s} \rightsquigarrow \boldsymbol{d}^{(1,i)} \rightarrow \boldsymbol{d}^{(1,i,1)} \rightsquigarrow \boldsymbol{d}^{(i)} \rightarrow \boldsymbol{d} & \\ \qquad\qquad\qquad\qquad\qquad \text{if } 2 \leq i \neq h \leq n, & \\ \boldsymbol{s} \rightsquigarrow \boldsymbol{d}^{(1,h)} \rightarrow \boldsymbol{d}^{(1)} \rightarrow \boldsymbol{d} & \text{if } i = h. \end{cases}$$

See Figure 10.

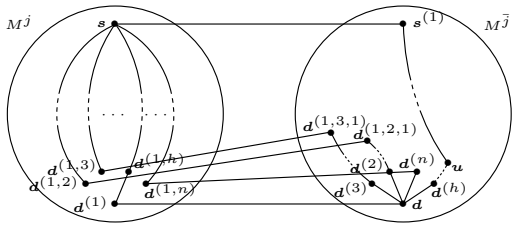


Fig. 10: After Step 6 in Procedure 4

4. Proof of Correctness and Estimation of Complexities

In this section, we prove the correctness of our algorithm and we give the estimates of time complexity $T(n)$ and maximum path length $L(n)$ for an n-dimensional möbius cube. Proof is based on induction on n.

Lemma *4:* In an M_n, the paths constructed by Procedure 1 are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The time complexity of Procedure 1 is $T(n-1)+O(n)$ and the maximum length of the paths constructed is $\max\{L(n-1), \lfloor n/2 \rfloor + 3\}$.

(Proof) The paths constructed in Step 1 are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$ by hypothesis of induction. The path constructed in Steps 2 and 3 is outside of M^j except for $\boldsymbol{s}$ and $\boldsymbol{d}$. Hence, the path cannot share any common node with the paths constructed in Step 1 except for $\boldsymbol{s}$ and $\boldsymbol{d}$, that is, it is disjoint with other paths constructed in Step 1 except for $\boldsymbol{s}$ and $\boldsymbol{d}$.

Step 1 takes $T(n-1)$ time to construct $(n-1)$ paths and the maximum length of them is $L(n-1)$ by hypothesis of induction. The path constructed in Steps 2 and 3 consists of two edges and a sub path by `spr`. Therefore, Steps 2 and 3 take $O(n)$ time to construct a path whose length is at most $2 + \lceil (n+1)/2 \rceil = \lfloor n/2 \rfloor + 3$. Hence, the time complexity

of Procedure 1 is $T(n-1)+O(n)$ and the maximum path length is $\max\{L(n-1), \lfloor n/2 \rfloor + 3\}$. □

Lemma *5:* In an M_n, the paths constructed by Procedure 2 are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The time complexity of Procedure 2 is $O(n^2)$ and the maximum length of the paths constructed is $\lceil n/2 \rceil + 3$.
(Proof)The n edges $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(i)}$ $(1 \leq i \leq n)$ selected in Step 1 are disjoint except for $\boldsymbol{s}$. The paths Q_i $(2 \leq i \leq n)$ constructed in Step 2 are disjoint each other except for $\boldsymbol{d}$. Therefore, the n paths R_i $(1 \leq i \leq n)$ are disjoint each other except for $\boldsymbol{s}$ and $\boldsymbol{d}$.

The time complexity for selecting the n edges in Step 1 is $O(n)$. From Lemma 3, it takes $O(n^2)$ in Step 2 to construct the $(n-1)$ paths Q_i $(2 \leq i \leq n)$ of lengths at most $\lceil n/2 \rceil + 2$. Hence, in total, it takes $O(n^2)$ to construct the n paths R_i $(1 \leq i \leq n)$ whose lengths are at most $1 + \lceil n/2 \rceil + 2 = \lceil n/2 \rceil + 3$. □

Lemma *6:* In an M_n, the paths constructed by Procedure 3 are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The time complexity of Procedure 3 is $T(n-1)+O(n^2)$ and the maximum length of the paths constructed is $L(n-1)+\lceil n/2 \rceil + 1$.
(Proof) The $(n-1)$ paths P_i $(2 \leq i \leq n)$ constructed in Step 1 are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$ by hypothesis of induction. The $(n-1)$ paths Q_i $(2 \leq i \leq n)$ constructed in Step 2 are disjoint with each other except for $\boldsymbol{d}$ from Lemma 3. Since Q_i is outside of M^j except for $\boldsymbol{d}^{(1,i)}$, it is disjoint from the paths P_i $(2 \leq i \leq n)$ except for $\boldsymbol{d}^{(1,i)}$. Then the paths R_i $(1 \leq i \neq h \leq n)$ obtained by connecting P_i and Q_i are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The path R_h: $\boldsymbol{s} \rightarrow \boldsymbol{d}^{(1)} \rightarrow \boldsymbol{d}$ is also disjoint with other paths R_i $(1 \leq i \neq h \leq n)$ except for $\boldsymbol{s}$ and $\boldsymbol{d}$.

Step 1 takes $T(n-1)$ time to construct $(n-1)$ paths and the maximum length of them is $L(n-1)$ by hypothesis of induction. From Lemma 3, it takes $O(n^2)$ in Step 2 to construct the $(n-1)$ paths Q_i $(2 \leq i \leq n)$ of lengths at most $\lceil n/2 \rceil + 2$. Thus, the length of the path R_1 is at most $\lceil n/2 \rceil + 2$, and the lengths of the paths R_i $(2 \leq i \neq h \leq n)$ are at most $L(n-1)-1+\lceil n/2 \rceil+2 = L(n-1)+\lceil n/2 \rceil+1$. Also, the length of the path R_h is 2. Hence, it takes $T(n-1)+O(n^2)$ time to construct the n paths R_i $(1 \leq i \leq n)$ whose lengths are at most $L(n-1)+\lceil n/2 \rceil + 1$. □

Lemma *7:* In an M_n, the paths constructed by Procedure 4 are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The time complexity of Procedure 4 is $T(n-1)+O(n^2)$ and the maximum length of the paths constructed is $\max\{n+2, L(n-1)+\lceil n/2 \rceil + 1\}$.
(Proof) The $(n-1)$ paths P_i $(2 \leq i \leq n)$ constructed in Step 1 are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$ by hypothesis of induction. The $(n-1)$ paths Q_i $(2 \leq i \leq n)$ constructed in Step 2 are disjoint with each other except for $\boldsymbol{d}$ from Lemma 3. Since Q_i is outside of M^j except for $\boldsymbol{d}^{(1,i)}$, it is disjoint from the paths P_i $(2 \leq i \leq n)$ except for $\boldsymbol{d}^{(1,i)}$. The path $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{u} \rightsquigarrow \boldsymbol{d}$ is also disjoint with other paths P_i and Q_i $(2 \leq i \leq n)$ except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The path $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}^{(1,h)} \rightarrow \boldsymbol{d}^{(1)} \rightarrow \boldsymbol{d}$ is disjoint with other paths P_i and Q_i $(2 \leq i \neq h \leq n)$ except for $\boldsymbol{s}$ and $\boldsymbol{d}$.

Step 1 takes $T(n-1)$ time to construct $(n-1)$ paths and the maximum length of them is $L(n-1)$ by hypothesis of induction. From Lemma 3, it takes $O(n^2)$ in Step 2 to construct the $(n-1)$ paths Q_i $(2 \leq i \leq n)$ of lengths at most $\lceil n/2 \rceil + 2$. It takes $O(1)$ time to select two edges $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(1)}$ and $\boldsymbol{d} \rightarrow \boldsymbol{d}^{(1)}$ in Step 3. Step 4 takes $O(n)$ to construct the path $\boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{d}$ of length at $\lceil (n+1)/2 \rceil = \lfloor n/2 \rfloor + 1$. In Step 5, we can find the node $\boldsymbol{u}$ as follows. We check each of the nodes $\boldsymbol{v}$ on the path constructed in Step 4 from $\boldsymbol{s}^{(1)}$ to $\boldsymbol{d}$ one by one. Since sub paths $\boldsymbol{d}^{(1,i,1)} \rightsquigarrow \boldsymbol{d}^{(i)}$ of Q_i are included in disjoint sub graphs M_{n-i}'s, we can check if $\boldsymbol{v}$ is included in one of them just by checking its preceding i bits. Hence, if $\boldsymbol{v}$ is included in the M_{n-h} that includes the sub path $\boldsymbol{d}^{(1,h,1)} \rightsquigarrow \boldsymbol{d}^{(h)}$ of Q_h, it is enough to check whether $\boldsymbol{v}$ is on the sub path or not to determine $\boldsymbol{v}$ is the node $\boldsymbol{u}$ or not. For each $\boldsymbol{v}$, it takes $O(n)$ time. Hence, it takes $O(n^2)$ time in Step 5 to find the node $\boldsymbol{u}$. In Step 6, discarding the sub path and the $(n-2)$ edges takes $O(n)$ time. In total, the time complexity is $T(n-1)+O(n^2)+O(1)+O(n)+O(n^2)+O(n) = T(n-1)+O(n^2)$. The length of the path R_1 is at most $1+\lfloor n/2 \rfloor + (\lceil n/2 \rceil + 1) = n+2$. The maximum length of the paths R_i $(2 \leq i \neq h \leq n)$ is $(L(n-1)-1)+(\lceil n/2 \rceil + 2) = L(n-1)+\lceil n/2 \rceil + 1$. The length of the path R_h is at most $L(n-1)+1$. Therefore, the maximum path length is $\max\{n+2, L(n-1)+\lceil n/2 \rceil + 1\}$. □

Theorem *1:* For a node $\boldsymbol{s}$ and a node $\boldsymbol{d}$ in an M_n, Algorithm N2N finds n paths from $\boldsymbol{s}$ to $\boldsymbol{d}$ that are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The time complexity $T(n)$ of N2N is $O(n^3)$, and the maximum path length $L(n)$ is

$$L(n) = \begin{cases} (n^2+6n-4)/4 & \text{if } n \geq 2, n\text{: even}, \\ (n^2+6n-3)/4 & \text{if } n \geq 3, n\text{: odd}. \end{cases}$$

(Proof) From Lemmas 4 to 7, the constructed paths are disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. Also, $T(n) = O(n^3)$ from $T(n) = T(n-1)+O(n^2)$ and $T(2) = O(1)$. From $L(n) = \max\{n+2, L(n-1)+\lceil n/2 \rceil + 1\}$ and $L(2) = 3$, the equation is derived. □

5. Performance Evaluation

To evaluate average performance of Algorithm N2N, we conducted a computer experiment by repeating following steps at least $10,000$ times for random pairs of the source node $\boldsymbol{s}$ and the destination node $\boldsymbol{d}$ in a 0-M_n and a 1-M_n for each n between 4 and 31.

1) Select a source node $\boldsymbol{s}$ randomly.
2) Select a destination node $\boldsymbol{d}$ randomly other than $\boldsymbol{s}$.
3) For $\boldsymbol{s}$ and $\boldsymbol{d}$, apply Algorithm N2N and measure the execution time and the maximum path length.

The algorithm was implemented using the programming language `C++`. The program was compiled with the GNU G++ compiler `g++` with a `-O` option. The target machine is equipped with an Intel Core i5-3230M CPU 2.60 GHz

and 4GB RAM. The program was running at Oracle VM VirtualBox with 1GB RAM.

The average execution time to construct a node-to-node disjoint paths and their maximum lengths are shown in Figures 11 and 12, respectively. Figure 11 shows that the average execution time is $O(n^{1.5})$. From Figure 12, we can see that the maximum path lengths on average is linear to n and the theoretical maximum path lengths are not easily attained.

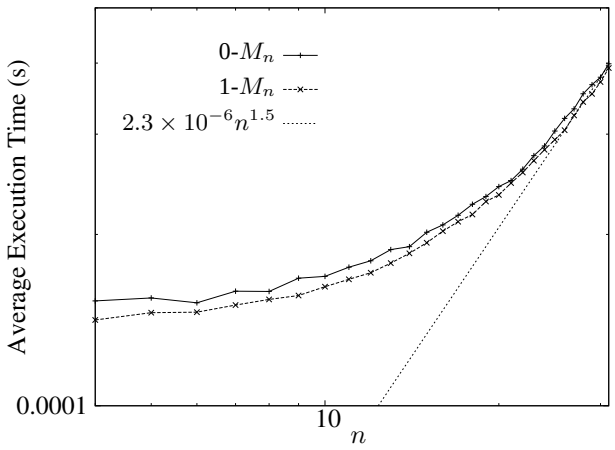


Fig. 11: Average execution time of Algorithm N2N.

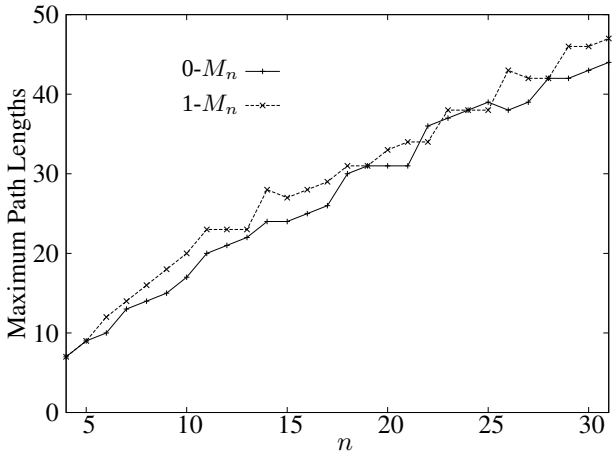


Fig. 12: Maximum lengths of paths constructed by Algorithm N2N.

6. Conclusions

In this paper, we proposed a polynomial-order time algorithm for the node-to-node disjoint paths problem in n-möbius cubes. Its time complexity is $O(n^3)$ and the maximum path length is $O(n^2)$. We also conducted a computer experiment and showed that the average execution time is $O(n^{1.5})$ and the maximum path lengths on average are linear to n.

Future works include theoretical analysis of the maximum path length of the algorithm as well as its average performance. Also, improvement of the algorithm to construct shorter paths in smaller execution time is also interesting for us.

Acknowledgments

This study is partly supported by a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science (JSPS) under Grant No. 25330079.

References

[1] S.B. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," IEEE Transactions on Computers, vol.38, no.4, pp.555–566, April 1989.

[2] S.G. Akl and K. Qiu, "Parallel minimum spanning forest algorithms on the star and pancake interconnection networks," Proceedings of Joint Conference on Vector and Parallel Processing, pp.565–570, 1992.

[3] S.G. Akl and K. Qiu, "A novel routing scheme on the star and pancake interconnection networks and its applications," Parallel Computing, vol.19, no.1, pp.95–101, Jan. 1993.

[4] S.G. Akl, K. Qiu, and I. Stojmenović, "Fundamental algorithms for the star and pancake interconnection networks with applications to computational geometry," Networks, vol.23, no.4, pp.215–226, July 1993.

[5] P. Berthomé, A. Ferreira, and S. Perennes, "Optimal information dissemination in star and pancake networks," IEEE Transactions on Parallel and Distributed Systems, vol.7, no.12, pp.1292–1300, Dec. 1996.

[6] A. Bossard, "A set-to-set disjoint paths routing algorithm in hyper-star graphs," ISCA International Journal of Computers and Their Applications, vol.21, no.1, pp.76–82, March 2014.

[7] A. Bossard and K. Kaneko, "Time optimal node-to-set disjoint paths routing in hypercubes," Journal of Information Science and Engineering.

[8] A. Bossard and K. Kaneko, "The set-to-set disjoint-path problem in perfect hierarchical hypercubes," The Computer Journal, vol.55, no.6, pp.769–775, June 2012.

[9] P.F. Corbett, "Rotator graphs: An efficient topology for point-to-point multiprocessor networks," IEEE Transactions on Parallel and Distributed Systems, vol.3, no.5, pp.622–626, May 1992.

[10] P. Cull and S.M. Larson, "The mbius cubes," IEEE Transactions on Computers, vol.44, no.5, pp.647–659, May 1995.

[11] M. Dietzfelbinger, S. Madhavapeddy, and I.H. Sudborough, "Three disjoint path paradigms in star networks," Proceedings of the IEEE Symposium on Parallel and Distributed Processing, pp.400–406, Dec. 1991.

[12] J. Fan, "Hamilton-connectivity and cycle-embedding of the möbius cubes," Information Processing Letters, vol.82, no.2, pp.113–117, 2002.

[13] L. Gardner, Z. Millter, D. Pritikin, and I.H. Sudborough, "Embedding hypercubes into pancake, cycle prefix and substring reversal networks," Proceedings of the 28th Annual Hawaii International Conference on System Sciences, pp.537–545, Jan. 1995.

[14] L. Gargano, U. Vaccaro, and A. Vozella:, "Fault tolerant routing in the star and pancake interconnection networks," Information Processing Letters, vol.45, no.6, pp.315–320, 1993.

[15] W.H. Gates and C.H. Papadimitriou, "Bounds for sorting by prefix reversal," Discrete Mathematics, vol.27, pp.47–57, 1979.

[16] Q.P. Gu and S. Peng, "Set-to-set fault tolerant routing in star graphs," IEICE Transactions on Information and Systems, vol.E79-D, no.4, pp.282–289, April 1996.

[17] Q.P. Gu and S. Peng, "Node-to-set disjoint paths problem in star graphs," Information Processing Letters, vol.62, no.4, pp.201–207, April 1997.
[18] Q.P. Gu and S. Peng, "Node-to-set and set-to-set cluster fault tolerant routing in hypercubes," Parallel Computing, vol.24, no.8, pp.1245–1261, 1998.
[19] Y. Hamada, F. Bao, A. Mei, and Y. Igarashi, "Nonadaptive fault-tolerant file transmission in rotator graphs," IEICE Transactions on Fundamentals, vol.E79-A, no.4, pp.477–482, April 1996.
[20] S.Y. Hsieh and C.H. Chen, "Pancyclicity on möbius cubes with maximal edge faults," Parallel Computing, vol.30, no.3, pp.407–421, 2004.
[21] K. Kaneko, "An algorithm for node-to-set disjoint paths problem in burnt pancake graphs," IEICE Transactions on Information and Systems, vol.E86-D, no.12, pp.2588–2594, Dec. 2003.
[22] K. Kaneko and Y. Suzuki, "An algorithm for node-to-set disjoint paths problem in rotator graphs," IEICE Transactions on Information and Systems, vol.E84-D, no.9, pp.1155–1163, Sept. 2001.
[23] S. Madhavapeddy and I.H. Sudborough, "A topological property of hypercubes — node disjoint paths," Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing, pp.532–539, Dec. 1990.
[24] K. Qiu, H. Meijer, and S.G. Akl, "Parallel routing and sorting on the pancake network," Proceedings of International Conference on Computing and Information, Lecture Notes in Computer Science, vol.497, pp.360–371, Springer Verlag, 1991.
[25] C.L. Seitz, "The cosmic cube," Communications of the ACM, vol.28, no.1, pp.22–33, Jan. 1985.
[26] Y. Suzuki and K. Kaneko, "An algorithm for node-disjoint paths in pancake graphs," IEICE Transactions on Information & Systems, vol.E86-D, no.3, pp.610–615, March 2003.
[27] C.H. Tsai, "Embedding of meshes in möbius cubes," Theoretical Computer Science, vol.401, no.1-3, pp.181–190, 2008.
[28] J.M. Xu, M. Ma, and M. Lü, "Paths in möbius cubes and crossed cubes," Information Processing Letters, vol.97, no.3, pp.94–97, Feb. 2006.
[29] X. Yang, G.M. Megson, and D.J. Evans, "Pancyclicity of möbius cubes with faulty nodes," Microprocessors and Microsystems, vol.30, no.3, pp.165–172, May 2006.

Decomposing BPC Permutations into Semi-Permutations for Crosstalk Avoidance in Multistage Optical Interconnection Networks

Gennady Veselovsky, Ritu Jain
Department of Computer and Network Engineering
Assumption University
Soi 24, Ramkhamhaeng Road, Huamark
Bangkok 10240, Thailand [1]

Abstract *- This paper introduces a simple O(N) algorithm that decomposes BPC (bit-permute-complement) permutations into semi-permutations for avoiding crosstalk when realizing them in N × N optical multistage interconnection networks (OMINs). Crosstalk means that two optical signals, sharing an optical switch, undergo a kind of undesired coupling. A semi-permutation is a partial permutation which meets the requirement for each switch in an input and output stages of the network to be used with only one optical signal at a time. It provides avoiding crosstalk in the first and the last stages of a network and creates the potential for crosstalk-free realization of a semi-permutation, and finally the whole permutation in question. The algorithm is based on employment the periodicity of appearing 1's and 0's in columns of transition matrices for BPC permutations.*

Keywords: *Multistage optical interconnection networks, BPC permutations, semi-permutations, crosstalk avoidance.*

1 Introduction

The vast number of processing elements in massively computers dictates heavy demands for performance of an interconnection network in use. Optical interconnection networks constitute a promising choice in the field because they offer gigabit transmission capacity, very big bandwidth, and low error probability. In what follows we consider hybrid optical multistage interconnection networks (OMINs) using guided wave technology and composed of electronically controlled directional couplers because other types of optical networks are still difficult to implement. In spite of the topological similarity of electronic and optical multistage interconnection networks, the latter cause some specific problems, and the major of them is optical crosstalk. It means that two optical signals, sharing an optical switch, undergo a kind of undesired coupling. The crosstalk reduces essentially signal to noise ratio and so limits the size of a network, whereas the number of processing nodes in modern supercomputers is increasing rapidly. To avoid crosstalk two optical signals should be sent at different time, if they use the same switch. It is called *time domain* approach in distinction from *space domain* approach when crosstalk avoidance is achieved with significant increase of hardware. Crosstalk avoiding problem is discussed in a large number of works, e.g. in [1], [2], [3], [4], [5], [6], to mention only a few. In what follows we have to refer often to the term *switch conflict*. It means using an optical switch (directional coupler) by two input signals at the same time resulting in above mentioned crosstalk. It is noteworthy also that for optical hybrid OMINs under consideration circuit switching rather than packet switching is usually preferred, since with packet switching the address information in each packet must be decoded in each stage that means conversion from optical signal to electronic and so can be very costly.

A permutation is one of the most common communication patterns in parallel computing systems. In the context of a parallel computing system, a permutation means simultaneous transferring of data items between the nodes, with all the destination nodes being different. The term *permutation* can be defined as a request for parallel connection of N sources to N destinations, where $N = 2^n$, with a distinct destination for each of the sources: $S_0 \to D_{0,} S_1 \to D_1, \ldots, S_{N-1} \to D_{N-1}$. Its components will be considered as n-dimensional vectors whose

The research has been funded by Assumption University of Thailand

elements are either 0 or 1: the vector $s_0 s_1 \ldots s_{n-2} s_{n-1}$ is identified with the integer S_i, s_0 being the most significant bit. The following sequence $s_0 s_1 \ldots s_{n-2} s_{n-1} d_0 d_1 \ldots d_{n-2} d_{n-1}$ is called a *transition sequence* for an input-output pair. The set of all transition sequences for a given permutation is called its *transition matrix* [7]. If a permutation does not possess any regularity, it is called an arbitrary permutation. On the contrary, if there is some general rule for producing a destination address from a source address, a permutation is called a regular one. There are known different classes of regular permutations. A permutation is called a bit-permute-complement (BPC) permutation if the destination address can be produced by permuting bits in the source address and/or complementing some or all of its bits positions. BPC permutations are widely used in parallel programming when solving various scientific problems (digital signal processing, large matrices processing etc). A transition matrix for a BPC permutation in a symbolic form can be represented as follows:

$$S_0 S_1 \ldots S_{n-2} S_{n-1} S_{\pi(0)} S_{\pi(1)} \ldots S_{\pi(n-2)} S_{\pi(n-1)} [7].$$

A semi-permutation is a partial permutation which meets the requirement for each switch in an input and output stages of the network to be used with only one optical signal at a time. It provides avoiding crosstalk in the first and the last stages and creates the potential for crosstalk-free realization of semi-permutations, and finally the whole permutation in question [1]. It is evident that for crosstalk-free realization of a semi-permutation crosstalk in intermediate switches also should be eliminated, and it is the next step in crosstalk avoiding. However, decomposing a permutation into semi-permutations can be considered as a specific problem. The problem is being solved for arbitrary permutations in [1], however, to our best knowledge decomposing BPC permutations to semi-permutations was considered before only in the work with participation of the first author of this paper but for some specific representatives of that class [8]. In this paper a fast algorithm for decomposing BPC permutations in general into semi-permutations is presented. Its time complexity is *O(N)*, where $N \times N$ is the size of a network. The algorithm presented in the paper is based on the regularity of BPC permutations, namely, on periodicity of appearing 1s and 0s in columns of their transition matrices. The algorithm is implemented in C language for two basic types of optical multistage interconnection networks (OMINs), namely with perfect shuffle interconnection pattern before the first stage and without it. In our work some computational experiments were carried out, including those in concern with BPC permutations known as worst cases from the viewpoint of routing. The total number of BPC permutations is $2^n n!$. There are some frequently used BPC permutations as given in [9] and in [10]. Such permutations are usually referred to by names, with each equation showing mapping a source $s_0 s_1 \ldots s_{n-2} s_{n-1}$ to the destination. The permutations which are referred to in our paper were taken from the list below:

1. *Perfect shuffle*

$$\pi_{PSH} = s_1 s_2 \ldots s_{n-1} s_0$$

2. *Unshuffle*

$$\pi_{USH} = s_{n-1} s_0 \ldots s_{n-3} s_{n-2}$$

3. *Vector reversal*

$$\pi_{VR} = \overline{s_0} \overline{s_1} \ldots \overline{s_{n-2}} \overline{s_{n-1}}$$

4. *Butterfly*

$$\pi_{BF} = s_{n-1} s_1 \ldots s_{n-2} s_0$$

5. *Exchange*

$$\pi_{EXCH} = s_0 s_1 \ldots s_{i-1} \overline{s_i} s_{i+1} \ldots s_{n-1} s_{n-2}$$

6. *Bit reversal*

$$\pi_{BR} = s_{n-1} s_{n-2} \ldots s_1 s_0$$

7. *Matrix transpose*

$$\pi_{MT} = s_l s_{l+1} \ldots s_{2l-1} s_0 s_1 \ldots s_{l-1} \text{ if } n=2l$$

$$\pi_{MT} = s_l s_{l+1} \ldots s_{2l} s_0 s_1 \ldots s_{l-1} \text{ if } n=2l+1$$

8. *Bit shuffle*

$$\pi_{BSH} = s_0 s_2 \ldots s_{n-2} s_1 s_3 \ldots s_{n-1} \text{ if } n=2l$$

$$\pi_{BSH} = s_0 s_2 \ldots s_{n-1} s_1 s_3 \ldots s_{n-2} \text{ if } n=2l+1$$

9. *Shuffle row major*

$$\pi_{SHRM} = s_0 s_l s_1 s_{l+1} \ldots s_l s_{2l-1} \text{ if } n=2l$$

$$\pi_{SHRM} = s_0 s_{l+1} s_1 s_{l+2} \ldots s_{l-1} s_{2l-1} s_l \text{ if } n=2l+1$$

2 An algorithm for decomposing BPC permutations into semi-permutations

The essential part of the algorithm consists in determining so called *antagonists*, i.e. the pairs of inputs to an interconnection network which cause switch conflicts either in the first (input) stage or in the last (output) stage of the network. The regularity of BPC permutations provides possibility for finding crosstalk-free set of outputs for any crosstalk-free set of inputs. In its turn finding crosstalk-free set of inputs to a network is trivial. Here we consider two network topologies: (2n-1)-stage shuffle exchange network (SEN) and Benes network [11]. It is proved in [1] that any semi-permutation can be realized in a single pass in Benes network under the constraint of avoiding crosstalk. Benes network is known as a classic rearrangeable network, since (*2n*-1)-stage SEN is also always rearrangeable [12], we can here conjecture the same about it. SEN topology is considered here because it offers an identical interconnection pattern across all stages thus simplifying implementation especially in a photonic integration environment [13].

As it follows from the definition of a semi-permutation given in the Introduction, the basic requirements when decomposing a given BPC permutation into semi-permutations are the next:

- None of the source-destination pairs should share the same switch in the first stage of a network with another pair.
- None of the source-destination pairs should share the same switch in the output stage of a network with another pair.

For producing semi-permutations, two lists of antagonists are needed. The first one is of the inputs to a network sharing the inputs of the same switches in the first stage. In a SEN all connections between inputs to a network and inputs to the switches of the first stage are permanent and are implemented in accordance with perfect shuffle connection pattern :

$$Shuffle(a_{n-1}a_{n-2}...a_1a_0) = a_{n-2}...a_1a_0a_{n-1}$$ [14].

In other words the connection is based on cyclic shift to the left by one bit position. So the antagonists at inputs can be found using the following formula for any possible case:

$$N = 2^n, \quad Antagoinsts\ at\ inputs:\ i + \frac{N}{2}, \qquad 0 \le i \le \frac{N}{2}$$

As an example, for 16x16 SEN antagonistic pairs of inputs shown below:

0/8, 1/9, 2/10, 3/11, 4/12, 5/13, 6/14, 7/15.

The inputs shown together cannot be presented in the same semi-permutation. On the contrary for a Benes network there is no rearranging at inputs of the network, and antagonists at the inputs of 16x16 Benes network look as follows:

0/1, 2/3, 4/5, 6/7, 8/9, 10/11, 12/13, 14/15.

However, producing the list of input pairs – antagonists - causing switch conflicts in the output stage of the network is not so trivial, and it is based on periodicity of 0's and 1's in the rightmost column of the transition matrix of a given BPC permutation. That periodicity defines the periodicity of appearing identical $n-1$ combinations in the remainder part (without the rightmost bit) of the destinations half of the matrix. In other words it defines antagonistic rows in the binary version of a transition matrix. If resort to the transition matrix in its symbolic form, then with the rightmost component in the destination address being S_0, the period of changing 0's and 1's in the sequence of output numbers on the right side of a network diagram or in the rightmost column of a transition matrix is $N/2$, with S_1 that period is $N/4$, etc. with increasing degrees of 2 in denominators up to N; so for S_{n-1} that period is 1. It is noteworthy that for all BPC permutations with the same rightmost component in their symbolic transition matrices the list of antagonists for outputs will be the same because permuting bits in the rest part of a destination address will not affect identity of the appropriate rows. Finally the formula for determining the afore said period R_i looks as follows: $R_i = 2^{(n-1)-I}$, $0 \le i \le n-1$, with i being the index of the rightmost component of the destination part in a symbolic transition matrix.

To summarize, decomposing a BPC permutation to semi-permutations includes the following steps:

1. Initialize N and n.
2. Initialize a transition matrix for a given BPC permutation in the symbolic form.

3. Produce the list of input pairs which are antagonistic at the network's first stage inputs basing on afore said reasoning.
4. Produce the list of input pairs causing switch conflicts in the output stage of a network (antagonists for the network's outputs) basing on the specific for a given permutation periodicity of 0's and 1's in the rightmost column of a binary version of a transition matrix in accordance with the formula for R_i above in the text.
5. Combine both lists of antagonists to one list as shown in the following examples, scan the lists of the inputs to a network with deleting antagonists to a current input. Each antagonist is encountered twice: either as an input or an output. Total number of pairs is $N/2$ in both cases, so exactly $N/2$ antagonists will be deleted (in other words, assigned to the second semi-permutation). Each antagonist is deleted once despite of encountering twice, so $N/2$ numbers of inputs are always left.

The above algorithm has time complexity *O (N)*.

3 Examples of Decomposing

An example of applying our approach for decomposing Unshuffle permutation for realizing it in 8×8 shuffle-exchange and Benes networks is given below. Crosstalk-free routing in intermediate stages has been done by trial and error. In our case its transition matrix is $S_0S_1S_2S_2S_2S_0S_1$ and so the period *R=2*.

For *N=8* the original Unshuffle permutation looks as follows:

0→0, 1→4, 2→1, 3→5, 4→2, 5→6, 6→3, 7→7.

In accordance with afore said it can be easily decomposed into a pair of semi-permutations for 8×8 SEN:

0 →0, 1→4, 6→3, 7→7 : 1st S-P;
4→2, 5→6, 2→1, 3→5 : 2nd S-P.

Decomposing the same permutation for Benes network:

0 →0, 1→4, 6→3, 7→7: 1st S-P;
4→2, 5→6, 2→1, 3→5 : 2nd S-P.

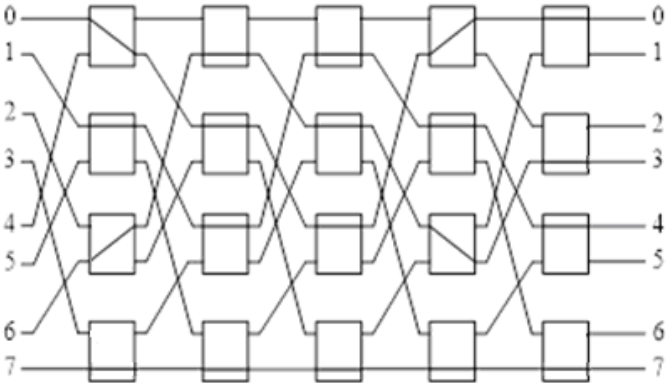


Figure 1. Routing the first semi-permutation of Unshuffle permutation on 8×8 5-stage shuffle-exchange network.

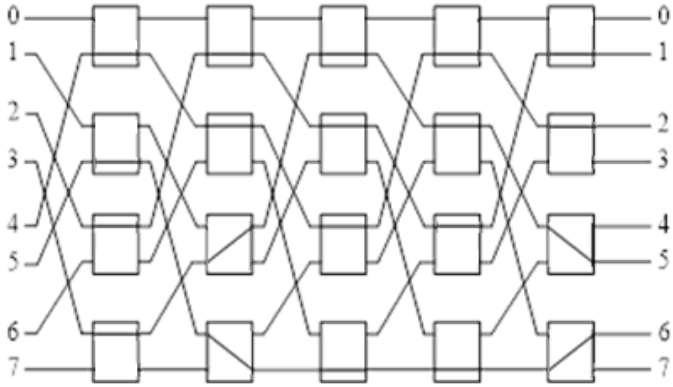


Figure 2. Routing the second semi-permutation of Unshuffle permutation on 8×8 5-stage shuffle-exchange network.

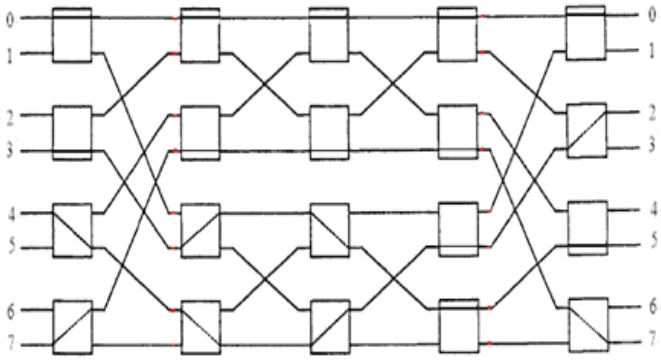


Figure 3. Routing the first semi-permutation of Unshuffle permutation on 8×8 5-stage Benes network.

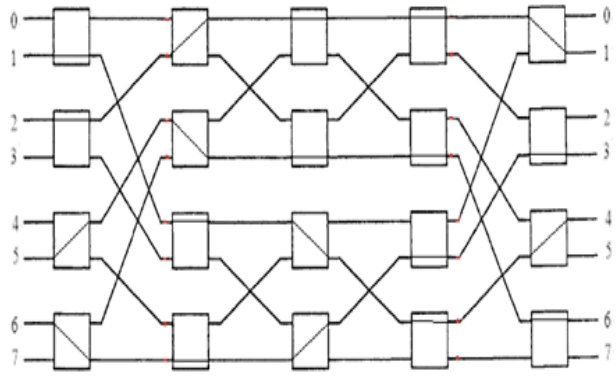


Figure 4. Routing the second permutation of Unshuffle permutation on 8×8 5-stage Benes network.

The second example is of larger size and concerns decomposing Butterfly permutation for 16×16 shuffle-exchange and Benes networks. In this case the transition matrix is $S_0S_1S_2S_3S_3S_1S_2S_0$, and the period $R=8$. The original permutation is:

0→0, 1→8, 2→2, 3→10, 4→4, 5→12, 6→6, 7→14, 8→1, 9→9, 10→3, 11→11, 12→5, 13→13, 14→7, 15→15.

More descriptive is its representing in the Table below.

Table 1
Butterfly permutation 16×16

Source Input $S_0S_1S_2S_3$	**Outputs** $S_3S_1S_2S_0$
0000 (0)	0000 (0)
0001 (1)	1000 (8)
0010 (2)	0010 (2)
0011 (3)	1010 (10)
0100 (4)	0100 (4)
0101 (5)	1100 (12)
0110 (6)	0110 (6)
0111 (7)	1110 (14)
1000 (8)	0001 (1)
1001 (9)	1001 (9)
1010 (10)	0011 (3)
1011 (11)	1011 (11)
1100 (12)	0101 (5)
1101 (13)	1101 (13)
1110 (14)	0111 (7)
1111 (15)	1111 (15)

It is easy to see from the Table 1 that periodicity of outputs which belongs to the same switch in the output stage, i.e. differ only in the rightmost bit, is really equals 8.

When decomposing afore said permutation for 16×16 SEN the steps are following.
Composing the list of antagonists at the input stage of the network (it is the same for any BPC permutation of a given size):

0/8, 1/9, 2/10, 3/11, 4/12, 5/13, 6/14, 7/15.

Composing the list of antagonists at the output stage is the next step. To be exact we find pair of inputs which evoke switch conflicts in the output stage for a given BPC permutation, e.g. simultaneous optical signals at inputs 0 and 8 would result in a switch conflict because corresponding outputs (0 and 1) belong to the same switch in the output stage of the network. All this can be easily seen from the Table 1. The rightmost component in the destination address is S_0, what means that we need to add 8 to the number of an input to find its antagonist from the viewpoint condition at the output stage:

0/8, 2/10, 4/12, 6/14, 1/9, 3/11, 5/13, 7/15.

The inputs shown together cannot be presented in the same semi-permutation; we delete the pair from the original permutation that uses the same switch either in an input or output stage.

We take a look at the first pair 0→0 from the original permutation.
From the lists of antagonist at inputs and outputs, 0 and 8 belong to the same switch.
Therefore, we delete 8→1 from the original permutation.

Next, we take a look at the pair 2→2.
From the lists of antagonist at inputs and outputs, 2 and 10 belong to the same switch.
Therefore, we delete 10→3 from the original permutation.

This is continued until we find the following semi-permutations:

0→0, 1→8, 2→2, 3→10, 4→4, 5→12, 6→6, 7→14 : 1st S-P
8→1, 9→9, 10→3, 11→11, 12→5, 13→13, 14→7, 15→15 : 2nd S-P

When decomposing the same permutation for a 16×16 Benes network, the list of antagonists for the first stage differs from that in the previous case and looks as follows:

0/1, 2/3, 4/5, 6/7, 8/9, 10/11, 12/13, 14/15.

However the list of antagonists concerning the output stage of the network is the same and produced in the same way by adding 8 to the number of an input:

0/8, 2/10, 4/12, 6/14, 1/9, 3/11, 5/13, 7/15.

The inputs shown together cannot be presented in the same semi-permutation; we delete the pair from the original permutation that belongs to the same switch in the same manner as in the previous case and as a result produce two following semi-permutations (the deleted pairs form another semi-permutation):

0→0, 2→2, 4→4, 6→6, 9→9, 11→11, 13→13, 15→15 : 1st S-P
1→8, 3→10, 5→12, 7→14, 8→1, 10→3, 12→5, 14→7 : 2nd S-P

4 Conclusion

In this paper an *O(N)* algorithm for decomposing BPC (bit-permute-complement) permutations into semi-permutations is introduced. The work contributes to solving the problem of crosstalk-free routing in hybrid optical multistage interconnection networks (OMINs). The algorithm is based on employment the property of periodicity 0's and 1's within columns of transition matrices for permutations belonging to the aforesaid class, it provides crosstalk-free conditions in the first and last stages of an OMIN. The algorithm is much simpler than known ones but for arbitrary permutations. The availability of semi-permutations creates the potential for crosstalk-free routing permutations for two passes through a network in an ideal case. Examples of decomposing some BPC permutations for (2n-1)-stage shuffle-exchange and Benes networks are given. The approach is applicable also to any type of banyan networks. The algorithm is implemented in C. A possible extension of the approach for providing crosstalk-free condition in intermediate stages of an OMIN is being studied.

5 References

[1] Y. Yang, J. Wang, Y. Pan, Permutation capability of optical multistage interconnection networks, *Journal of Parallel and Distributed Computing, 60,* 2000, 72-91.

[2] Y. Pan, C. Qiao, Y. Yang, Optical multistage interconnection networks: new challenges and approaches, *IEEE Communication , 2*, 1999, 50-56.

[3] S.Kaur, R. Vohra, S. Kaur, Analysis of various crosstalk avoidance techniques in optical multistage interconnection network, *International Journal of P2P Network Trends and Technology, 1(2)*, 2011, 26-30.

[4] X. Lin, Y. Zhao, and Y. Wu, An efficient crosstalk-free routing algorithm based on permutation decomposition for optical multi-$\log_2 N$ switching networks, *Proc. 10th IFIP International Conference, Guiyana, China, LNCS 8147,* 2013, 207-219.

[5] R. Bashirov and T. Karanfiller, On path dependent loss and switch crosstalk reduction in optical networks, *Information Sciences, (180)*, 2005, 1040-1050.

[6] S. Mishra, N. Chaudhary, K. Singh, Overview of optical interconnect technology, *International Journal of Scientific & Engineering Research*, 3(4), 2012, 1-7.

[7] X. Shen, Optimal realization of any BPC permutation on *k*-extra-stage Omega networks, *IEEE Trans. Computers, 44(5)*, 1995, 714-719.

[8] G. Veselovsky and A. Agrawal, On crosstalk-free BPC permutations routing in an optical variable-stage shuffle-exchange networks, *Proc. IASTED Intl. Conf. PDCN 2014, Innsbruck, Austria,* 2014, 232-238.

[9] J. Lenfant, A versatile mechanism to move data in an array processor, *IEEE Trans. Computers, C-34 (6)*, 1985, 506-522.

[10] M. D. Grammatikakis, D. Frank Hsu, M. Kraetzl, *Parrallel System Interconnections and Communication*, CRC Press, 2000.

[11] H. J. Siegel, *Interconnection networks for large-scale parallel processing. Theory and case studies*, 2nd ed. McGraw-Hill International Editions, 1990.

[12] A. Abdennadher and T. Y. Feng, On rearrangeability of Omega-Omega networks, *Proc. IEEE Int. Conf. Parallel Proc., I,* 1992, 159-165.

[13] A. Shacham, *Architectures of optical interconnection networks for high performance computing*, VDM Verlag Dr. Muller, 2007.

[14] H. Stone, Parallel processing with the perfect shuffle, *IEEE Trans. Computers, 20(2*), 1971, 153-161.

A New Efficient Distributed Load Balancing Algorithm for OTIS-Star Networks

A. Awwad[1], **J. Al-Sadi**[2]
[1]Department of CS, University of Petra, Amman, Jordan
[2] Department of ITC, Arab Open University, Amman, Jordan

Abstract - *The OTIS-Star interconnection network is one of the promising interconnection networks for future high speed distributed computing, The OTIS-Star topology consists of both edge and vertex symmetry, it has many attractive topological proprieties based on it is own hierarchical structure nature. Many research efforts have been devoted on this attractive network in literature, but these efforts did not address the problem of load balancing which is an important issue that must be addressed in any new topology. In this paper, we are investigating and proposing a new efficient algorithm for load balancing problem for the OTIS-Star network. The investigated algorithm is called OTIS-Star Electronic-Optical-Electronic Exchange Method; OSEOEM for short; to be implemented on the OTIS-Star network. The proposed algorithm is based on the FOFEM method presented for OTIS-Cube networks. The OSEOEM algorithm has shown to be an efficient in redistributing the load balancing for OTIS-Star networks, the outcome resulted an equal distribution of loads among all nodes within the network.*

Keywords: Electronic Interconnection networks, Optical networks, Load balancing, Parallel Algorithms, OTIS-Star network.

1 Introduction

The Star graph was proposed by Akers and et al as one of the attractive topologies, it was proposed as an alternative to the cube network [1]. The Star graph has excellent topological properties when we compare it with network similar sizes [2]. The star graph showed to have many attractive properties over many networks including the well-known cube network including: smaller diameter, smaller degree, and smaller average diameter. The star graph proved to have a hierarchical structure which will enable it building large network size of smaller ones, the star graph has both edge and vertex symmetry [1,2].

With the new advances of technology, new era of Optical networks has been appeared. Many previous researches addressed the emerge of the Optical technology with the traditional electronic interconnection topology. OTIS-Star was one of the proposed networks in this era due to its attractive properties and features[3].

Although some algorithms proposed for the OTIS-Star graph such as routing algorithms and distributed fault-tolerant routing algorithm [3, 4], still there is a shortage of efforts to solve the problem of Load Balancing using OTIS-Star networks.

To our knowledge there is no enough results proposed in literature about implementing and proposing efficient algorithms for load balancing on OTIS-Star network. In this paper we are filling this gap by proposing and embedding the **OSEOEM** algorithm on the OTIS-Star graph which is based on the FOFEM algorithm which was shown to be efficient on OTIS-Cube networks [5]. The main mechanism of this algorithm is to redistribute the load equally among the processors of the network [6]. Efficient implementation of the **OSEOEM** algorithm on the OTIS-Star network will make this network more suitable network for real life application in connection to load balancing problem. The rest of the paper is organized as follows: In section 2 we present the necessary basic notations and definitions, in section 3 we introduce some of the related work on load balancing, in section 4 we present and discuss the implementation of the **OSEOEM** algorithm on the OTIS-Star graph, also we present an example of **OSEOEM** on OTIS-3-star network, finally section 5 concludes this paper.

2 Definitions and Basic Properties

During the last decade a numerous number of interconnection networks for Parallel Computers have been proposed in literature [7, 9]. Some of these networks are the star and the hypercube interconnection network. The OTIS-*star graph* [1,3] is another example, which has been proposed as an attractive alternative to its factor network [1]. Since its appearance the OTIS-star network has attracted a lot of research efforts, many topological properties of the OTIS-star network have been investigated in the literature including its basic topological properties [1], parallel path [8], load balancing algorithm [6] and embedding [10]. Sadi and Awwad [3] have employed the good properties star graph and its lead over the properties of similar network to present an efficient fault-tolerant routing algorithm and also a load balancing algorithm. These properties are including a lower degree, a smaller diameter, smaller average diameter, and Symantec architecture.

The structure of the OTIS-star network plays an effective step for proposing any algorithms for it. The authors in [8] have shown that the OTIS-*n*-Star graph may be seen as $n! \times n!$, where the symbol (!) means the factorial of an number, and each node labeled with a unique permutation on $\langle n \rangle = \{1,\ldots,n\}$ of the factor star network.

However, there has been relatively a limited research efforts have been dedicated to design efficient algorithms for the OTIS-star graph including routing and Broadcasting algorithms [11, 12], broadcasting [13] and load balancing [14], [5]. In an attempt to overcome this problem we present an efficient algorithm for redistribute the load balancing problem on the OTIS-Star network to redistribute the load balancing among all processors of the network equally as possible.

The topological properties of the OTIS-Star network along with those of the star network are discussed and derived below. These topological properties have been derived using the theoretical framework for analyzing topological properties of the OTIS-Networks in general which was proposed by Al-Ayyoub and Day [2], beside other related research work [11].

In this paper we will refer to *g* as the group address and *p* as the processor address. An intergroup edge of the form ($\langle g, p \rangle$, $\langle p, g \rangle$) represents an optical link and will be referred to as OTIS or optical move.

An OTIS based network contains N^2 nodes partitioned into *N* groups with *N* nodes each. A node is indexed by a pair $\langle x, y \rangle$, $0 \leq x, y < N$ where *x* is the group index and *y* is the processor index. Nodes within the same group are connected by a certain interconnecting topology; while inter-group links are achieved by transposing group and node indexes.

The OTIS-network is constructed by "multiplying" a known topology by itself. The vertex set is equal to the *Cartesian* product on the vertex set in the factor network. The edge set consists of edges from the factor network and new edges called the *transpose* edges. The formal definition of the OTIS-*n*-Star graph is given below.

Definition* 1*: The OTIS-*n*-Star Graph, which is denoted by OTIS-*n*-Star has *n*!x*n*! nodes each labeled with a sole permutation $\langle n \rangle = \{1 \ldots n,\ 1 \ldots n\}$. The first part of the label refers to the group address and the second part refers to the factor address of the node within the group. Any two nodes of OTIS-*n*-Star are connected if, and only if, their corresponding permutations differ exactly in the first position and any other position.

***Definition* 2:** Let *n*-Star = (V_0, E_0) be an undirected star graph representing a factor network. The OTIS-*n*-Star = (V, E) network is represented by an undirected graph obtained from *n*-Star as follows $V = \{\langle x, y \rangle \mid x, y \in V_0$ such that $x \neq y\}$ and $E = \{(\langle x, y \rangle, \langle x, z \rangle) \mid$ if $(y, z) \in E_0\} \cup \{(\langle x, y \rangle, \langle y, x \rangle) \mid x, y \in V_0$ such that $x \neq y\}$.

From the above definitions the set of edges *E* consists of two subsets, one is from *n*-star, called *n*-star type edges, and the other subset contains the *transpose* OTIS edges. The OTIS approach suggests implementing *n*-star edges by electronic links since they involve intra-chip short links and implementing transpose edges by free space optics. Throughout this paper, the terms "*electronic move*" and the "*OTIS move*" (or "*optical move*") will be used to refer to data transmission based on electronic and optical technologies, respectively.

Fig. 1 shows the OTIS-*3*-Star graph with 6 groups; 3!=6; each containing 6 vertices (i.e. six copies of 3-star graphs).

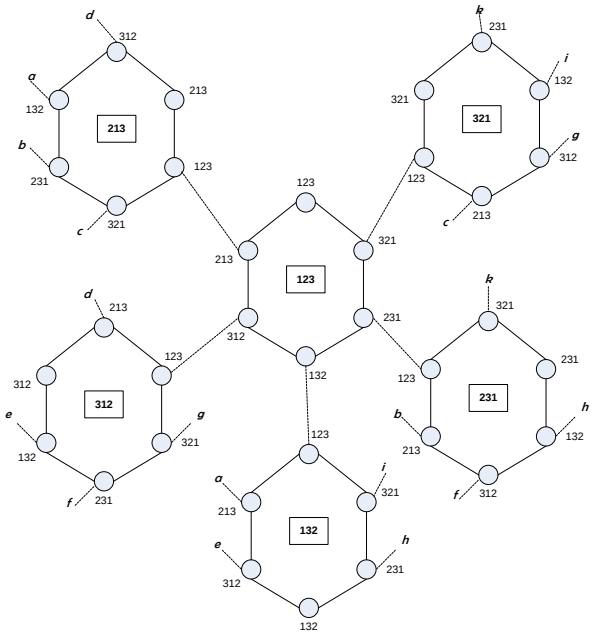


Fig. 1: The OTIS-3-Star Graph

3 Background and Related Work

The attractive properties of OTIS-*n*-Star proved by researchers in literature of the graph make it a one of the strongest competitor's topology for High Speed Parallel Computers (HSPC) and a strong candidate network for real life applications [1, 2]. This fact has motivated us to investigate the load balancing problem on the OTIS-*n*-Star network since the OTIS-*n*-Star graph suffers from limited number of efficient algorithms proposed for it in general and for the Load Balancing problem in specific. The load balancing problem has been investigated on various types of infrastructure ranging from electronic networks [5] and OTIS networks [6].

Load balancing problem is one of the well-known and important types of problems which were studied from different point views and different approaches. This problem was studied and investigated by Ranka, Won, and Sahni [15], they proposed and introduced the Dimension Exchange Method (DEM) on the hypercube topology. The DEM algorithm was based on the idea of finding the average load of neighbours' nodes, such that the dimension of the network is *n*, the neighbours which are connected on the n^{th} dimension they will exchange their loads to redistribute the load and achieve evenly load balancing as possible, the processor which have

more load will broadcast the extra amount of the load to its direct neighbour node. The main advantage of the Dimension Exchange Method is that every node will be able to redistribute tasks to its direct neighbours to reach even load balancing among all nodes. Ranka and *et al* have achieved that in the worst case in the DEM method to redistribute load balancing was log2n on the cube network [15].

The researchers Zaho, Xiao, and Qin have presented hybrid scheme of diffusion and dimension exchange called DED-X for load balancing on Optical Transpose Interconnection System (OTIS) [16] , [17]. The proposed algorithm works by dividing the load balancing task to three different phases. The results achieved on OTIS networks showed that the load balance efficiently redistributed almost evenly. On the other hand the achieved results of the simulation from Zaho et al of the proposed algorithms on load balancing has shown a considerably major advancement in enhancement of efficiency and stability [16], [17]. In another research done by Zaho and Xiao they have presented different DED-X schemes for load balancing on homogeneous OTIS networks and they proposed new algorithm structure called Generalized Diffusion-Exchange- Diffusion Method, the proposed scheme enabled load balancing on Heterogeneous OTIS networks [18].

Another algorithm presented in [6] called SCDEM is based on the Clustered Dimension Exchange Method CDEM for load balancing for Optical Transpose Interconnection system on Hypercube factor network [6]. The worst time case complexity of CDEM for load balancing on OTIS-Hypercube was O(Sqrt(p)*M log2 p). Also the number of communication steps which is required by CDEM proved to be 3log2 p [6].
Furthermore authors of [5] presented a new load balancing algorithm for OTIS-Cube networks, they have shown that the usability of the new proposed load balancing methods to be better than the DED-X traditional load balancing algorithm [18].

4 The Proposed Algorithm

The main objective of this paper is to propose and present a new load balancing algorithm for the OTIS-*n*-Star networks named OSEOEM based on the algorithm of FOFEM presented in [5].

The main achievement of the new presented OSEOEM is to obtain even load balancing for the OTIS- *n*-Star network by redistributing number of tasks between different nodes on different groups. The number of exchanges needed between different nodes in the OSEOEM is $2(n!)+1$, where n is the diameter of factor network; *n*-Star. Fig. 2 presents the OSEOEM algorithm for load balancing problem on OTIS- *3*-Star network

The OSEOEM load balancing algorithm is based on the following 3 phases:

- Phase 1: The main aim of this phase is to balance the load of all nodes within each factor group of the network. The 1st stage of this phase is achieved by redistributing the load balancing of all direct neighbour nodes that their corresponding permutations differ exactly in the 1st and 2nd position.

The 2nd stage is to redistribute the load balancing of any two neighbour nodes that their corresponding permutations differ exactly in the first and 3rd position.
In general we continue redistributing the load balancing of any two neighbour nodes that their corresponding permutations differ exactly in the first and i^{th} position, where $2 \leq i \leq n$. This process of phase 1 should be repeated *2* times to achieve an optimal equal redistribution of load among all nodes within each group of the factor network.

- Phase 2: This phase will exchange the load of every two nodes connected via an optical link. This will yield to redistribute the load equally among all groups but it also will have different node load within the group itself which leads to the necessity of performing the third phase.
- Phase 3: repeat phase 1 to achieve an equal distribution of load among all nodes of the OTIS-*n*-Star

Note that *n*-1 is the number of neighbors' of any processor in the factor network *Sn*:
Figure 3 illustrates the three phases of OSEOEM in details:

1. *for* $m = 2; m \leq n; m{+}{+}$ // start of phase 1
2. *for* all *n*-1neighbour nodes; p_i and p_j which they differ in 1st and *m* position of S_n do in parallel
3. Give-and-take p_i and p_j total load sizes of the two nodes
4. *TheAverageLoad* $p_{i,j}$ = Floor ((*Load* p_i + *Load* p_i)/2)
5. *if* (*Totalload* p_i >= excess *AverageLoad* $p_{i,j}$)
6. *Send* excess load p_i to the neighbour node p_i
7. Load p_i = Load p_i – extra load
8. Load p_j = Load p_j + extra load
9. *else*
10. Receive extra load from neighbour p_j
11. Load p_i = Load p_i + extra load
12. Load p_j = Load p_j – extra load
13. Repeat steps 3 to 12 one more time // end of phase 1
14. *for* all adjacent nodes via an optical link, exchange the loads of the nodes // phase 2
15. Redistribute the weight within each group by repeating steps 1 – 13 // phase 3

Fig. 2: The OSEOEM load balancing Algorithm

OSEOEM algorithm works on redistributing the load balancing among all nodes of the network, phases: one, two and three are described as follows:

- Phase 1: The load balancing between the nodes of S_n based on OSEOEM algorithm is exchanged as in steps 2 to 12 in parallel, in first step the load exchange will be between all the

nodes in which they differ in 1st position and 2nd position for all the factor networks of S_n. Then the same process will be repeated continually until it reaches the neighbours p_j that are n-1 positions far away from p_i. To enhance the load balancing efficiency between different processors of n factor networks, the algorithm suggests repeating the steps 2 to 12 as mentioned above

- Phase 2: To redistribute the loads among all the groups of the network, every two nodes that are connected via an optical link will exchange their loads. This step is conducted on parallel.
- Phase 3: As a final phase all adjacent nodes which they differ in first position and any other position i.e. p_i and p_j. The algorithm will redistribute the weights among the nodes p_i and p_j. Basically, this phase is a repetition of phase 1 to rebalance the weights of all nodes within each group one more time.

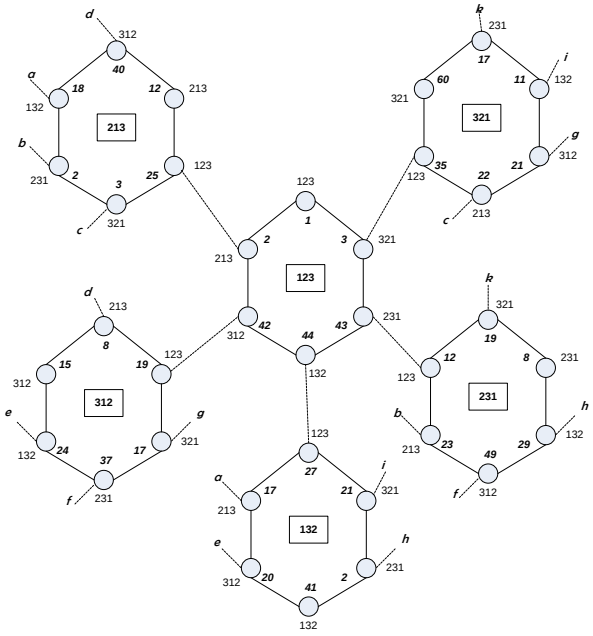


Fig. 3: OTIS-3-Star network – load balancing - initial state

Example: - To explain the OSEOEM algorithm which presented in Fig. 2, the following example implements the load balancing algorithm on the OTIS-3-Star where the factor network is S_3.

Fig. 3 shows the OTIS-3-Star network, each factor network S_3 has 6 nodes with a specific load assigned to each one. Each node connected to two electronic direct nodes within the group, in addition to a third optical connection to another group. Note that the number in bold and italic which is assigned next to each node represents the starting load.

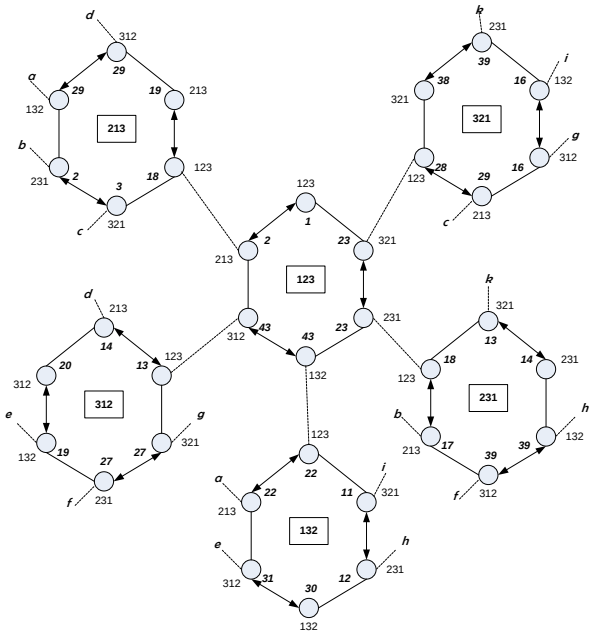


Fig. 4: OTIS-3-Star network – load balancing phase 1 step 1

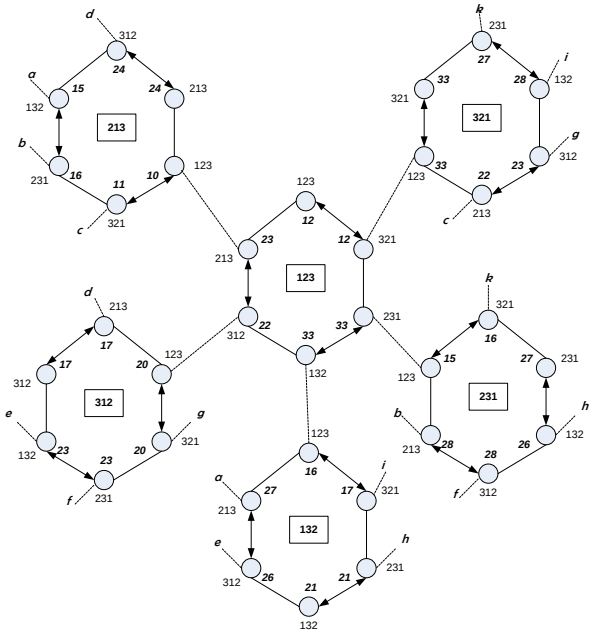


Fig. 5: OTIS-3-Star network – load balancing phase 1 step 2

First we start by implanting phase 1 of the OSEOEM algorithm by following the steps 2-12. The figures 4, 5 and 6 reflect phase1: of the OSEOEM algorithm, each pair of the nodes which they differ in 1st position, 2nd, 3rd and 4th position. At the end of this phase Fig. 6 shows the new load balancing distribution of the phase 1 of the algorithm. Finally in phase 3 outcome is shown in figure 11 where all nodes will

redistribute their load balancing by repeating phase 1 stages, this figure shows that all nodes will have almost the same load approximately.

The new achieved distribution of the load balance shown to be efficient and optimal. The final distribution is achieved in $2(n!)+1$ communication steps where n is the degree of the OTIS-n-star network.

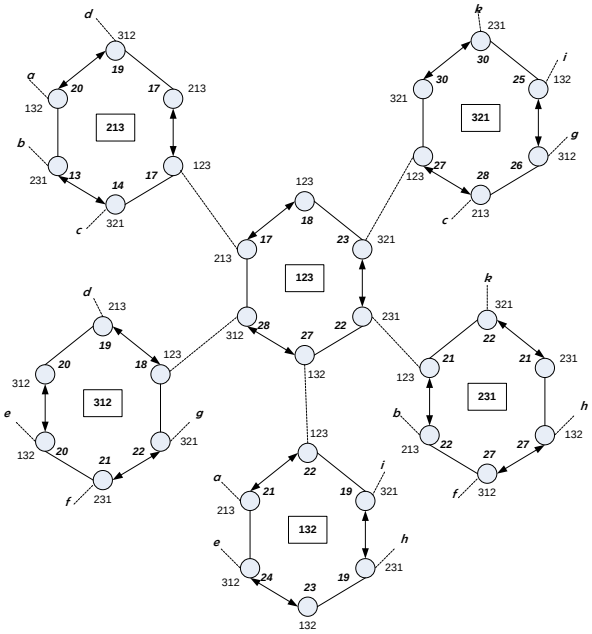


Fig. 6: OTIS-3-Star network – load balancing phase 1 step 3

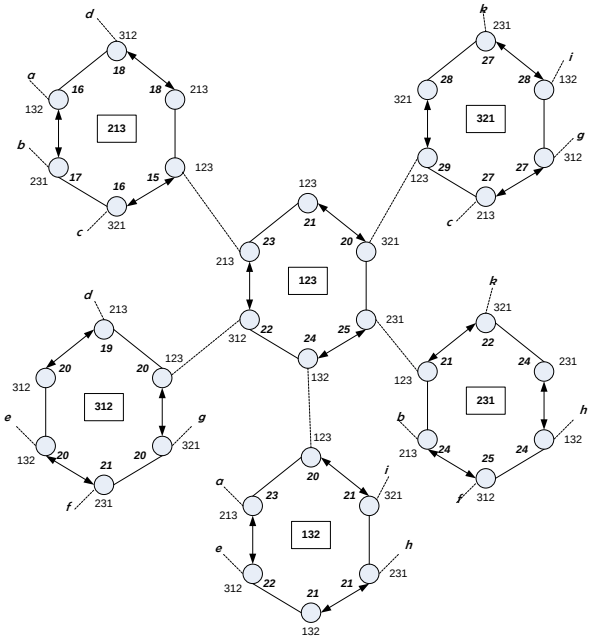


Fig. 7:OTIS-3-Star– load balancing phase1 step 4

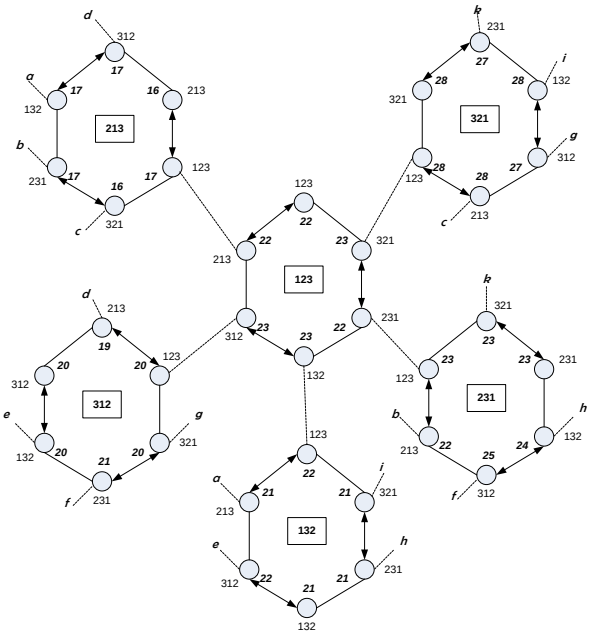


Fig. 8: OTIS-3-Star network – load balancing phase 1 step 5

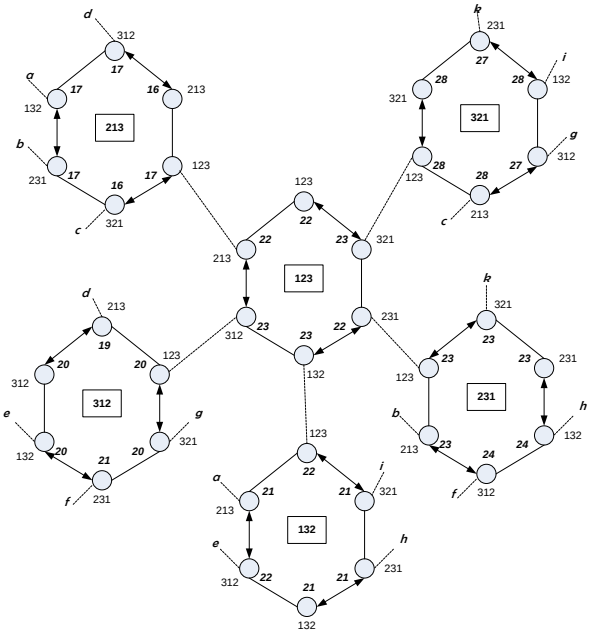


Fig. 9: OTIS-3-Star network – load balancing phase 1 step 6

In phase 2 we repeat the same steps one more time to redistribute the load balancing among the neighbours' nodes as suggested in OSEOEM algorithm presented in Fig. 2 by Phase 1 has n! stages (steps 2-12 in the algoritms), Figures 4-9 illustrate these stages that have been disrobed in the algorithm above.

Figure 10 illustrates phase 2 where nodes exchange their weights via optical links.

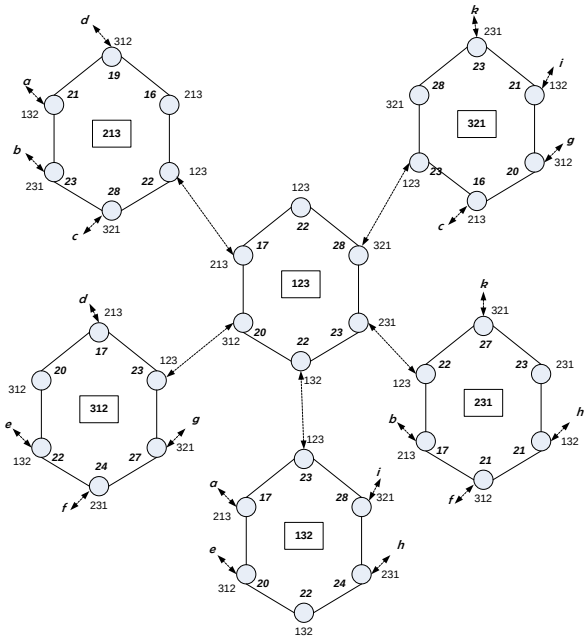


Fig. 10: OTIS-3-Star network – load balancing phase 2

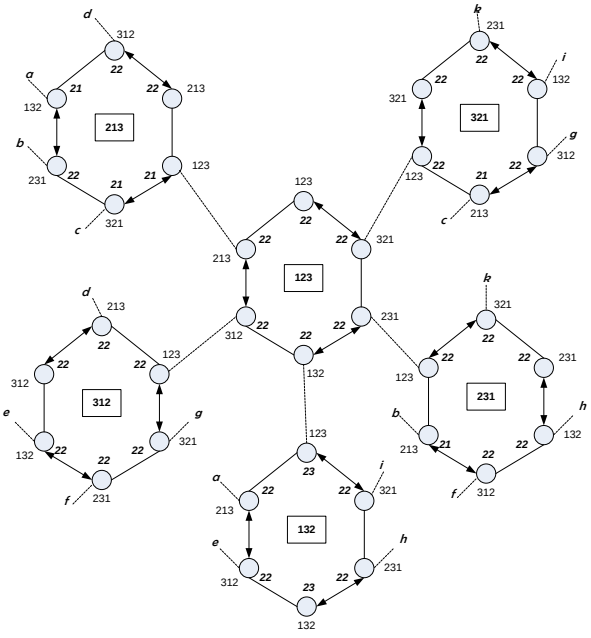


Fig. 11: OTIS-3-Star network – load balancing- end of phase 3

5 Conclusions

This paper presented an efficient algorithm for distributing the load balancing of nodes in OTIS-*n*-star network. The proposed algorithm is called OSEOEM which is based on the well-known algorithm FOFEM method presented for OTIS-Cube networks. The proposed algorithm OSEOEM resulted in almost an even redistributions load balancing among the all nodes of OTIS-*n*-star network. The algorithm is able to redistribute load balancing among all nodes in $2(n!)+1$ communication steps which is considered to be efficient.

As future extension of this research work we will do some analytical estimation and analysis including: execution time, load balancing accuracy, communication steps and speed to prove the OSEOEM is efficient mathematically.

References

[1] S. B. Akers, D. Harel and B. Krishnamurthy, "The Star Graph: An Attractive Alternative to the n-Cube" *Proc. Intl. Conf. Parallel Processing*, 1987, pp. 393-400.

[2] K. Day and A. Tripathi, "A Comparative Study of Topological Properties of Hypercubes and Star Graphs", IEEE Trans. Parallel & Distributed Systems, vol. 5.

[3] J. AL-Sadi, A. M. Awwad, B. F. AlBdaiwi, Efficient Routing Algorithm on OTIS-Star Network, Proceedings of the IASTED International Conference on Advances in Computer Science and Technology, Virgin Islands USA, November 22-24, 2004, ACTA Press, pp. 157 – 162.

[4] Khaled Day and Abdel-Elah Al-Ayyoub, "Node-ranking schemes for the star networks", Journal of parallel and Distributed Computing, Vol. 63 issue 3, March 2003, pp 239-250.

[5] Jehad Al-Sadi, "Implementing FEFOM load balancing algorithm on the enhanced OTIS-n-Cube topology", The Second International Conference on Advances in Electronic Devices and Circuits - EDC2013, Kuala Lumpur, Malaysia, May, 2013.

[6] B.A. Mahafzah and B.A. Jaradat, "The Load Balancing problem in OTIS-Hypercube Interconnection Network", J. of Supercomputing (2008) 46, 276-297.

[7] S. B. Akers, and B. Krishnamurthy, "A Group Theoretic Model for Symmetric Interconnection Networks," *Proc. Intl. Conf. Parallel Proc.*, 1986, pp. 216-223.

[8] K. Day and A. Al-Ayyoub, "The Cross Product of Interconnection Networks", *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 2, Feb. 1997, pp. 109-118.

[9] A. Al-Ayyoub and K. Day, "A Comparative Study of Cartesian Product Networks", *Proc. of the Intl. Conf. on Parallel and Distributed Processing: Techniques and Applications*, vol. I, August 9-11, 1996, Sunnyvale, CA, USA, pp. 387-390.

[10] I. Jung and J. Chang, "Embedding Complete Binary Trees in Star Graphs," *Journal of the Korea Information Science Society*, vol. 21, no. 2, 1994, pp. 407-415.

[11] Berthome, P., A. Ferreira, and S. Perennes, "Optimal Information Dissemination in Star and Panckae Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 12, Aug. 1996, pp. 1292-1300.

[12] P. Fragopoulou and S. Akl, "A Parallel Algorithm for Computing Fourier Transforms on the Star Graph," *IEEE Trans. Parallel & Distributed Systems*, vol. 5, no. 5, 1994, pp. 525-31.

[13] Mendia V. and D. Sarkar, "Optimal Broadcasting on the Star Graph," *IEEE Trans. Parallel and Distributed Systems*, Vo;. 3, No. 4, 1992, pp. 389-396.

[14] N. Imani et al, "Perfect load balancing on star interconnection network", J. of supercomputers, Volume 41 Issue 3, September 2007. pp. 269 – 286.

[15] Ranka, Y. Won, S. Sahni, "Programming a Hypercube Multicomputer", IEEE Software, 5 (5): 69 – 77, 1998.

[16] Zhao C, Xiao W, Qin Y (2007), "Hybrid diffusion schemes for load balancing on OTIS networks", In: ICA3PP, pp 421–432

[17] G. Marsden, P. Marchand, P. Harvey, and S. Esener, "Optical Transpose Interconnection System Architecture," *Optics Letters*, 18(13), 1993, pp. 1083-1085.

[18] Qin Y, Xiao W, Zhao C (2007), "GDED-X schemes for load balancing on heterogeneous OTIS networks", In: ICA3PP, pp 482–492.

SESSION

SOFTWARE TOOLS AND SYSTEMS, PARALLELIZING COMPILERS, PROGRAMMING LANGUAGES, OS, AND MIDDLEWARE

Chair(s)

TBA

Hotspot: a Framework to Support Performance Optimization on Multiprocessors

Fernando G. Tinetti[1], **Andres More**[2]
[1]III-LIDI, Fac. De Informática, UNLP
Comisión de Inv. Científicas Prov. de Bs. As.
1900, La Plata, Argentina
[2]Intel Corp., Argentina Software Design Center
5000, Córdoba, Argentina

Abstract— *High Performance Computing (HPC) programs are usually developed by domain specialists, and they are not always experts on performance optimization. Application domain specialists rely on a systematic and unattended (and/or partially guided) method to support performance analysis and optimization. This paper presents a framework that simplifies the job, including several case studies as a validation step and as a proof of concept. The main goal is to make the data recollection task straight-forward, allowing to focus on experimentation and optimization. The framework gathers context information about the program and the underlying system, detailing scaling behavior, execution profile, resource utilization, and bottlenecks. In a nutshell, this work contributes with a performance report generator for OpenMP programs.*

Keywords: Computer Aided Analysis, Performance Analysis, High Performance Computing, Parallel Processing

1. Introduction

High Performance Computing (HPC) optimized code runs much faster than a naïve implementation [1]. This implies that investing on program parallelization and optimization leads to large gains in productivity for processing intensive programs, which usually require multiple runs to simulate or solve a problem.

Parallel programming is used to fully take advantage of available computing power. However, parallelization increases complexity, debugging, and performance evaluation [2] [8]. The optimization and parallelization tasks are usually performed ad-hoc, without knowledge of available tools and their capabilities. Also, lacking the use of quantitative information to direct optimizations is another (not minor) problem underlying such parallelization and optimization work. Many times, well-known algorithms are (re)implemented instead of using already heavily optimized libraries, with proven correctness and a supporting community.

This paper is focused on simplifying the tedious and error-prone task of performance analysis. The methodology implies repetitive execution of multiple executions under different input conditions, taking advantage of the expertise on different tools to instrument behavior and understand resource utilization internals. The framework runs benchmarks on the system, apply profiling tools, and graphically show results into a final detailed report with statistical data on scaling and bottlenecks.

The rest of this paper is structured as follows: in Section 2 we introduce performance analysis background and related work. Section 3 the problem we aim to solve is reviewed, and in Section 4 we cover our proposed solution and the implemented framework. Finally, in Section 5, we apply the framework to several well-known computing kernels before concluding remarks are discussed.

1.1 Related Work

Performance optimization is relevant to almost any discipline involving computing processing. An introduction can be found in [11], a general revision in [12], just to mention a few. A proposal of the direction of the state-of-the-art in [13]. Useful reusable design patterns in [14]. Several ideas for gathering relevant performance information can be found in [15], and an effort on automatic optimization in [16]. Further details on the state-of-the-art can be found in [17].

2. Performance Analysis

Performance is characterized by a metric which must be quantifiable units of achieved work in contrast with utilized time and resources. Metrics allows the relative comparison of systems, and also understand a reconfigured system behavior.

There are several laws that provide some guidance for estimating performance gains when incrementing computing resources. The so called Amdahl Law [6] provides some insight in potential speedup according to the serial and parallel parts of an program. Gustafson [7] establishes something similar, but taking into account how many times we can compute the same problem and, also, the fact that increasing processing facilities implies increasing problem size (thus, also increasing processing requirements). A *raw* representation on how parallelism ratios impact speedup when doubling processing units is shown in Fig. 1.

The procedure to perform optimizations involves cycles of measurement, bottleneck identification, optimization, and

	0.1	0.3	0.5	0.8	0.9	0.95
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.05	1.14	1.33	1.60	1.82	1.90
4	1.08	1.23	1.60	2.29	3.08	3.48
8	1.10	1.28	1.78	2.91	4.71	5.93
16	1.10	1.31	1.88	3.37	6.40	9.14
32	1.11	1.32	1.94	3.66	7.80	12.55
64	1.11	1.33	1.97	3.82	8.77	15.42
128	1.11	1.33	1.98	3.91	9.34	17.41
256	1.11	1.33	1.99	3.95	9.66	18.62
512	1.11	1.33	2.00	3.98	9.83	19.28
1024	1.11	1.33	2.00	3.99	9.91	19.64
2048	1.11	1.33	2.00	3.99	9.96	19.82
4096	1.11	1.33	2.00	4.00	9.98	19.91
8192	1.11	1.33	2.00	4.00	9.99	19.95
16384	1.11	1.33	2.00	4.00	9.99	19.98
32768	1.11	1.33	2.00	4.00	10.00	19.99
65536	1.11	1.33	2.00	4.00	10.00	19.99

	0.1	0.25	0.5	0.75	0.9	0.95
1	1	1	1	1	1	1
2	1	1	2	2	2	2
4	1	2	3	3	4	4
8	2	3	5	6	7	8
16	3	5	9	12	15	15
32	4	9	17	24	29	30
64	7	17	33	48	58	61
128	14	33	65	96	115	122
256	27	65	129	192	231	243
512	52	129	257	384	461	486
1024	103	257	513	768	922	973
2048	206	513	1025	1536	1843	1946
4096	411	1025	2049	3072	3687	3891
8192	820	2049	4097	6144	7373	7782
16384	1639	4097	8193	12288	14746	15565
32768	3278	8193	16385	24576	29491	31130
65536	6555	16385	32769	49152	58983	62259

Fig. 1: Speedup Limits According to Amdahl (left) and Gustafson (right)

gain comparison and analysis. Every decision should be made using strongly meaningful data in order to focus the work on maximizing impact. In the case the measurement of a metric has deviations, it is required to take several samples and use an average to avoid transient noise. Usually, a longer execution time of a problem will help to stabilize results. If the system has dynamic configuration, then results reproduction is non-trivial. On this case, it is advisable to run together old and new versions of the program for a direct comparison.

2.1 Performance Tools

There are many available tools for performance analysis [8]. Tools work at different levels of abstraction: from hardware-based event counters, to resource monitoring inside operating system kernels, code and binary instrumentation, up to the simple utilization of runtime as a reference metric.

Benchmarks are specially-built synthetic programs, or even a predefined set of real-world programs that provide a reference figure to compare performance. The required features for a benchmark are portability, simplicity, stability and results reproduction. It is also required that execution time is reasonable, and problem size can be adjusted to keep applicability over time and the evolution of technologies. High Performance Computing Challenge (HPCC) [21] is a package that includes several HPC benchmarks and provides multiple metrics at once.

An incremental approach is proposed in this paper, applying tools in the sequence shown in Table 1, every step adding more information and detail to the analysis. The overall system capacity extracted using the HPCC benchmarks allows to set a practical limit on performance of the system being used. The timing of workload executions allows to understand their runtime deviation. The execution profile will show call flows and bottlenecks at function, source code line and assembly levels. The program scaling behavior shows speedup trends when incrementing either problem size or processing units. A system profile show impact on

Table 1: Performance Tools Incremental Application.

Information	Tools
Overall System Capacity	HPCC Benchmark
Workload Execution	time, gettimeofday()
Execution Profile	gprof, perf
Program Scaling	gprof, perf
System Profile	Perf
Vectorization	Compilers
Hardware Counters	Perf

available system resources. At last, low-level reports on vectorization details and hardware counter status can lead to fine-tuning needs at CPU instruction level.

3. Optimization Problems

The approach in this paper identifies three different dimensions in which the problems have to be addressed: performance analysis, optimization methods implementation, and supporting infrastructure.

3.1 Performance Analysis

The performance analysis has multiple challenges:

- Human interaction is always a source of involuntary mistakes. Miss investing valuable time in tasks that can be automated. Usually, one person should run tests, gather results, and draw charts in a performance report to rely on during subsequent analysis. Absolute discipline is mandatory as the main time-consuming task is to execute the same program under different configurations to record its behavior.
- Required expertise in tools. Learning about the proper use of the multiple tools takes considerable time. However, those tools povide high quality information necessry to take data-driven decisions at the time of optimization efforts. The analysis requires the correct use of statistics to average results, discard noise and outliers, and understand limits on potential improvements.
- Data gathering and representation of results. The analysis requires the gathering of supporting data about both the program and the system being analyzed. Idetifying metrics, how to get them and even how to represent them is non-trivial and requires time and expertise.
- Early optimization. An optimization made in the wrong place implies wasted effort and potentially little impact on the overall execution runtime of the program.
- Naïve implementation of algorithms. Direct implementation of an algorithm may guarantee correctness but not efficiency without first understanding underlying low-level details of the computer architecture. Reuse of portable math libraries guarantee both correctness and performance with only a minimum effort on learning how to apply them.

And many of them (maybe all) are interraleted, so they cannot be treated indepently of each other.

3.2 Optimization Methods Implementation

The usual optimization methods target different aspects of a given program:

- Code. The source code is analyzed to improve jump prediction by pre-fetching units, in-lining frequently used routines to avoid unnecessary returns, align code and unroll loop to make program steps easy to map into vectorised instructions, among many other aspects.
- Execution. The generated assembly level code is inspected to verify the usage of lightweight and vectorised instructions, trying also to reduce the number of used registers.
- Memory. Program structures are analyzed to minimize cache failures, align inner components and improve the locality of data to enhance the memory hierarchy performance.
- Prefetching. Use of pre-fetching instructions are analyzed to help prediction unit to be more effective.
- Floating point arithmetic. Given a problem it can be considered to relax representation assumptions following standards, in order to exchange accumulated error and final precision by processing speed.

And, again (as in the case of the performance analysis explained above), many of them (maybe all) are interraleted, so they cannot be treated indepently of each other.

3.3 Supporting Infrastructure

Clearly, performance analysis needs (semi)automated support, the following are some requirements that any framework should include:

- Reusability. The framework should be applicable to a wide range of programs, without requiring its modification. Its deployment should only depend on the same tools that any user may need to gather performance-related information manually.
- Configuration. The framework should be configurable and parameters should include how to compile and execute the program, the input range to consider, and the number of repetitions to check for stability, among other details.
- Portability. The framework should be implemented in a portable (and maybe interpreted) language, to allow easy review and the expansion with new tools, new data gathering experiments or the modification on how things are being done.
- Extensibility. The framework should be designed for extension from scratch, incorporating new tools, charts o sections on a final report should be an almost trivial task given the user already knows how to gather the metric manually.
- Simplicity. The framework should reuse the same tools available at system level to a regular user. It should generate log files of all the issued commands and their output so any user can check them if required. It should be possible to use the framework to run overnight and allow incremental optimization.

4. Proposed Solution

We propose an approach that combines a generic procedure on top of an automated framework that takes care of running the program multiple times, applying performance analysis tools, identifying bottlenecks, gathering metrics and representing results graphically into a performance report. This allow the user to solely focus on optimization of the identified bottlenecks, freeing him of performance-related data recollection work.

4.1 Method

The performance analysis procedure needs to be done incrementally to guarantee progress no matter the allowed time for the task. We propose then a process which starts analyzing system computing power. The computing power is taken into account to follow a sequence of improvement cycles of execution of the program for gathering performance metrics and revealing bottlenecks. Thus, optimization efforts can be focused on those identified bottlenecks, in order to maximize overall impact. The following steps reflects how the process should be done:

1) Run the HPCC benchmark to get insights on overall system performance.
2) Run the program multiple times using the same workload to check deviations.
3) Establish a baseline using geometric mean to avoid measurement noise.
4) Run the program scaling problem size to dimension its behavior.
5) Run the program scaling computing resources to dimension its behavior.
6) Extract execution profile of the program
7) Extract system profile while running the program.

4.2 Framework

A framework named hotspot (https://github.com/more-andres/hotspot) was built, implementing the previous procedure. The automation behaves exactly like a regular user running commands and checking their output. It executes tools like gcc, make, prof, gprof, pidstat (among others) to gather relevant information and finally using the Latex typesetting environment to compile a human-friendly report including the data and associated charts.

Currently, only GNU/Linux systems with kernels above 2.6.32 and OpenMP threaded programs are supported out-of-the-box. Old kernel versions do not properly support the tools used to identify bottlenecks at assembly level. OpenMP threaded programs easily let to change the number of processing units in order to understand scaling behavior.

At a lower abstraction level, the framework is designed with two simple object class hierarchies as shown in Fig. 2. The first one keeps the main engine which runs the second one as independent sections that gather information and report back metrics. The latter one can be extended to add more sections to the final report. The implementation

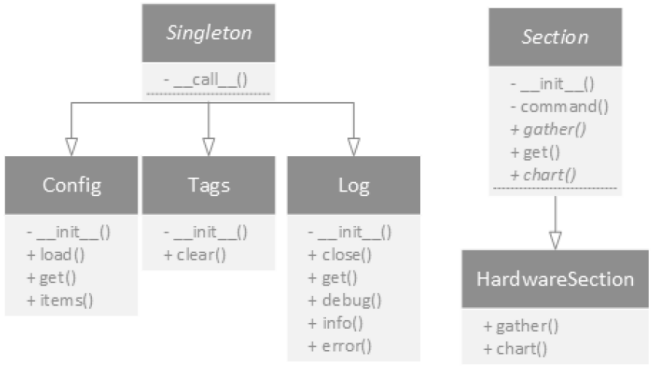


Fig. 2: Speedup Limits According to Amdahl (left) and Gustafson (right)

defines multiple objects as part of these class hierarchies. The components of the framework engine are:

- Singleton: extended by other object to guarantee unique instance.
- Tags: storage of keywords to be replaced at the report template.
- Log: manager used to structure output messages.
- Config: manager used to read and share configuration attributes.

And the components that hold the responsibility and knowledge of running experiments and gathering relevant metrics are the following:

- Section: extended by sections to be included in the report.
- HardwareSection: hardware-related information such as used CPU and Memory.
- ProgramSection: program-related information such as used problem input range.
- SoftwareSection: program-related information such as used compiler and libraries.
- SanitySection: basic check on that the workload can run without issues.
- BenchmarkSection: runs HPCC and gather relevant metrics.
- WorkloadSection: reports on program footprint and its stability.
- ScalingSection: reports on problem size scalability.
- ThreadsSection: reports on computing scalability.
- OptimizationSection: reports on program behavior under compiler optimizations.
- ProfileSection: reports on program execution profile, call flow and bottlenecks.
- ResourcesSection: reports on the use of system resources while the program runs.
- AnnotatedSection: reports annotated bottlenecks mapping code to assembly.
- VectorizationSection: reports loops being vectorised or not and the impediments
- CountersSection: reports hardware counter status.
- ConfigSection: reports used configuration for the overall framework execution.

The framework uses a per-program hidden directory to keep record on executions, classified per timestamp and caching the different results to avoid long waiting times.

A configuration file is used to define the framework parameters to handle any program. It is important to note that the framework needs to know how to run, compile and instrument each program as part of the configuration. The framwork will rely on variables that hold problem input size, numbers of OpenMP threads to use and even compiler flags to instrument binaries.

4.3 Operation and Report

At high level the design mimics how a user manually interacts with the system. The framework depends on tools available on the system, the program, and its matching configuration plus the LaTeX typesetting system.

After reading the configuration, the workload is executed multiple times and its wall time is checked for stability by charting a histogram using as a baseline a normally distributed curve. The geometric mean is extract to be used as a reference in future executions. As second step, the program is executed over the full range of input size and available computing resources, with this scaling charts are generated, plotting ideal scaling as a comparison approach as well. Using this scaling information the potential speedups are computed according to Amdahl and Gustafson laws. Then using expanded debugging information bottlenecks are identified on logical, source and assembly program levels. One last execution is done while at the same time resource usage is recorded to understand system bottlenecks. After all these executions the gathered information is used to generate a detailed PDF report to support the performance optimization task.

The general considerations followed when building the report include:

- Format similar to a scientific paper.
- Hyperlinks to full logs of tools output.
- Brief explanation of each section and chart objective.
- Inclusion of ideal trends and behavior in charts.
- References to base bibliography.

Several pieces of information are given, such as: a) Abstract: a brief summary introducing the framework and the location of the supporting output for detailed review, b) Content: reduced table of content with direct links to the information, c) Program: details of the program under analysis, timestamp

of the analysis and input parameters used during tests, d) System computing poer: beyond specifics about hardware and software on the system, the framework includes the metric reported by an HPCC execution, e) Workload: working set and in-memory structures, histogram of execution to understand its deviation and the geometric mean used as a baseline. Also a chart checking how the compiler optimization levels improve time.

5. Case Study - Examples

This section will showcase the hints that the framework can lead to, but will not include solutions to those as they are out-of-scope. The framework was used to analyze three well-known compute kernels: a) Mtrix Multiplication: a naïve implementation of dense matrix multiplication., b) 2D Heat Distribution: a naïve implementation of iterative heat distribution, and c) Mandelbrot Set: a naïve implementation of a recursive fractal algorithm.

The HPCC benchmarks executed and its multiple reference metrics are included to be used as top reference of system capabilities, shown in Table 2.

Table 2: Performance Metrics Reported by HPCC

Benchmark	Value	Unit
hpl	0.00385977 TFlops	tflops
dgemm	1.03413 GFlops	mflops
ptrans	0.997656 GBs	MB/s
random	0.0274034 GUPs	MB/s
stream	5.19608 MBs	MB/s
fft	1.2658 GFlops	MB/s

5.1 Examples

Fig. 3 shows the initial performance and metrics report provided by our tool. As a first step towards optimization,

Execution time:
(a) problem size range: `1024 - 2048`
(b) geomean: `4.61694` seconds
(c) average: `4.61757` seconds
(d) stddev: `0.07678`
(e) min: `4.52168` seconds
(f) max: `4.73660` seconds
(g) repetitions: 8 times

Fig. 3: Initial Matrix Multiplication Report

the different compiler optimization options are also reported, shown in Fig. 4. We have seen almost the same *pattern* for the different compiler optimization levels on several programs, e.g. a huge improvement for -O1 and -O2, and relatively small improvement for -O3. Fig. 5 shows the information about serial and parallel fractions as well as the limits of parallelization according to de the so called Amdhal and Gustafson laws.

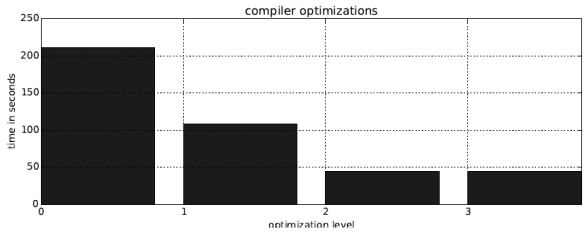


Fig. 4: Compiler Optimizations on Matrix Multiplication

1. Parallel Fraction: `0.52260`.
   Portion of the program doing parallel work.
2. Serial: `0.47740`.
   Portion of the program doing serial work.

Optimization limits can be estimated using scaling laws.

1. Amdhal Law for 1024 procs: `2.09466 times`.
   Optimizations are limited up to this point when scaling problem size. [2]
2. Gustafson Law for 1024 procs: `535.61564 times`.
   Optimizations are limited up to this point when not scaling problem size. [3]

Fig. 5: Matrix Multiplication Parallelization Reports

The tool on the naïve implementation of heat distribution in 2 dimensions identifies:

- Workload is stable, although deviation is greater than in the previous case.
- Compiler optimizations have little impact.
- Execution time does not grow monotonically when scaling input size; it neither decreases when adding more computing units.
- Parallelism only reaches 65%, therefore speedup has a limit on 2.9x.
- There are two main bottlenecks with 30% and 15% of overall execution time.
- CPU utilization is not constant, and loops are not vectorised.

On the third example, the naïve implementation of Mandelbrot sets [23], the tool reports:

- Structures used to represent complex numbers are aligned, without holes that may consume cache memory.
- Workload is stable.
- 50% of the time is spent on the same line of code.
- There are already optimized cycles using the movss and adds vectorised instructions.

6. Conclusions and Further Work

The performance optimization process is not trivial and requires disciplined analysis of used resources and gathering of metrics characterizing program behavior. This works reviews the development of a supporting framework that streamlines the process by running in unattended mode and

generating a report to direct optimization efforts. The generated report combines multiple tools to identify bottlenecks at function, source, and assembly level (e.g. vectorization). It also records system configuration and resource usage. All of this is offered as an unattended automated task before undergoing the program optimization analysis. We hence propose a systematic procedure supported with an automated framework that is suitable for both newcomers and experts. It also allows the exchange of standardized performance reports between research and development groups.

Extension possibilities on this framework are straightforward considering new sections can be added to the report including the application of new tools, charts or contextual information. The application of this framework to a well-known open source program will size its usefulness and provide feedback on both the procedure and the generated report. At last, it may be interesting to move from a static report format to a dynamic one offering pivot tables and charts that can be reconfigured easily. This may be better achieved using HTML5 technologies over a browser for instance.

References

[1] A. More. A Case Study on High Performance Matrix Multiplication. Technical Report, Universidad Nacional de La Plata, 2008. Available at http://mm-matrixmultiplicationtool.googlecode.com/files/mm.pdf

[2] P. E. McKenney. Is Parallel Programming Hard, And, If So, What Can You Do About It? kernel.org, 2010.

[3] Intel Corporation. Intel Math Kernel Library. Reference Manual. Intel Corporation, 2009.

[4] R. Garabato, V. Rosales, A. More. Optimizing Latency in Beowulf Clusters. CLEI Electron. J., 15(3), 2012.

[5] J. Jeffers, J. Reinders. Intel Xeon Phi Coprocessor High Performance Programming. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[6] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In Proceedings of the April 18-20, 1967, spring joint computer conference, AFIPS '67 (Spring), pages 483-485, New York, NY, USA, 1967. ACM.

[7] J. L. Gustafson. Reevaluating Amdahl's Law. Communications of the ACM, 31:532-533, 1988.

[8] A. H. Karp, H. P. Flatt. Measuring parallel processor performance. Commun. ACM, 33(5):539-543, May 1990.

[9] P. J. Fleming, J. J. Wallace. How not to lie with statistics: the correct way to summarize benchmark results. Commun. ACM, 29(3):218-221, March 1986.

[10] K. Atkinson. An Introduction to Numerical Analysis. Wiley, 2 edition, 1989.

[11] C. U. Smith. Introduction to software performance engineering: origins and outstanding problems. In Proceedings of the 7th international conference on Formal methods for performance evaluation, SFM'07, pages 395-428, Berlin, Heidelberg, 2007. Springer-Verlag.

[12] J. C. Browne. A critical overview of computer performance evaluation. In Proceedings of the 2nd international conference on Software engineering, ICSE '76, pages 138-145, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[13] M. Woodside, G. Franks, D. C. Petriu. The future of software performance engineering. In 2007 Future of Software Engineering, FOSE '07, pages 171-187, Washington, DC, USA, 2007. IEEE Computer Society.

[14] T. Mattson, B. Sanders, B. Massingill. Patterns for parallel programming. Addison-Wesley Professional, First edition, 2004.

[15] K. A. Huck, O. Hernandez, V. Bui, S. Chandrasekaran, B. Chapman, A. D. Malony, L. Curfman McInnes, B. Norris. Capturing performance knowledge for automated analysis. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, pages 49:1-49:10, Piscataway, NJ, USA, 2008. IEEE Press.

[16] T. Margalef, J. Jorba, O. Morajko, A. Morajko, E. Luque. Performance analysis and grid computing. In V. Getov, M. Gerndt, A. Hoisie, A. Malony, B. Miller, editors, Performance analysis and grid computing, chapter Different approaches to automatic performance analysis of distributed applications, pages 3-19. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[17] F. Wolf B. Mohr. Automatic performance analysis of hybrid mpi/openmp applications. J. Syst. Archit., 49(10-11):421-439, November 2003.

[18] B. Gregg. Linux Performance Analysis and Tools. Technical report, Joyent, February 2013.

[19] J. D. McCalpin. Memory bandwidth and machine balance in current high perfor-mance computers. IEEE Technical Committee on Computer Architecture (TCCA) Newsletter, Dec 1995.

[20] U. Drepper. What Every Programmer Should Know About Memory. Tech-nical report, Red Hat, November 2007.

[21] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. Mccalpin, D. Bailey, D. Takahashi. Introduction to the HPC Challenge Benchmark Suite. Technical report, 2005.

[22] B. B. Mandelbrot, D. E. Passoja, editors. Fractal aspects of materials: metal and catalyst surfaces, powders and aggregates: extended abstracts, volume E-4 of Materials Research Society extended abstracts, Pittsburgh, PA, USA, 1984. Materials Research Society.

Empirical Study of Time Efficiency and Accuracy of Support Vector Machines Using an Improved Version of PSVM

S. Tavara[12], H. Sundell[1], and A. Dahlbom[2]
[1]Department of Information Technology, University of Borås, Borås, Sweden
[2]School of Informatics, University of Skövde, Skövde, Sweden

***Abstract*—We present a significantly improved implementation of a parallel SVM algorithm (PSVM) together with a comprehensive experimental study. Support Vector Machines (SVM) is one of the most well-known machine learning classification techniques. PSVM employs the Interior Point Method, which is a solver used for SVM problems that has a high potential of parallelism. We improve PSVM regarding its structure and memory management for contemporary processor architectures. We perform a number of experiments and study the impact of the reduced column size p and other important parameters as C and γ on the class-prediction accuracy and training time. The experimental results show that there exists a threshold between the number of computational cores and the training time, and that choosing an appropriate value of p effects the choice of the C and γ parameters as well as the accuracy.**

Keywords: Parallel SVM; Processor Technology; Training Time

I. Introduction

High Performance Computing (HPC) tools are promising for improving performance in respect of time efficiency. As more computational power can be spent on less time, this enables the accuracy of results to be improved as well. The importance of utilizing HPC tools has been growing and parallel computing as the underlying method of HPC plays an important role for improving the time efficiency. Message Passing Interface (MPI) is one of the well-known parallel library standards that was originally designed for distributed memory systems, although it can handle shared and hybrid (combination of shared and distributed) memory architectures as well. The advantages of using the MPI library standard is well-known, albeit the efficiency of the parallel processing can be degraded due to data dependency, memory bandwidth, synchronization and communication bottlenecks.

Digital data is growing exponentially and hence analysis and calculation processes regarding big data are becoming computationally expensive. Within this context, machine learning is one of the fields that can take advantage of using HPC tools for improving the performance. Support Vector Machine (SVM) [17] is one of the classification machine learning techniques that has a wide area of applications and has got considerable attention during the last decade. The SVM problem is set up as a minimization problem and can thereafter be solved using classical optimization algorithms. Interior Point Method (IPM) [18] is a popular choice thanks to the high degree of parallelism inherent in it. However, IPM requires computing the inverse of a matrix which is computationally expensive. Besides, in most of cases the coefficient matrix derived from the system is ill-conditioned, meaning that the matrix is either singular or close to singularity which makes the problem computationally unstable. Therefore approximation and preconditioning methods are applied to prevent the ill-conditioning and to reduce the computational costs.

Cholesky Factorization (CF) [20] is one of the techniques that is used for achieving stable numerical solutions. CF factorizes matrix $A \in R^{n \times n}$ into a lower triangular matrix, i.e., $A = LL^T$, where $L \in R^{n \times n}$. Incomplete Cholesky Factorization (ICF) [20] is a truncated form of CF, i.e., $A = \hat{L}\hat{L}^T$, where $\hat{L}$ is a $n \times p$ sparse lower triangular matrix close to L, where p is the rank of $\hat{L}$. In ICF approximation, only p column vectors are calculated which makes this approximation quick and economical to compute since $p \ll n$. However, calculating the appropriate column rank value, p, is non-trivial. A lower value of p degrades the accuracy and a higher value of p increases the computational time. In this paper, we study the trade-off between the class-prediction accuracy and time efficiency for different p settings and different kernel functions. Furthermore, we study the correlation between the choice of p and the hyperparameters C and the γ value, in respect to the effect of the Gaussian and Laplacian kernels on the class-prediction accuracy and the training time.

The advantage of using distributed parallelism as MPI on SVM problems is well-known, although how to choose the appropriate numbers of computational cores is still unclear and non-trivial. In theory, increasing the number of machines from 1 to 10 will enhance the time efficiency 10 times, although this is way too idealistic. The reason for that is due to data communication, memory bandwidth and synchronization between different machines. In this paper, we investigate when the data communication part overtakes the parallel computation part in SVM, i.e., when increasing the number of cores no longer is beneficial. As Chang et al. [20] mention in their paper, due to communication and synchronization overheads, the speed-up is not linearly increased by increasing the number of cores, and after a specific number of cores the time efficiency even degrades.

In this respect, it is interesting to study the existence of a threshold that can give an idea of the appropriate number of cores. We theoretically show that there exists a threshold that suggests a minimum number of computational cores, while the maximum number of cores depends on the machines used.

In the following sections, we briefly describe SVM using PSVM software. We mention important processor technologies and then describe the preparation for experiments including the complexity analysis of SVM algorithm using PSVM. We continue with the experiments and the results and finally we discuss the obtained results.

II. Support Vector Machines

The basic idea in SVM is to search for a bipartite hyperplane that has a furthest possible distance to the closest training data points from both sides of the hyperplane. In order to find the optimal bipartite hyperplane, a simple SVM problem can be formulated as an primal quadratic optimization problem as following,

$$\begin{aligned} min \quad & P(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi \\ s.t. \quad & y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \quad , i = 1, 2, ..., n \\ & \xi_i > 0 \qquad\qquad\qquad\qquad , i = 1, 2, ..., n \end{aligned} \tag{1}$$

Where $\mathbf{w}$ is the weight vector for the hyperplane, $\mathbf{x}$ is the vector of observations, y_i is the class label and $y_i \in \{+1, -1\}$, b is the bias parameter, $\Phi(\mathbf{x})$ is the map function that maps the input vector $\mathbf{x}$ to the feature space, ξ_i is a classification error for sample i, and C is the parameter that makes a balance between the classification error and maximum margin.

With the help of Lagrangian multipliers, the primal optimization problem (1) can be reformulated into another optimization problem called dual optimization problem. Lagrangian multipliers relax the constraints of the primal optimization problem and reformulates the problem into a new quadratic optimization problem with simpler constraints as follows:

$$\begin{aligned} min \quad & D(\alpha) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \mathbf{1}^T\boldsymbol{\alpha} \\ s.t. \quad & \sum_{i=1}^{n} y_i\alpha_i = 0 \qquad , i = 1, 2, ..., n \\ & 0 \leq \alpha_i \leq C \quad , i = 1, 2, ..., n \end{aligned} \tag{2}$$

Where $\boldsymbol{\alpha}$ is Lagrangian multipliers and $\alpha_i \in \boldsymbol{\alpha}$, $\mathbf{1}^T$ is a vector of ones and Q is a matrix of size $n \times n$ where $Q_{ij} = y_i y_j \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$. Equation (2) is known as dual equation. We reformulate Q_{ij} as $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, where $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$ is called a kernel function. The advantage of using kernel function is that we can compute Q with out knowing the map function $\Phi(.)$ explicitly and instead we can choose an appropriate kernel function to calculate Q. Four well-known kernel functions are as follows:

- Linear kernel that is an inner product or dot product of input vector and is used for linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$,
- Gaussian kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$,
- Laplacian kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma|\mathbf{x}_i - \mathbf{x}_j|)$,
- Polynomial kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + const)^d$, where $const = 0$ for homogeneous polynomial kernels and $const = 1$ for inhomogeneous kernels.

Different mathematical solvers are utilized to solve the primal, the dual, or the primal-dual optimization problem. Interior Point Method (IPM) starts from an initial point located in the interior feasible region and moves towards the optimal point(s) in an iterative manner. One of the advantages of IPM is its high degree of inherent parallelism compared to other solvers. IPM can use approximation methods such as Incomplete Cholesky Factorization (ICF) to approximate the original matrix $Q_{n\times n}$ to a smaller matrix as $H_{n\times p}$, where $p \ll n$. This approach can improve the time efficiency of the computations. In this paper, we have chosen the PSVM software that solves the SVM problem by utilizing IPM solver.

A. Parallel Support Vector Machines

We have conducted our experiments using the Parallelizing Support Vector Machines (PSVM) software that is originally written by Chang et al. [20]. PSVM uses ICF to reduce the problem size and IPM to solve the primal-dual optimization problem (1) and (2). ICF approximates the original $n \times n$ linear system Q to the smaller $n \times p$ linear system H, i.e., $Q \approx H^T H$ where n is the number of samples or instances, p is the reduced column size and $p \ll n$. The parallel ICF (PICF) is computed by a row-based round-robin algorithm and distributed evenly to the machines. The primal-dual problem is then solved by a parallel implementation of IPM and makes use of PICF. All the parallelization is done by utilizing the MPI library standard. Chang et al. claim that based on their empirical results when the reduced column size p is chosen as $\sqrt{n}$, then the error ϵ is negligible, where $trace(Q - H^T H) < \epsilon$. On the one hand they show the class-prediction accuracy obtained with some different values of p smaller than $\sqrt{n}$ in table 1 in their paper [20], albeit on the other hand they do not discuss further the impact of varying the p value for different kernel functions in a SVM problem. Therefore we further study the impact of different value of p on the class-prediction accuracy.

III. Related Works

SVM problems has got considerable attention in the last decade and the optimization problem derived from SVM problems are well studied. Challenges in SVM problems can be mentioned as the need for a large amount of memory for training samples [8][19] and the intense training time [1] when the problem size is large. This gives a motivation to use optimization methods along with HPC tools and parallel programming to involve more computational power to spilt the original problem into smaller sub-problems in order to

fit into memory [19]. Decomposition methods [12][6][16] are one of the highly invested methods in SVM. In the decomposition methods only a subset of variables is updated [16]. Based on this idea software as LIBSVM [4], SVM^{light} [10] have been implemented. Although software as LIBSVM and SVM^{light} using decomposition methods are efficient to solve SVM problems however when the problem size is large their performance is degraded since their computations are done in serial manner.

IV. Technologies for High Performance Computing

In this section, we briefly describe the technologies needed for the study of the PSVM communication and the improvement of the code.

A. Single Instruction Multiple Data

Single Instruction Multiple Data (SIMD) is an architecture of parallel machines that use multiple processing elements to operate a single instruction on multiple data elements simultaneously. Today most compilers can automatically optimize simple code structures using vectors to take benefit from SIMD instructions. A common mistake is to make data dependent whereby the compiler can't optimize it [14]. In such cases the code needs to be restructured to make use of SIMD. The gain from SIMD is dependent on CPU architecture where the old SSE instructions gives a two fold improvement while the newest AVX with FMA can give up to 8 times improvement.

V. Preparation for Experiments

Before we describe the conducted experiments, we do a study of the SVM algorithm using the PSVM software and point out areas of interest that we have our main focus on. These parts are chosen based on their computational time relative to the total calculation time. The areas we focus on can benefit from HPC improvements which affect the total calculation time. Minor HPC improvements on heavy computational parts can have a large effect on the total computation time when the size of the problem is very large, and if the calculation time is not the issue of interest, we can improve accuracy for the same calculation time.

A. Used Factorizations

CF function calculates the Cholesky factorization of the original matrix A, i.e., CF calculates a lower triangular matrix G such that $A \approx GG^T$. CF solves a linear system by forward and backward substitutions. ICF approximates the original $n \times n$ matrix Q to a smaller $n \times p$ matrix H, i.e., $(Q - H^T H) \leq \epsilon$, where $p \ll n$ and ϵ is the error.

B. Algorithm

The solving process regarding (1) is done in two steps, first create,

$$Q \approx H^T H + \epsilon \tag{3}$$

then solve by IPM which is similar to solve by Newton steps [18]. The detailed information about Newton method used in IPM is mentioned by Boyd [3] and Mehrotra [15].

$$\triangle\boldsymbol{\lambda} = -\boldsymbol{\lambda} + \mathbf{vec}(\frac{1}{t(C-\alpha_i)}) + diag(\frac{\lambda_i}{(C-\alpha_i)})\triangle\mathbf{x} \tag{4}$$

$$\triangle\boldsymbol{\xi} = -\boldsymbol{\xi} + \mathbf{vec}(\frac{1}{t\alpha_i}) - diag(\frac{\xi_i}{\alpha_i})\triangle\mathbf{x} \tag{5}$$

$$\triangle\nu = \frac{\mathbf{y}^T\boldsymbol{\Sigma}^{-1}\mathbf{z} + \mathbf{y}^T\boldsymbol{\alpha}}{\mathbf{y}^T\boldsymbol{\Sigma}^{-1}\mathbf{y}} \tag{6}$$

$$D = \mathbf{diag}(\frac{\xi_i}{\alpha_i} + \frac{\lambda_i}{C-\alpha_i}) \tag{7}$$

$$\triangle\mathbf{x} = \boldsymbol{\Sigma}^{-1}(\mathbf{z} - \mathbf{y}\triangle\nu) \tag{8}$$

Minimize $P(\mathbf{w}, b, \boldsymbol{\xi})$ and $D(\boldsymbol{\alpha})$ along $\triangle\boldsymbol{\xi}$ and $\triangle\boldsymbol{\alpha}$ respectively (9)

$$\boldsymbol{\Sigma} = \mathbf{Q} + \mathbf{diag}(\frac{\xi_i}{\alpha_i} + \frac{\lambda_i}{C-\alpha_i}) \tag{10}$$

$$\mathbf{z} = -\mathbf{Q}\boldsymbol{\alpha} + \mathbf{1}_n - \nu\mathbf{y} + \frac{1}{t}\mathbf{vec}(\frac{1}{\alpha_i} - \frac{1}{C-\alpha_i}) \tag{11}$$

To compute $\Sigma^{-1}z$ the Sherman-Morrison Woodbury formula is used:

$$\begin{aligned}\Sigma^{-1}z &= (D+Q)^{-1}z \approx (D + HH^T)^{-1}z \\ &= D^{-1}z - D^{-1}H(I + H^T D^{-1}H)^{-1}H^T D^{-1}z \\ &= D^{-1}z - D^{-1}H(GG^T)^{-1}H^T D^{-1}z\end{aligned} \tag{12}$$

The equation above containing Σ^{-1} is the most interesting parts of the algorithm with respect to amount of computations and therefore it is divided up into sub steps as following,

$$E = I + H^T DH \tag{13}$$

$$GG^T = E \tag{14}$$

C. Complexity

By going through the SVM algorithm using the PSVM software, we calculate the complexity of both the computation and the communication. We denote the amount of rows on each CPU by η where η is calculated by the amount of training samples divided by the amount of cores, i.e., $\eta = \frac{n}{k}$. Notice that all equations are solved once per iteration except equation (3) which is only solved once. Equation (3) needs $O(p^3 + \eta p^2)$ computations and $O(log(k)(p+f))$ communication where f is the number of features, p is the amount of columns on each CPU, η is the amount of rows on each CPU, and k is the amount of cores. Equation (4), (5) and (7) are just vector operations and therefore has a complexity of $O(\eta)$ computations. Equation (6) is solved by using the result from (13) and (14). Since the system is solved by backward and forward substitution with GG^T for both Σ^{-1}z and Σ^{-1}y therefore we get a complexity of $O(p\eta + p^2)$ computation and $O(log(k)p)$ communication. In a similar manner we

calculate the corresponding complexities for equation (8), i.e., $O(p\eta + p^2)$ for computation and $O(log(k)p)$ for communication. The line search (9) has a complexity of $O(p^2)$ computations. The equation (13) is a matrix multiplication and therefore has the complexity $O(\eta p^2)$ for computations and $O(log(k)p^2)$ for communication. The last equation (14) which is the CF is done serially on the master and has complexity $O(p^3)$.

Among the above mentioned equations, i.e., equations (3) to (14), we focus mainly on equations (13) and (14), since they are the most computationally expensive functions relative to the total computation time and thus they are more interesting for further study and investigation for potential improvements.

VI. Experimental Design

Our goal is to explore, and study the behaviour of the SVM algorithm regarding high performance computing point of view. In this respect, we choose an exploratory approach to discover new insights regarding SVM algorithm that can affect the class-prediction accuracy and the training time using PSVM. PSVM is implemented by Chang et al.[20]. We improve the PSVM code regarding structure, memory allocation, de-allocation and parallelism point of view as mentioned in section IV-A. We conducted experiments 1) to study the impact of changing hyperparameter C and γ on total training time by considering target class-prediction accuracy, 2) to study the impact of changing the number of columns p on the training time and the class-prediction accuracy using Gaussian and Laplacian kernels and to study the trade-off between the class-prediction accuracy and the time efficiency by changing p settings, 3) to evaluate the existence of a threshold for the appropriate number of computational nodes in order to get the advantage of parallelism as much as possible.

We measure the class-prediction accuracy of the models based on the number of correct predictions among all the correct and incorrect predictions.

$$Accuracy = \frac{T_- + T_+}{T_- + T_+ + F_- + F_+}$$

Where T_+, T_-, F_+ and F_- are true positive, true negative, false positive and false negative respectively.

The reduced number of columns in ICF is denoted by $p = n^r$ where n is the number of samples and r is the reducing ratio between 0 and 1. For all the experiments unless stated otherwise, we use the improved PSVM and the publically available *cod-rna, covertype, webspam* and *url* datasets provided by UCI data repository for machine learning [11] and by Fan et al. [5].

A. Experiment 1: Sensitivity of PSVM Regarding C and γ

One of the challenges of SVM problems using Gaussian and Laplacian kernels is to choose appropriate values for hyperarameter C and γ [8][7], since the class-prediction accuracy and time efficiency [13] are influenced by these parameters. Intuitively, γ means how far the influence of a single training sample is. A large value of γ shows the low influence of a single training sample while a low value of γ shows the high influence. The γ parameter has an inverse relationship with the radius of influence of support vectors [2]. The hyperparameter C makes a balance between the misclassification and the maximum margin. A low value of C makes the decision surface smooth while a large value of C gives freedom to the model to choose more support vectors among samples that results in more precise and accurate classification of the training samples [2]. One of the common way to find a suitable hyperparameter C and γ is cross-validation [9], however finding the best value of these parameters are still unclear.

In the first experiment, we study the impact of C and γ parameters on the sensitivity of training time meaning how much the training time varies by changing the C and γ parameters. In this experiment, we chose C between 0.1, 1, ... , 100000 and chose γ between 0.1, 1, ... , 1000. We conduct this experiment on two datasets, *cod-rna* and *covertype* and we study the total calculation time for training the samples.

B. Experiment 2: Reduced Column p

The reduced number of columns p in ICF has impact on the class-prediction accuracy and the training time and finding an appropriate value for p in ICF is non-trivial and controversial. Larger value of p results in higher class-prediction accuracy but slower total training time while smaller value of p results in fast training of samples but poor class-prediction accuracy. Although Chang et al. [20] suggest $p = \sqrt{n}$ based on their experimental results, however the trade-off between the class prediction accuracy and the time efficiency by changing p has been unclear. It is also unclear how the value of p influences different kernels.

In experiment 2, we study the impact of different p settings on the class-prediction accuracy and the training time. To study the performance of PSVM for different p settings, we divide the second experiment into sub-experiments. In the first sub-experiment, we have chosen a range of different p from $n^{0.3}$ to $n^{0.6}$ for the fixed values of C and γ and we measure the class-prediction accuracy. In the second sub-experiment, we have improved the first sub-experiment by choosing the best C and γ parameters for each p settings and we measure the class-prediction accuracy. In the third sub-experiment, we study the impact of p settings on total training time and training time on heavy computational parts of SVM algorithm as calculation of E, CF, ICF, Updating variables and other parts. In addition, we compare the training time regarding the original PSVM with the improved PSVM and do a short study of how the changes that we did affect the proportions. We conduct experiment 2 for webspam dataset with 300000 samples and 254 features and covertype datasets with 500000 samples and 54 features. We use both Gaussian and Laplacian kernel functions in this experiment.

C. Experiment 3

Although using HPC tools is promising for higher performance, however due to communication overheads choosing

appropriate number of computational powers such as number of computational nodes are still non-trivial. Based on the Amdahl's law , the maximum speed up of a program using parallel computing with multiple processor is limited regardless of the number of processors.

In experiment 3, we study the relation between the complexities we found in earlier chapter and the actual values when running the datasets. We run the experiment on improved PSVM with 8, 16, 32, 64, 128, and 256 computational nodes and we measure the total training time and the training time regarding E, CF, updating variable. For clarity, we measure the proportional training time on heavy computational parts of SVM algorithm as calculation of E, CF, ICF, Updating variables and other parts and we also measure the time for communication as the communication in E and the communication for Updating variables.

VII. Results

In this section we present the results of all three experiments and the corresponding sub-experiments.

A. Experiment 1

The results of first experiment are presented in Tables I and II. Table I represents the total time for training 59535 samples with 8 features for cod-rna dataset. In the first experiment, we choose a target class-prediction accuracy inside 10 percentage point of the best accuracy obtained. The red cells in table I shows that for the given C and γ the target accuracy is not achieved. Table I shows no trends between different settings of C and γ and the total training time. As the table shows the lowest total training time is 8 times slower than the highest total training time. The results of the first experiment on covertype dataset is shown in table II and it represents the total training time for 500000 samples with 54 features. In table II the target accuracy for $\gamma = 0.1$, $\gamma = 1$ and $\gamma = 10$ is not achieved. As the table shows the lowest total training time is 5.5 times slower than the highest total training time and same as table I, we do not detect any trends between C and γ and total training time. The kernel function that is used during this experiment is Gaussian kernel.

TABLE I
NUMERICAL RESULTS OF TOTAL TRAINING TIME WITH RESPECT TO DIFFERENT C AND γ SETTINGS FOR COD-RNA DATASET WITH 59535 SAMPLES AND 8 FEATURES USING GAUSSIAN KERNEL

Parameter	γ=0.1	γ=1	γ=10	γ=100	γ=1000
C=0.1	3.33	1.83	1.28	1.25	0.73
C=1	1.87	1.23	1.06	0.79	0.68
C=10	9.2	1.09	1.07	0.77	0.7
C=100	9.24	1.35	1.59	1.13	1.04
C=1000	8.91	9.11	8.69	0.99	9.02
C=10000	8.99	8.95	9.18	1.27	8.98
C=100000	9.17	8.96	9.24	1.96	10.41

B. Experiment 2

Figure 1 is related to the first sub-experiment in experiment 2 and it shows the class-prediction accuracy with respect to different column sizes. We have selected C and γ parameters as $C = 64$ and $\gamma = 2$ mentioned by Hsieh, Si and Dhillon [8] for webspam dataset. As figure 1 shows by increasing the number of columns, p from $n^{0.3}$ to $n^{0.5}$, the class-prediction accuracy degrades drastically, where for $p = \sqrt{n}$ the accuracy reaches it's lowest value, by increasing the value of p more than $n^{0.5}$, the class-prediction accuracy increases. We observe that the class-prediction accuracy is unstable by increasing the number of columns and for the fixed values of C and γ for all p columns.

TABLE II
NUMERICAL RESULTS OF TOTAL TRAINING TIME WITH RESPECT TO DIFFERENT C AND γ SETTINGS FOR COVERTYPE SCALED DATASET WITH 500000 SAMPLES AND 54 FEATURES USING GAUSSIAN KERNEL

Parameter	γ=0.1	γ=1	γ=10	γ=100	γ=1000
C=0.1	332.45	300.26	225.48	241.24	133.39
C=1	85.25	96.55	90.96	137.25	95.99
C=10	42.81	55.24	77.88	104.35	106.94
C=100	52.27	63.12	77.5	71.31	135.23
C=1000	68.58	80.54	92.37	42.99	178.69
C=10000	88.54	137.79	171.65	141.97	415.62
C=100000	284.75	178.96	367.56	417.05	414.29

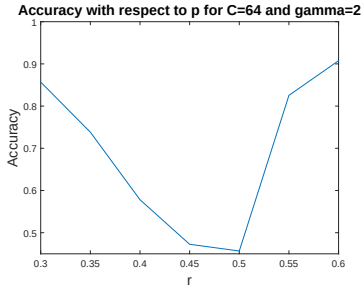


Fig. 1. The class-prediction accuracy with respect to different column numbers (p) for webspam dataset with 300000 samples and 254 features using C = 64, $\gamma = 2$ and Gaussian kernel function.

Figure 2 shows the second sub-experiment results for both Gaussian and Laplacian kernel functions. This figure gives a better insight that we can observe that replacing C and γ parameters with the best C and γ for each r results in stable class-prediction accuracy where $r = log(p)/log(n)$.

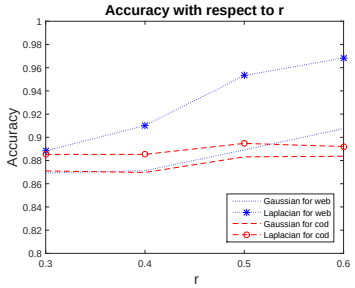


Fig. 2. The class-prediction accuracy with respect to different r ($r = log(p)/log(n)$) for webspam and cod-rna datasets using best C and γ for each r.

Figure 3 shows that the best C and γ stays close when p is changed. Figure 4 is related to the third sub-experiment of experiment 2 and shows the elapsed training time for E, CF, updating variables and other part of SVM algorithm regarding different p settings in webspam dataset. The upper sub-plot shows the elapsed time in seconds and the lower sub-plot shows the proportion time of each parts. The increased proportion of CF and E follows the results of the complexity analysis earlier. The difference between original and the improved PSVM can be seen in figure 5. As figure 5 shows the elapsed time regarding parts E and CF decreases in improved PSVM compared to the original PSVM when r increases.

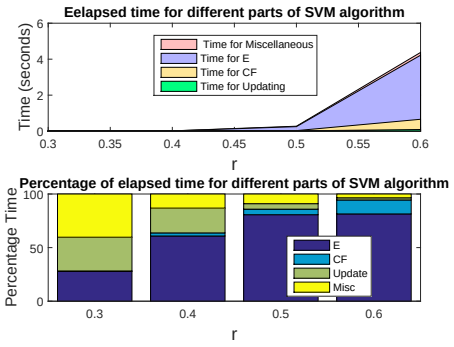


Fig. 4. The elapsed time for different parts of SVM algorithm with respect to different column numbers (p) for webspam dataset with 300000 samples and 254 features using Laplacian kernel function and best C and γ for each p.

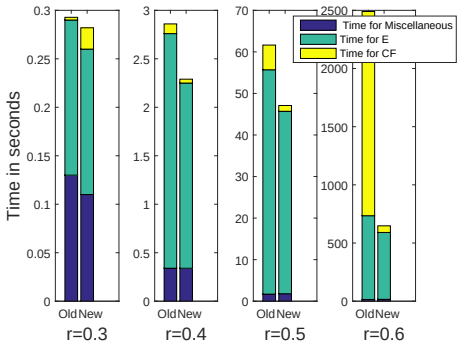


Fig. 5. The comparison between the original PSVM (Old) and the improved PSVM (New) software with respect to r.

C. Experiment 3

Figure 6 shows the training time and the time for calculating E, CF, updating variables of the SVM algorithm with respect to the number of computational nodes for covertype dataset. The upper sub-plot shows how the corresponding training time decreases by increasing the number of nodes from 8 to 64 while increasing the number of nodes more than 64 nodes does not show further improvement in training time. The lower sub-plot gives a better insight about the proportion of training time in E, CF, updating variables, ICF, and other parts of PSVM algorithm for covertype dataset along with communication time for E and communication time for updating variables. With this dataset, 128 nodes were enough to reach the threshold predicted in the complexity analysis.

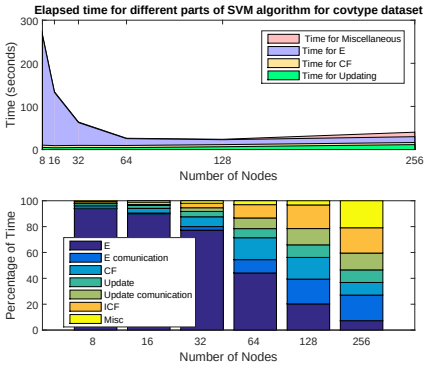


Fig. 6. The elapsed time for different parts of SVM algorithm with respect to the number of nodes, N for covtype dataset with 500000 samples and 54 features. Note the large proportion of communication at 128 nodes.

Figure 7 shows the same experiment as above for URL dataset. The upper sub-plot in figure 7 shows how the corresponding training time decreases by increasing the number of nodes from 8 to 128 while increasing the number of nodes more than 128 shows not remarkable improvement in the training time. The lower sub-plot gives a better insight about the proportion of training time in E, communication in E, CF, update variables, communications for updating variables, ICF, and other parts of the PSVM algorithm for URL dataset.

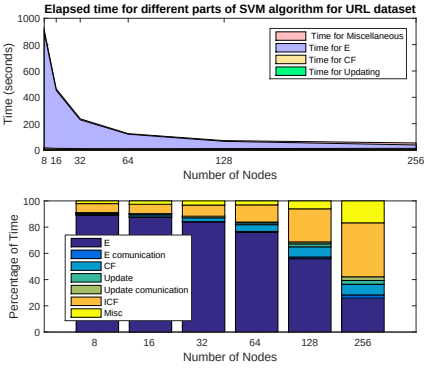


Fig. 7. The elapsed time for different parts of SVM algorithm with respect to the number of nodes N for URL dataset with 2150000 samples and 3231961 features.

VIII. Conclusion

In experiment 1, we study the impact of C and γ parameters on the training time considering the target accuracy. We did not find any interesting trend in the training time by changing these parameters. However this experiment helped us to get better insight about experiment 2 where we observed an improvement in the class-prediction accuracy while

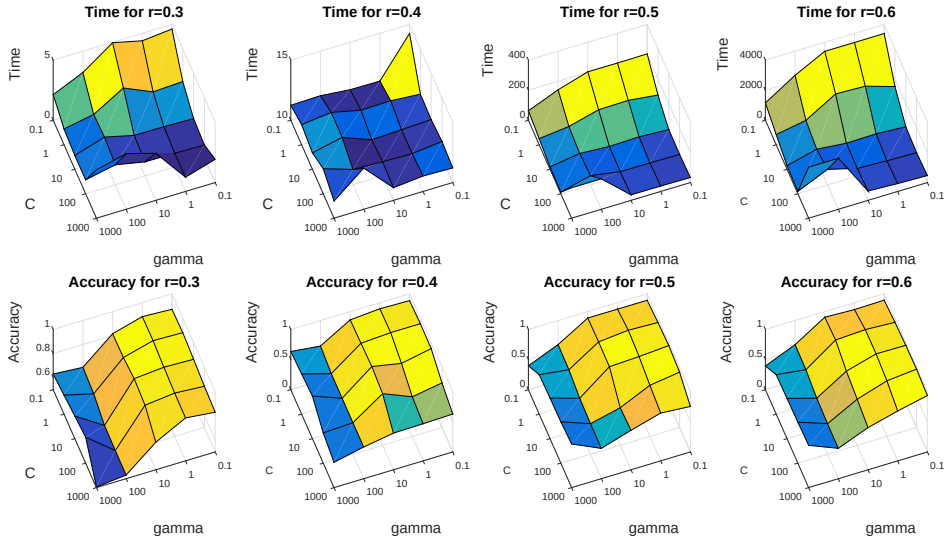


Fig. 3. The training time and accuracy with respect to different column numbers (p) for cod-rna considering different C and γ settings for Laplacian kernel.

increasing the number of columns. The result of experiment 2 showed that choosing appropriate value of p affects the choice of C and γ, this is clearly shown by figure 1. The common way to choose C and γ is done by cross-validation. The result of experiment 2 showed that choosing the best values for C and γ for special column size p is not necessarily the best value for another p. The complexity analysis for CF did predict a fast growth of calculation time for CF when p is increased which could clearly be seen by experiment 2. In figure 5, the original software was tested against the improved software for different p and showed 4 times improvement of performance by our modification on PSVM at large p because of the CF calculation. Already at smaller p we got an 20% improvement on the calculation of E. In experiment 3, we showed the existence of a threshold between the training time and the number of cores as predicted in a complexity analysis.

IX. Acknowledgements

Our experiments were using the Triolith system from National Supercomputer Center (NSC) at Linkoping University.

References

[1] A fast parallel optimization for training support vector machine. In Petra Perner and Azriel Rosenfeld, editors, *Machine Learning and Data Mining in Pattern Recognition*, volume 2734 of *Lecture Notes in Computer Science*. 2003.

[2] scikit-learn developers (BSD License) 2010 2014. RBF SVM parameters. http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html, 2015.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[5] Chih-Chung Chang, Chih-Jen Lin, and Rong-En Fan. LIBSVM data: Classification, regression, and multi-label. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/, 2015.

[6] Fu Chang, Chien-Yang Guo, Xiao-Rong Lin, and Chi-Jen Lu. Tree decomposition for large-scale SVM problems. *The Journal of Machine Learning Research*, 11:2935–2972, 2010.

[7] Asdrúbal López Chau, Xiaoou Li, and Wen Yu. Support vector machine classification for large datasets using decision tree and fisher linear discriminant. *Future Generation Computer Systems*, 36:57–65, 2014.

[8] Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. A divide-and-conquer solver for kernel support vector machines. *arXiv preprint arXiv:1311.0914*, 2013.

[9] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification, 2003.

[10] T. Joachims. Making large-scale SVM learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

[11] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[12] Chih-Jen Lin. On the convergence of the decomposition method for support vector machines. *Neural Networks, IEEE Transactions on*, 12(6):1288–1298, 2001.

[13] Gaëlle Loosli and Stéphane Canu. Comments on the "core vector machines: Fast SVM training on very large data sets". *J. Mach. Learn. Res.*, 8:291–301, May 2007.

[14] S. Maleki, Yaoqing Gao, M.J. Garzaran, T. Wong, and D.A. Padua. An evaluation of vectorizing compilers. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 372–382, Oct 2011.

[15] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[16] John Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methodssupport vector learning*, 3, 1999.

[17] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson International Edition. Pearson Addison Wesley, 2006.

[18] Tamás Terlaky. *Interior point methods of mathematical programming*, volume 5. Springer Science & Business Media, 1996.

[19] Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *The Journal of Machine Learning Research*, 7:1467–1492, 2006.

[20] Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, Hang Cui, and Edward Y. Chang. Parallelizing support vector machines on distributed computers. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 257–264. Curran Associates, Inc., 2008.

The Support of an Experimental OpenCL Compiler on HSA Environments

Chun-Chieh Yang, Shao-Chung Wang, Chou-Chuan Chen and Jenq-Kuen Lee
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
{jet, scwang, ccchen}@pllab.cs.nthu.edu.tw
jklee@cs.nthu.edu.tw

Abstract— *In recent years, with the increasing computing power and programmability on GPU, GPU has become an important role on hardware accelerator. Heterogeneous System Architecture (HSA) announced by HSA Foundation is an approach to benefit both CPUs and GPUs advantages. Open Computing Language (OpenCL) is one of the well-known programming frameworks for parallel computing on heterogeneous architecture. In this paper, an OpenCL framework is designed and implemented on HSA platform. An OpenCL compiler for HSA uses low-level virtual machine (LLVM) and an OpenCL runtime extends Portable Computing Language (PoCL) framework are built. PoCL is a portable OpenCL implementation for different parallel hardwares. Furthermore, HSA-related tools released by HSA Foundation are also integrated in our framework. Experimental results indicate that our framework provides enough features to support for advanced research.*

Keywords: HSA, OpenCL, Compiler, Runtime, LLVM

1. Introduction

To satisfy the performance and energy consumption constraints, heterogeneous multi-core architecture has become the popular design on many devices. Especially, GPUs has become a popular hardware accelerator with increasing programmability and computing power. Many applications are benefited from parallel execution, such as medical image [1], bioinformatics [2], and fluid dynamics [3].

Recently, Heterogeneous System Architecture (HSA) has been widely discussed [4], [5]. HSA is announced by HSA Foundation which is an approach to benefit advantages of both CPUs and GPUs [6]. HSA can integrate properties of CPUs, GPUs, and other various processing units. In HSA, CPUs and GPUs share the same Shared Virtual Memory (SVM). They could directly access data on other devices by using pointers and user applications could dispatch the kernels to GPU without data movement overhead. HSA utilizes the computing ability of GPUs to dramatically increase performance. Furthermore, it also provides low level runtime API [7] to manage tasks, resources, and Heterogeneous System Architecture Intermediary Language (HSAIL) that is a LLVM intermediary representation (IR) to represent parallel kernel from high level language like Open Computing Language (OpenCL), Open Multi-Processing (OpenMP) and even Java.

However, it is not easy to program on heterogeneous architecture. Many programming frameworks are proposed to solve this issue such as Halide [8], C++ AMP [9], OpenACC [10], and APARAPI [11]. OpenCL is one of the famous framework for heterogeneous and parallel computing [12], [13], [14]. It is an open standard maintained by Khronos Group [15], and designed to run on different kinds of parallel hardware resources like CPU, GPU, DSP, and FPGA. OpenCL involves host program and kernel code. Programmers write the host program by using some runtime APIs to get the information of computing devices, allocate memory buffer, create parallel kernel, and dispatch tasks. Kernel code executed in parallel is written by OpenCL C Language with some C99 extension such as memory qualifier and vector type.

In this paper, an OpenCL framework is designed and implemented on HSA platform. OpenCL C compiler and runtime that support OpenCL 1.2 on a HSA platform are built. Our compiler supports some OpenCL feature extensions, such as memory qualifiers and vector operations. The compiler translates the OpenCL kernel into HSAIL and the HSAIL is encoded into BRIG binary code format by using the tool released from HSA Foundation [16]. Furthermore, Portable Computing Language (PoCL) is used to build our OpenCL runtime [17]. PoCL is an open source OpenCL framework that targets various parallel hardware resources. In our framework, a HSA GPU device is added in the PoCL device list, and a bridge is written between HSA runtime APIs and OpenCL Runtime APIs that handle some special HSA features for OpenCL running on HSA GPUs. For example, the runtime arranges kernel arguments to execute kernels and dispatches kernels by filling the packet of Architected Queuing Language (AQL). Furthermore, dynamic and local memory usages from the OpenCL program are handled to work on HSA platform

In our experiments, OpenCL programs from AMD SDK 2.8 benchmarks are executed on AMD HSA-enabled processor [18]. From experimental results, the execution time of our framework of the OpenCL program is better than that of HSA-HLC-Stable compiler released by HSA Foundation, but is worse than the OpenCL compiler. The execution time of

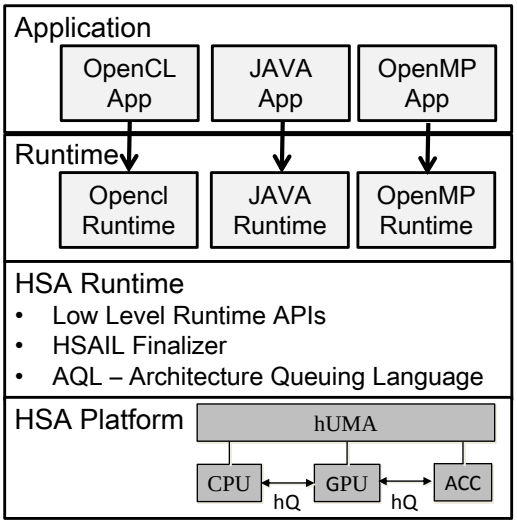


Fig. 1: HSA Software Architecture

```
__kernel void vec_add (
  __global float *a,
  __global float *b,
  __global float *c,
  const unsigned int n)
{
  int id = get_global_id(0);

  c[id] = a[id] + b[id];
}
```

Fig. 2: OpenCL Sample

our framework of the OpenCL kernel is better than the other two compilers in some applications, but is worse than in some other applications. Our framework successfully passes some benchmarks and is enough for advanced research such as vector operations and memory layout optimizations on OpenCL kernels and HSA task scheduling between CPUs and GPUs.

The remainder of this paper is organized as follows. Section 2 presents details on HSA and OpenCL. Section 3 describes the design and implementation details, and some experimental results are shown in section 4. Finally, conclusions and future work are described in section 5.

2. Overview of HSA and OpenCL

In this section, some introductions of HSA and OpenCL are introduced. Our compiler and OpenCL runtime is developed by using technologies of HSA and OpenCL. Some related properties of them will be introduced.

2.1 HSA Overview

The design goal of HSA can be shown in Fig. 1. The two hardware technologies Heterogeneous Uniform Memory Access (hUMA) and Heterogeneous Queuing (hQ) are introduced in HSA. hUMA can make both CPUs and GPUs share a single memory space that GPUs can directly access CPU memory addresses include reading and writing data and CPUs also read and write at the same time. Traditionally, users should view the CPUs and GPUs memory as completely separate memory even through CPUs and GPUs are integrated into the same chip. hQ lets CPUs place tasks of GPUs directly into GPU task queue without OS help and GPUs can also place its tasks into the GPU or CPU queue to be dispatched. hQ also uses Architecture Queuing Language (AQL) to define a standard Queuing Format. All HSA agents should be described as AQLs so that the agent can directly dispatch tasks into different HSA component hardware queues without software translation.

To obtain the benefit from the HSA features, HSA also provides a middle layer with low level runtime APIs and HSAIL to make high level heterogeneous language to lower into this layer. Therefore, high level heterogeneous language should have its compiler to translate the program into HSAIL and its runtime should be able to interact with HSA runtime. At execution time, the HSA runtime uses these APIs to dispatch tasks and manage resources. HSA runtime also invokes HSA Finalizer to translate the HSAIL into target machine code.

2.2 OpenCL Overview

OpenCL provides a programming framework to accelerate applications with tasks and data parallelism. OpenCL has C99-extended programming language, named as OpenCL C, for writing the kernel to run in parallel. OpenCL C involves some qualifier and vector operations to extend and built-in functions. Figure 2 is the simple OpenCL Kernel code. The *__kennel qualifier* identifies this kernel code that runs on OpenCL computing devices and is identified as an entry point. The memory qualifier *__global* means to place variables on global memory. The built-in function *get_global_id()* returns the thread id and the array uses this id as an index to access the data in the sample program.

In addition to the kernel code, programmers also must write the host program with OpenCL runtime APIs to control the task how to run on computing devices. For example, the kernel code is executed by many parallel threads on GPU or working-items in OpenCL specifications. Programmers must decide how these threads are divided into groups in host program. The threads in different groups can run independently and the threads in the same group can be synchronized during execution. Programmers also should decide the kernel code execution order, synchronization between kernels, memory allocations, and kernel code compilations. All of the above actions should be interacted with HSA

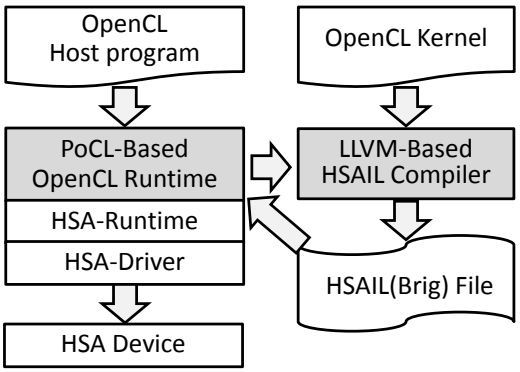


Fig. 3: OpenCL Flow on HSA

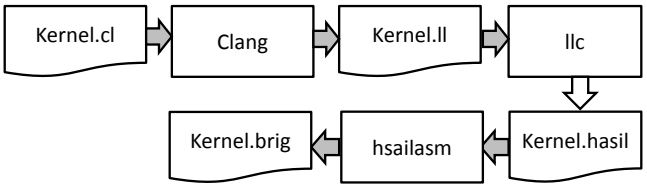


Fig. 4: OpenCL Compilation Flow

```
version 1:0:$full:$small;

kernel &__OpenCL_vec_add_kernel(
        kernarg_u32 %arg_val0,
        kernarg_u32 %arg_val1,
        kernarg_u32 %arg_val2)
{
@__OpenCL_vec_add_kernel_entry:
  workitemabsid_u32       $s0, 0;
  shl_s32         $s0, $s0, 2;
  ld_kernarg_u32  $s1, [%arg_val0];
  add_s32         $s1, $s1, $s0;
  ld_global_u32   $s1, [$s1];
  ld_kernarg_u32  $s2, [%arg_val1];
  add_s32         $s2, $s2, $s0;
  ld_global_u32   $s2, [$s2];
  add_s32         $s1, $s2, $s1;
  ld_kernarg_u32  $s2, [%arg_val2];
  add_s32         $s0, $s2, $s0;
  st_global_u32   $s1, [$s0];
  ret;
};
```

Fig. 5: A Sample Code of HSAIL

runtime and some information should be passed to HSA runtime for packeting in AQL format.

3. The Proposed Framework with Compiler and Runtime

In this section, the proposed framework with compiler and runtime will be explained. Figure 3 shows our OpenCL flow on HSA. First, OpenCL host program runs on our PoCL-based OpenCL runtime. Runtime will invoke our LLVM-Based HSAIL compiler to compile OpenCL kernel into HSAIL and also invoke HSAIL assembler to translate the HSAIL into BRIG binary file. Our OpenCL runtime is built on the HSA runtime and driver released by HSA Foundation. The detailed compiler and runtime flow is described in the following.

3.1 The Prototype Compiler for OpenCL to HSAIL

Currently, most of OpenCL compilers are implemented by LLVM[19]. Khoronos also define OpenCL portable IR, Standard Portable Intermediate Representation (SPIR), which is based on LLVM IR with some specific annotations for OpenCL C extension. Therefore, our compiler uses *Clang* as front end and LLVM *llc* as backend. Figure 4 is the proposed OpenCL to HSAIL compilation flow. The OpenCL kernel code is converted to SPIR by Clang and then uses LLVM llc to compile bitcode to HSAIL. Finally, the hsailasm tool which is released by HSA Foundation is utilized to convert HSAIL code into BRIG format.

Our compiler has already supported some OpenCL features, such as kernel/memory qualifiers and some vector operations. Figure 5 is the simple HSAIL code converted from OpenCL codes shown in Fig. 2 by using our compiler. The *__kernel qualifier* for function *vec_add* is annotated in LLVM IR which is used to annotate kernel for function *vec_add* in HSAIL and kernarg for *vec_add* parameters. For the same reason, *ld_global_xx* and *st_global_xx* instructions are generated because the loaded and stored data is annotation with *__global* memory qualifier in OpenCL kernel. The built-in functions in OpenCL are mostly supported by intrinsic functions.

For example, the built-in function *get_global_id(0)* in Fig. 2 is directly translated to *workitemabsid_u32* in Fig. 5. Vector accesses and some basic vector arithmetic instructions are also supported that makes the generated code more efficiently. Furthermore, if vector load instructions are supported, the vector data can be loaded directly instead of generating four scalar load instructions. As shown in Fig. 6, in the dark and light gray parts, four *ld_global_u32* instructions can be merged into one *ld_v4_global_u32*. In this situation, the hardware does not load 32-bit data four times but directly load 128-bit data once.

3.2 PoCL-Based OpenCL Runtime for HSA

PoCL is used to build our OpenCL runtime for HSA. PoCL is one of the open sources implemented for OpenCL standard that has already supported different hardware such as homogeneous muti-core X86/ARM and VLIW-style TTA processors. PoCL has already passed some OpenCL benchmarks and been designed easily to adapt to new targets and

```
ld_kernarg_u64  $d0, [%b];
ld_global_u32   $s0, [$d0];
ld_global_u32   $s1, [$d0+4];
ld_global_u32   $s2, [$d0+8];
ld_global_u32   $s3, [$d0+12];
ld_kernarg_u64  $d1, [%a];
ld_global_u32   $s4, [$d1];
ld_global_u32   $s5, [$d1+4];
ld_global_u32   $s6, [$d1+8];
ld_global_u32   $s7, [$d1+12];
```

```
ld_kernarg_u64  $d0, [%b];
ld_kernarg_u64  $d1, [%a];
ld_v4_global_u32 ($s0, $s1, $s2, $s3), [$d0];
ld_v4_global_u32 ($s4, $s5, $s6, $s7), [$d1];
```

Fig. 6: A Simple HSAIL of Vector Load

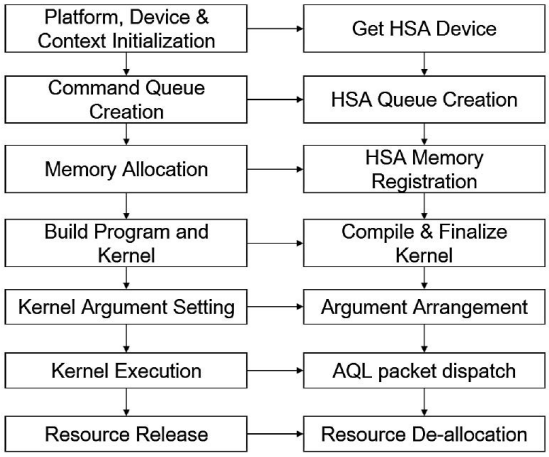


Fig. 7: The Runtime Flow between OpenCL and HSA

devices. PoCL also uses the LLVM as a kernel compiler. Therefore, it is a good choice for building our OpenCL runtime for HSA by using PoCL.

Runtime flows of both OpenCL and HSA are similar. They need to request equipment, set work queues, load in memory addresses, compile kernel codes, forward parameters, and execute kernel codes. After executing the kernel codes, host programs release resources used. Figure 7 shows the relationship of runtime behaviors between the OpenCL and HSA.

When running an OpenCL host program, it should initialize an environment. The environment includes a platform that provides devices, which are supplied by hardware vendors or some open source projects, for kernels and OpenCL contexts to run. In a HSA host program, users also need to initialize HSA runtime first and get a HSA device by using HSA agent iteration functions. According to the user assignment in the host program, runtime can determine which devices are used. In our implementation, if the device type is *CPU*, a PoCL runtime API initializes device pthread that is the original device of PoCL and the kernel will be translated to pthread model in the build steps. If the device type is *GPU* or *default*, a Pocl runtime API initializes the HSA device. If the device type is *ALL*, both GPU and CPU devices are initialized. Besides, HSA runtime APIs also are invoked to obtain the device feature and setup some OpenCL device information.

Table 1: AQL Packet Correspondence

AQL Packet Member	Source
header	HSA_PACKET_TYPE_DISPATCH
dimensions	*work_dim*
workgroup_size_x	*local_work_size[0]*
workgroup_size_y	*local_work_size[1]*
workgroup_size_z	*local_work_size[2]*
grid_size_x	*global_work_size[0]*
grid_size_y	*global_work_size[1]*
grid_size_z	*global_work_size[2]*
private_segment_size	HSA code descriptor
group_segment_size	HSA code descriptor + dynamic local size
kernel_object_address	HSA code descriptor
kernarg_address	HSA kernel argument allocated address
completion_signal	HSA signal creation API

Then, corresponding to OpenCL standards, users need to create a queue and program object for communication between hosts and devices. If the command queue is for HSA GPU, it also needs to invoke HSA runtime APIs to create HSA queue.

The OpenCL kernel code object is created by using opencl creating APIs, and different OpenCL APIs are used to store the OpenCL kernel source file or BRIG files. If the input file is OpenCL kernel code, kernel files are compiled to BRIG files as introduced in the compiler flow. After compiling, the BRIG file is loaded as a BRIG module and the module becomes the member of both HSA program object and OpenCL program object. Finally, the BRIG module is finalized at OpenCL kernel creating APIs. In this step, the kernel name can be obtained from the parameters with API and is used to find the correspondent symbol offset which must be filled in a finalization list. The finalizer finalizes the kernel object in the finalization list one by one.

At OpenCL kernel argument setting and kernel execution stage, the kernel argument for HSA should be initialized and the HSA agent packet should be filled in AQL prepared for dispatching kernel to execute. Table 1 illustrates the HSA dispatch packet members and the source. *Header* identifies the packet type. There are three types of the packet type, *DISPATCH*, *AGENT_DISPATCH*, and *BARRIER*. *DISPATCH* is used to dispatch the kernel from a host to a device,

and *AGENT_DISPATCH* is used to dispatch kernel from a device to a device. This feature can be used to support the OpenCL 2.0 device feature. *BARRIER* is used to delay packets for describing packet dependencies.

Dimensions, *grid_size* and *workgroup_size* correspond to *work_dim*, *global_work_size* and *local_work_size* which can be obtained from OpenCL APIs used by defining the number of thread creation. Private and Group segment size can be queried from HSA code descriptor acquired from finalizing the BRIG module. The group segment size only indicates static group size which is the total variable declaration size with *__local* qualifier in the OpenCL kernel code. The dynamic group size will be acquired from the OpenCL kernel argument setting API. *Kernel_object_address* is used to point to the address where the kernel function resides, and it can be queried from the HSA code descriptor. In the HSA runtime specification, a user should find a block of memory called as region that can be used in kernel arguments, and allocate the region as the place to store kernel arguments by using the HSA memory allocation API. The API would output the start address of the block of memory, and the address should be passed to *Kernelarg_address*. *Complete_signal* is used to identify whether the kernel dispatched finishes executing or not. A HSA signal object is created by using the HSA signal API and is passed to *complete_signal*.

In our kernel argument initialization mechanism, the argument buffer is allocated and initialized based on different types. The type of kernel arguments is accessed by using the kernel argument descriptor which is built when a user invokes the kernel creating API, and an argument structure array is generated that includes size and value obtained from the kernel argument setting API. Then, a buffer size is created which is queried from *kernelarg_segment_size*, and the required kernel argument data is copied to the buffer one by one. A pointer is used to point to the start of the current argument data that should be copied. If the type of an argument that gets from the kernel argument descriptor is a local type, the *group_segment_size* should be copied as a pointer to the buffer instead of a null pointer created by users.

Dynamic group size also is added into *group_segment_size*, because group memory is divided into two parts according to HSA runtime specifications, static group memory and dynamic group memory. Static group memory is allocated from address 0x0 to the size *group_segment_size*, and dynamic group memory starts from *group_segment_size*. The amount of group memory size is obtained when a local qualifier argument is accessed. In terms of structure types, OpenCL and HSA access structure arguments in different ways. In OpenCL, the structure argument is regarded as a block of continuous memory. However, in HSA, the structure argument is regarded as a pointer. Therefore, the pointer of structure argument is copied instead of the total structure argument such as scalar type. After finishing initializing the kernel argument, all of the buffer data is copied to the address *kernarg_address* in the AQL packet for dispatching.

4. Evaluation

In this section, some experimental results are performed to show the performance of our proposed compiler, HSAIL-HLC-Stable compiler, and OpenCL runtime on HSA platform.

4.1 Environment

AMD Kaveri A10-7850K APU was used as our experimental platform which includes one 4-core CPU and GPU and is the first HSA-enabled processor. The platform ran Ubuntu 14.04.2 LTS and AMD HSA runtime and driver 1.0 version released by HSA Foundation were installed. Our OpenCL to HSAIL compiler is built on LLVM 3.3 and OpenCL runtime is built on PoCL 0.8. Moreover, our benchmarks are selected from AMD APP SDK 2.8.

4.2 Experimental Results

In our experiments, three different OpenCL frameworks were compared. One is current AMD official OpenCL framework from AMD GPU driver Catalyst Omega 14.12. Another one uses our PoCL-based OpenCL runtime but uses HSAIL-HLC-Stable compiler released by HSA Foundation. The other framework uses our OpenCL runtime and compiler. Figure 8 illustrates the comparison of execution time between three frameworks. As shown in Fig. 8, the execution time of Pocl-based Runtime with the proposed compiler is faster than that of AMD official OpenCL framework in some benchmarks, such as BitonicSort and FloydWarshall.

However, most of the execution time of the proposed compiler is longer than AMD official OpenCL framework. It means that our framework still can be improved in the execution time. Although the execution time of our framework in most applications are more than OpenCL runtime, the execution time of our framework decreased more than 70% when comparing with HSAIL-HLC-Stable. This is because the compilation time of HSAIL-HLC-Stable is too long for performing compiler optimizations. Therefore, the execution time of the kernel compiled by HSAIL-HLC-Stable is faster in most of applications as shown in Fig. 9 and Table 2.

Figure 9 shows the comparison of kernel execution time between three frameworks and Table 2 lists the kernel execution time. The execution time of the kernel generated by our proposed compiler is better than AMD official OpenCL framework but worse than HSAIL-HLC-Stable. From the above two experiments, our proposed compiler has ability to do advanced research about HSA compiler and runtime optimizations.

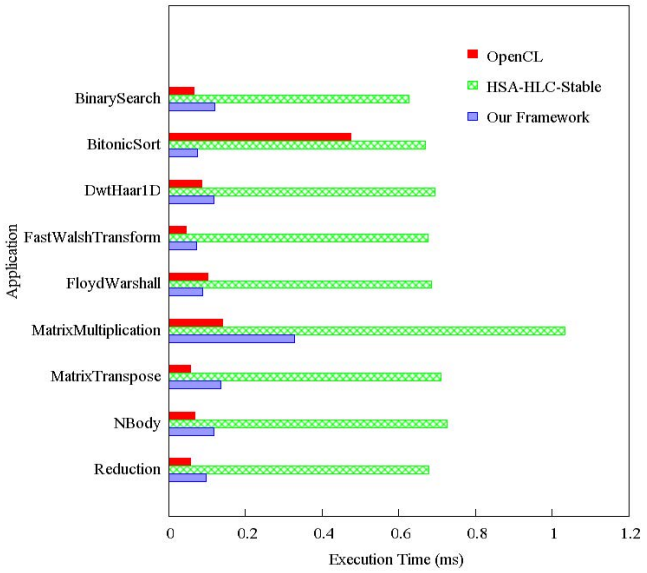


Fig. 8: Comparison of OpenCL Program Execution Time

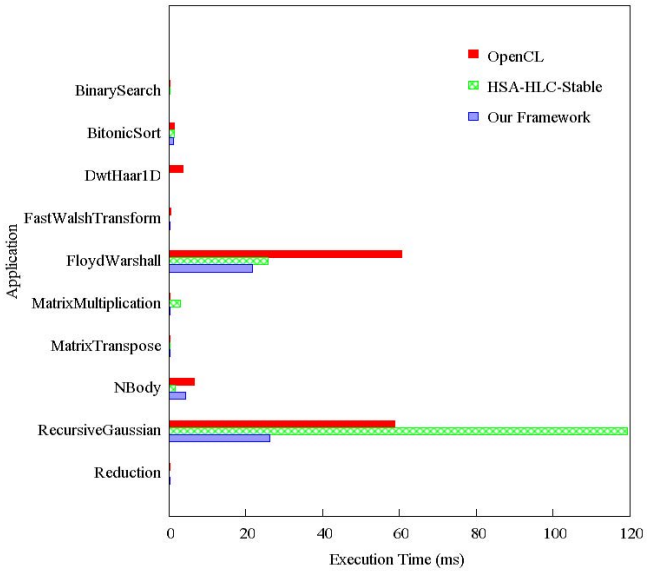


Fig. 9: Comparison of OpenCL Kernel Execution Time

Table 2: The List of each Kernel Execution Time(ms)

	OpenCL	HSA-HLC-Stable	Our Framework
BinarySearch	0.154	0.075	0.085
BitonicSort	1.272	1.128	1.117
DwtHaar1D	0.319	0.146	0.192
FastWalshTransform	0.42	0.294	0.275
FloydWarshall	54.105	23.049	19.388
MatrixMultiplication	0.185928	2.719	0.265
MatrixTranspose	0.283	0.214	0.255
NBody	5.868	1.509	3.872
RecursiveGaussian	52.321	106.35	23.298
Reduction	0.162	0.089	0.116

5. Conclusions and Future Work

In this paper, an OpenCL framework is built on the HSA Platform. Our OpenCL to HSAIL compiler is built based on LLVM and runtime is based on PoCL. Our proposed compiler supports some OpenCL feature extensions, such as memory qualifier and vector operation. Our proposed compiler translates the OpenCL kernel code into HSAIL and the HSAIL is encoded into BRIG binary format by using the tool released from HSA Foundation. Therefore, a HSA GPU device is added in the PoCL device list, and some special HSA features are handled for OpenCL running on HSA GPUs, such as filling agent packet with AQL for kernel dispatching.

From experimental results, our proposed framework can pass some benchmarks from AMD APP SDK. Our proposed framework supports enough OpenCL features for advanced research, and is planned to support for more OpenCL features for performing some optimizations on compiler and runtime, such as performing vector and memory layout compiler optimizations on OpenCL kernels, and HSA task scheduling between CPUs and GPUs.

Acknowledgment

This research is supported in part by the Ministry of Science and Technology of Taiwan, the Ministry of Economic Affairs of Taiwan, and MediaTek Inc.

References

[1] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, "A survey of medical image registration on multicore and the gpu," *Signal Processing Magazine, IEEE*, vol. 27, no. 2, pp. 50–60, 2010.

[2] J.-S. Varré, B. Schmidt, S. Janot, and M. Giraud, *Manycore high-performance computing in bioinformatics*. chapter, 2010, vol. 8.

[3] Y. Wang, A. Malkawi, Y. Yi, and T. C. Center, "Implementing cfd (computational fluid dynamics) in opencl for building simulation," *Proceedings of The 12th International Building Performance Simulation (Building Simulation 2011)*, 2011.

[4] "HSA Foundation," http://www.hsafoundation.com/.

[5] P. Rogers and A. FELLOW, "Heterogeneous system architecture overview," in *Hot Chips*, 2013.

[6] L. T. Su, "Architecting the future through heterogeneous computing," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*. IEEE, 2013, pp. 8–11.

[7] "HSA Runtime Specification 1.0," http://www.hsafoundation.com/standards/.

[8] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 519–530, 2013.

[9] "Shevlin Park: Implementing C++ AMP with Clang/LLVM and OpenCL," http://llvm.org/devmtg/2012-11/Sharlet-ShevlinPark.pdf.

[10] "OpenACC," http://www.openacc.org/.

[11] "aparapi," https://code.google.com/p/aparapi/wiki/UsersGuide.

[12] M. J. Dinneen, M. Khosravani, and A. Probert, "Using opencl for implementing simple parallel graph algorithms," in *Proceedings of the 17th annual conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'11), part of WORLDCOMP*, vol. 11, 2011, pp. 1–6.
[13] A. Munshi, "Opencl-beyond programmable shading course." SIGGRAPH, 2008.
[14] M. Xin, H. Li, and J. Lu, "A research of mapreduce with gpu acceleration," in *PDPTA*, vol. 12, 2012, pp. 625–631.
[15] "Khronos," https://www.khronos.org/.
[16] "HSA Programmer Reference Manual Specification 1.0," http://www.hsafoundation.com/standards/.
[17] P. Jääskeläinen, C. S. de La Lama, E. Schnetter, K. Raiskila, J. Takala, and H. Berg, "pocl: A performance-portable opencl implementation," *International Journal of Parallel Programming*, pp. 1–34, 2014.
[18] "Accelerated parallel processing (app) sdk. Advanced Micro Devices Inc. ," http://developer.amd.com/tools-and-sdk/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk.
[19] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*. IEEE, 2004, pp. 75–86.

Modeling Parallel Applications for Scalability Analysis: An approach to predict the communication pattern

Javier Panadero[1], Alvaro Wong[1], Dolores Rexachs[1] and Emilio Luque[1]
[1]Department of Computer Architecture and Operating System (CAOS),
University Autonoma of Barcelona, Spain
javier.panadero@caos.uab.es, alvaro.wong@caos.uab.es, dolores.rexachs@uab.es, emilio.luque@uab.es

Abstract— *The performance of message-passing applications varies depending on the parallel system, causing potential inefficiencies when its number of processes increases. By this reason, it is critical to predict the application behavior before executing it, in order to use the system efficiently. We propose a methodology that allows us to predict the application scalability behavior in a specific system, providing information to select the most appropriate resources to run the application. The methodology strives to use a bounded analysis time, and a reduced set of resources. This paper presents the general methodology, focusing on validating the step of the methodology concerning to the generation of scalability communication model. We can predict the evolution of the communication pattern using a reduced set of resources. Analyzing from 16 to 256 processes, we can predict the communication pattern for 4,096 processes.*

Keywords: Modeling MPI applications, Application Scalability, Communication pattern

1. Introduction

With the advent of multicore and the constant hardware evolution, High Performance Computing (HPC) clusters have increased the number of cores significantly [1]. The users of these systems want to get the maximum benefit from this large number of cores, scaling their applications.

To achieve an efficient use of these HPC systems using a large number of cores, a point to consider before executing an application is to know its performance in the system. It is known that using more resources does not always imply a higher performance. The lack of this information may produce an inefficient use, resulting in not achieving the expected speedup.

Parallel applications are composed of a set of phases, which are segments of code delimited by communications events, that are repeated throughout the application [2]. These phases were written in the application code using specific communicational and computational patterns, which follow behavior rules. When the application increases the number of processes, the number of phases remains constant, but its patterns change their behavior following their behavior rules, being functionally constant. To obtain these phases, we use the PAS2P tool [3], which identifies the application phases and allows us to create the application signature. As is shown in fig. 1, the signature only contains the relevant application phases and their repetition rates (weights). Therefore, it allows us to cover approximately 97% of the total application code, in about 1% of the application execution time.

We propose a methodology to analyze and predict the strong scalability [4] behavior for message-passing applications on a given system, by running a set of small-scale signatures. It strives to use a bounded analysis time, and a reduced set of resources. Moreover, the methodology could also be useful for scheduling and code optimization.

The methodology focuses on characterizing and analyzing the communication and computational patterns of each phase, in a transparent way (without analyzing and modifying the source code), from a set of executions for a small-scale signatures, in order to model general behavior rules, to build the Scalable Logical Trace (STL), which is machine independent, depending on the way in which the application was developed. The STL will be generated for a N number of processes, and it will be used in the future, to predict the communication and computational time, in order to obtain the application execution time.

We present an overview of the methodology, focusing on explain in detail the scalability communication model, to predict the evolution of the communication pattern (spatial and volume parameters) of each phase as the application scales. In a previous paper [5], the key ideas of the method were presented. In this study, the whole procedure and its algorithms are explained in detail. In order to validate the method, we executed from 16 to 256 processes and we predict the communication pattern (Spatial and volume parameters) for 4,096 processes, which is validated with the real communication pattern obtained with PAS2P.

This paper is organized as follows: Section II presents the related work, Section III presents the proposed methodology, Section IV presents the scalability communication model, Section V presents experimental validation and finally Section VI, the conclusions and future work.

2. Related Work

Similar works to our approach have been presented in the literature. Wu et al. [6] generate the application communi-

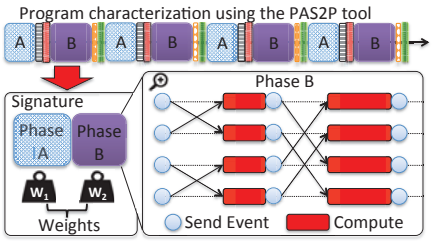


Fig. 1: Relevant phases that represent the signature.

cation trace for a large number of nodes, by extrapolating from a set of smaller traces. Their methodology is focused on SPMD (Single Program, Multiple Data) applications with stencil/mesh topology. Our proposal differs from this work, because it covers a wide range of MPI application, not only SMPD applications. Zhai et al [7] present the tool FACT, which collect MPI communication traces and extract application communication patterns through program slicing. This tool uses a set of code analysis techniques to generate a program slice that only contains the variables and code sections related to MPI events, and then executes the program slice to acquire communication traces. This tool allows to predict the communication pattern for a specific number of processes, our methodology differs because we execute a set of small-scale signatures to predict evolution of the communication pattern as the application scales.

There are other works based on analytical regression and machine learning methods, from executions for small-cores. Barnes et al [8] propose studying the scalability using regression models, isolating computation and communication to predict the application performance. Ipek et al [9] present a different approach based on multilayer neural networks. From a training set of the application executions, the application model is created automatically. This approach is interesting for its ease of use and its obliviousness to details of application internals. These works are based on the input parameter space to obtain the regression models and extrapolate its behavior. Our methodology focuses on obtaining the general behavior rules for each relevant application phase, to extrapolate its behavior to predict the application performance.

3. Proposed methodology

The main goal of the methodology is to analyze and predict the strong scalability behavior for parallel applications on a given system, using as input a limited set of small-scale signatures, as is shown in fig. 2.

The methodology is made up of three steps: Application characterization, Generation of the logical application model and Performance prediction.

As we mentioned before, the parallel applications are typically composed of patterns of computation and communication that are repeated throughout the application. These patterns are grouped in phases, which compose the application signature. The number of phases remains constant when the number of processes increases, but their patterns change their behavior following behavior rules. The objective of the characterization step is to obtain information about the communication and the computation patterns of each phase, to model the general behavior rules, which will be used to predict their behavior, as the number of the application processes increases. The methodology is restricted to applications where the communication pattern follows deterministic behavior rules.

To obtain the predicted application execution time, we carry out a set of signature executions for small-scale, which will be analyzed to obtain information from each phase. When the signature is executed in the system, it generates a trace file per process, which contains information of each phase. The trace provides information about the phase id, the type of MPI primitive, the source and destination of the communication, the communication volume in bytes, the computational time in nanoseconds and finally, the number of instructions for the computational time.

Once the phases have been characterized, their communication pattern, the computational pattern and the weight of each phase are analyzed and modeled to generate the general behavior rules, in order to construct the Scalable Logical Trace (STL) for a N number of processes. The input parameters of the general behavior rules will project the STL for a specific number of processes. The STL is composed of the intrinsic parameters for each phase needed to model the scalability of the parallel application, which are: the phase ID, communication pattern (spatial and volume parameters), number of instructions of the computational time and phase weight. The STL is generated per process instead of a global trace, with the objective being to model each process independently.

The STL has to be complemented with the computational time, in order to generate the physical trace, which is dependent of the machine, because contains the information

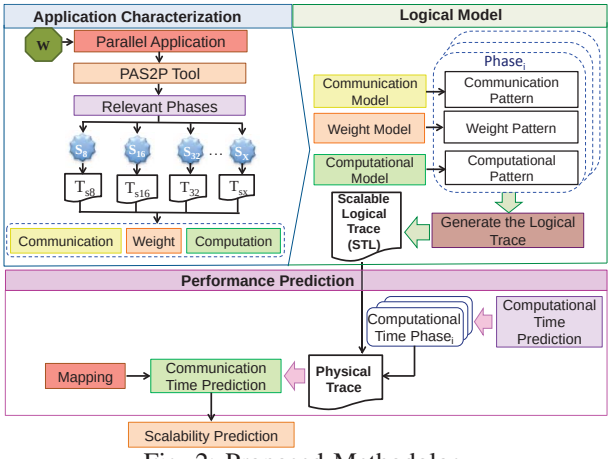


Fig. 2: Proposed Methodology

of the STL more the computational times for each phase for a N number of processes. To predict the computational time, we use a regression-based model by phase. The physical trace will be used to predict the communication time and obtaining the application performance.

To predict the communication time, the physical trace will be executed by segments of processes in a reduced number of resources, in an iterative way, until all the processes have been executed. Once the communication time has been predicted, the predicted execution time of each phase will be obtained. Then, we apply eq. 1 to obtain the application performance, where PET is the Predicted application Execution Time, m is the number of phases, *PhaseETi* is the Phase i Execution Time and *Wi* is the predicted weight of the phase i. As the objetive of this paper is to focus on explaining the computation method, we do not explain this step in detail.

$$PET = \sum_{i=1}^{m} (PhaseETi)(Wi) \quad (1)$$

In the next section, we explain in detail the scalability communication model, which predicts the evolution of the communication pattern for a large number of processes.

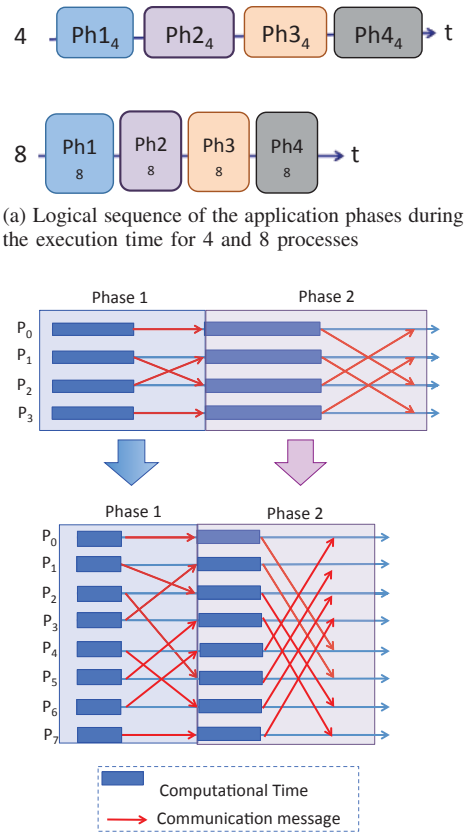
4. Scalability Communication Model

The scalability communication model comprises the general behavior equations and the data volume equations for each communication of each phase. The general behavior equations calculate the message destination from the source, while the data volume equations calculate the size of the message.

When we analyze the behavior of the phases, we know that as the application scales, the communications (number of messages and destinations), the communication volume, the computational time and the number of instructions of computation of a phase can change, but the work to be carried out will still be the same, distributed among more processes.

To model the communication behavior of each phase, it is necessary to recognize and relate the phases of the small-scale signatures. In order to relate the phases for a different number of processes, we use functional similarity. Two phases will have functional similarity, when the computation work to be carried out for both phases is the same, because is the same code segment, distributed between different number of processes, changing only the structure of the communication pattern.

To relate the phases, we use a method which is based on how the sequence of phases occurs, since it does not depend on the number of processes, only the way in which the application was developed. Fig 3 shown an example. As we can see in the fig. 3.a, the number of phases remains constant as the application increases from 4 processes to 8 processes. If we focus on the fig. 3.b, where the phases 1



(a) Logical sequence of the application phases during the execution time for 4 and 8 processes

(b) Behavior of the phases for 4 and 8 processes

Fig. 3: The functional similarity relates the phases for different number of processes

and 2 are showed in detail, we can see that the behavior is different from 4 processes to 8 processes, because the phases have different communication pattern and different computational time between them, but the work carried out by the phases is the same, distributed between different number of processes, because they are in the same logical position in the application.

Once the phases have been related, the predicted data volume of each communication will be obtained by mathematical regression models, while for obtaining the general communication rules (Source-Destination), an algorithm has been proposed. This algorithm is based on obtaining the communication equations (eq.$_{processes.phase.comm}$) for each phase (Local Equations) of the set of small-scale signatures executed, which identifies the communication pattern for each phase. From these Local Equations, the General Equations are modeled, which are used to predict the communication pattern for a N number of processes. Fig. 4 shows an example, where it has been considered that each phase has only one communication, and therefore one local equation. Once the local equations have been obtained for each phase of each signature, they are analyzed to model the General Equations (GE_{Ph_i}), as is shown in the figure for phase 1. The algorithm converts the source and destination of the

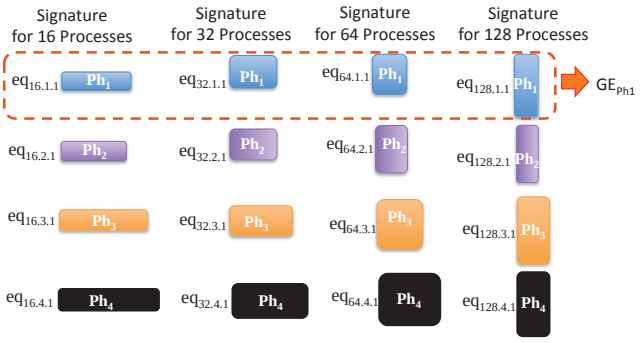


Fig. 4: Obtaining the General Equations from the Local Equations

communications from decimal to binary to work at bit level.

4.1 Generating the Local Equations

This stage is composed of two steps: a first step of analysis, in which the information obtained in the characterization stage is analyzed to obtain information about the communication pattern of each phase, and a second step of modeling, where the Local Equations for each phase are generated. The Local Equations are a representation of the communication pattern of a phase for a specific number of processes. During the analysis step, the dependencies between processes, the pattern type: Static (Mesh, Ring, etc.) or Dynamic (Exchange, Permutation, etc.), and the distance matrix between processes are obtained for each phase. All this information is provided to the second step to generate the Local Equations. In this second step, the Local Equation of each communication of the phase is obtained using an algorithm of identification. This algorithm is based on the fact that the application is well developed, and it executes a deterministic communication pattern for all the processes, without non-predictive conditional sentences as the number of processes increases. The algorithm compares the source-destination of each send primitive for all the processes of the phase, to identify the specific rule to obtain the destination from the source for that number of processes. Moreover, the repeatability of a set of communications is sought to generate easier equations and simplify the analysis and modeling for the General Equations. Finally, the Local Equations for each communication are generated.

The algorithm uses two different structures to generate the Local Equations, because the way to predict the communication pattern is different, depending on the pattern type. If the pattern is dynamic, the way to obtain the destination processes is based on the exchange of certain numbers of source bits, which are called *bits involved*. For this type of pattern, the EC1 structure is used. In case of static patterns, to obtain the destination processes, the distance between the processes and the repeteability of the communications are identified. For this type of pattern the EC2 structure is used. The EC1 structure has as parameters the phase number (#Phase), the number of communication in the phase (#Comm), the algorithm type (Exchange, Permutation) and the list of *bits involved*, (*EC1(p) = {#Phase, #Comm, Type , List of bits involved }*). The EC2 structure has the number of phase, the number of communication in the phase and the list of communication distances and its number of repetitions (*EC2(p) = {#Phase, #Comm, list[communication distances{#repetition}] }*).

Fig. 5 shows a brief example of the procedure. We have a phase with 8 processes (p=8) and three communications. These three communications compose the communication pattern of the phase, which is static because it is a 4x2 mesh, identified in the analysis phase. Then, the EC2 structure will be used. If we focus on the first communication of the pattern (Step 1), we generate the matrix distance between the source and the destination (Step 2). Then, we search for repetitions, in this case, the sequence {+1,+1,+1,-3} is repeated two times (Step 3), once for processes from 0 to 3 and another for processes from 4 to 7. Once we have the sequences and repeteability, we create the Local Equation with the structure of communication EC2 (Step 4).

Depending of the communication pattern, it can be possible that the processes of the phase do not have the same number of communications. With the aim of obtaining the correct Local Equations, all the processes of the phase must have the same number of communications for all the processes, because the algorithm could not relate the communications properly.

To solve this problem, the algorithm selects the process with the maximum number of communications, and it completes the phase structure for the other processes with null communications, until all the processes have the same number. To complete the phase structure with null communications, the algorithm is based on the fact that the application is written in a deterministic way and the communication events follow a logical order with a specific behavior.

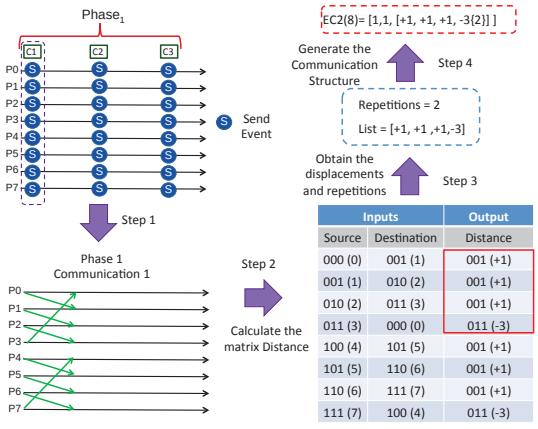


Fig. 5: Example of generating the Local Equations

4.2 Modeling the Global Equations

From the Local Equations, the General Equations of each phase are modeled, which will be used to predict the communication pattern for a N number of processes. To generate the General Equations of each phase, the Local Equations are analyzed in order to model the evolution of the communication pattern. The method consists of comparing the Local Equations to model by a function, as the parameters change their values as the number of processes increases, as is shown in fig. 6. The General Equations have the number of processes to predict as input. The structure of these equations is the same as the one for the Local Equations, the difference is that the parameters have been modeled as a function. Finally, this structure will be simplified to manage easier equations to use.

In some applications, when the number of processes increases, the communication pattern expands communicating with new processes, and new communications appear. To predict these communications, the algorithm models the behavior of how these new communications will appear (number of communications and their destination) for N processes. Fig. 7 shows this procedure. The signature traces of processes 2, 4, 8 and 16 were obtained by the signature executions, and we want to predict the communication pattern for 64 processes. As we can observe, when the number of processes is increased per two, a new communication appear. To predict the communication pattern for 64 processes, first of all, the algorithm models a function to predict the number of communications of the phase as the application scales. The function has as input parameter the number of processes to predict the number of communications. When we the number of communications of the phase has been obtained, we apply the General Equations to predict the destination.

Once we have modeled the General Equations and the communication volume equations, which are obtained by regression models, we have evolution of the communication pattern (spatial and volume) for each phase of the application. These equations will be used to generate the STL.

5. Experimental validation

In this section, the scalability communication model has been validated. Of all the experiments that we have made, we present the BT and CG from the NPB NAS [10] suite, using as input class D, and the applications: Sweep3D [11] and N-Body. We have selected this set of applications because of their distinctive behavior. As an experimental environment,

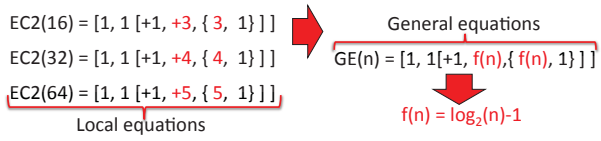


Fig. 6: Modeling the General Equation from Local Equations

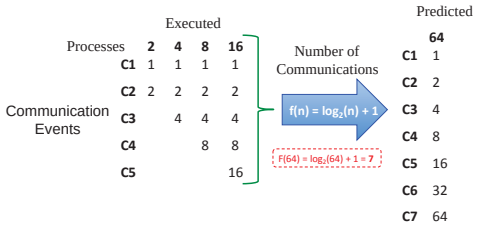


Fig. 7: Generating new communications

a cluster of 8 nodes with 64 processors AMD Opteron 6262 (512 cores) was used.

To carry out the experimental validation, we follow this workflow:

a) We executed five signatures for a small-scale and we obtain their physical traces. Four signatures are used to generate the model and the last one to validate our model before predicting. We executed the signatures with a 1:1 mapping (one process per core).
b) We generate the Local Equations for each phase of the four signatures executed.
c) We model for each phase the General Equations from the Local Equations, and the regression equations of the communication volume.
d) To validate our model, we use the fifth signature to compare whether the models that we predicted are correct. If the predicted values are correct, we use these models to predict for a greater number of processes. Otherwise, we use the fifth signature to improve our model and we create another signature with a greater number of processes to validate the model. We consider that the generated model is correct if we are able to predict the communication pattern without error, and the communication volume with an error less than 10%.
e) Finally, we use these models to generate the Scalability Logical Trace (STL) for a N number of processes.

To validate the generated STL, we compared them with the traces obtained through PAS2P tool. We are only interested in their logical information, so for executing the signatures, we allocate with an x:1 mapping (more than one process by core). By lack of space, we focus on showing the experimentation of process 0.

For BT, we executed the small-scale signatures for 16, 36, 64, 81 and 100 processes. We obtained 6 phases. We used the signatures from 16 to 81 processes to generate the model and the fifth signature to validate the results. Using the generated model, we predicted the communication pattern (spatial and volume) for 1024 processes.

Once we executed the signatures, we generated the communication model for each phase of the application. The predicted values of the communication pattern (Dest.) were obtained by the General Equations, which are shown at the left of Table 1. Due to the type of pattern (static), we used the EC2 type structure to model the General Equations

Table 1: Generated Communication Model and predicted communication pattern from these equations

Communication Model			Real Phases			Predicted Phases			Prediction Error
Phase ID	Global Comm. Equations (Dest.)	Comm. Volume Equations (Bytes)	MPI Prim.	Src-Dest.	Comm Volume	MPI Prim.	Src-Dest.	Comm. Volume	Comm. Volume
BT for process 0 with n = 1024 processes									
1	1)$f(n)=1$	$y(n)=4E+07n^{(-0.997)}$	1)Isend	0-1	40,560	Isend	0-1	39,883	1.69%
	2)$f(n)=\sqrt{(n)}-1$	$y(n)=4E+07n^{(-0.997)}$	2)Irecv	0 -31	40,560	Irecv	0-31	39,883	1.69%
	3)$f(n)=\sqrt{(n)}-1$	$y(n)=4E+07n^{(-0.997)}$	3)Wait	0-31	40,560	Wait	0-31	39,883	1.69%
2	1)$f(n)=n-\sqrt{(n)}+1$	$y(n)=7E+06n^{(-0.997)}$	1)Isend	0-993	6,760	Isend	0-993	6,979	3.13%
	2)$f(n)=17+2(\sqrt{(n)}-9)$	$y(n)=7E+06n^{(-0.997)}$	2)Irecv	0-63	6,760	Irecv	0-63	6,979	3.13%
	3)$f(n)=17+2(\sqrt{(n)}-9)$	$y(n)=7E+06n^{(-0.997)}$	3)Wait	0-63	6,760	Wait	0-63	6,979	3.13%
Sweep3D for process 0 with n = 4096 processes									
1	1) if $log_2(n) mod 2 = 0 \rightarrow, f(n)=\sqrt{(n)}$	$y(n)=6E+07n^{(-1)}$	1)Send	0-64	15120	Send	0-64	14648	3.2%
	if $log_2(n) mod 2 != 0 \rightarrow, f(n)=\sqrt{(2*n)}$								
	2)$f(n)=1$	$y(n)=6E+07n^{(-1)}$	2)Recv	0-1	15120	Recv	0-1	14648	3.2%
2	1)$f(n)=1$	$y(n)=6E+07n^{(-1)}$	1)Send	0-1	15120	Send	0-1	14648	3.2%
	2) if $log_2(n) mod 2 = 0 \rightarrow, f(n)=\sqrt{(n)}$	$y(n)=6E+07n^{(-1)}$	2)Recv	0-64	15120	Recv	0 -64	14648	3.2%
	if $log_2(n) mod 2 != 0 \rightarrow, f(n)=\sqrt{(2*n)}$								
CG for process 0 with n = 4096 processes									
1	1..18)[$\#Comm., c(n)=log_2(n)/2$,	$y(n)=8E+08n^{(-1)}$	1)Isend	0-1	187504	Isend	0-1	195312	4%
		$y(n)=8E+08n^{(-1)}$	2)Irecv	0-1	187504	Irecv	0-1	195312	4%
	$f(y)_{y=1..\#Comm.,c(n)}=2^{(y-1)}$],	$y(n)=8E+08n^{(-1)}$	3)Wait	0-1	187504	Wait	0-1	195312	4%
		...	...	...	...	...	...	...	...
		$y(n)=8E+08n^{(-1)}$	16)Isend	0-32	187504	Isend	0-32	195312	4%
		$y(n)=8E+08n^{(-1)}$	17)Irecv	0-32	187504	Irecv	0-32	195312	4%
		$y(n)=8E+08n^{(-1)}$	18)Wait	0-32	187504	Wait	0-32	195312	4%
N-Body for process 0 with n = 4096 processes									
1	1)$f(n)=1$	$y(n)=1E+07n^{(-1)}$	1)ISend	0-1	2342	ISend	0-1	2441	4.0%
	2)$f(n)=n-1$	$y(n)=1E+07n^{(-1)}$	2)IRecv	0-4095	2342	IRecv	0-4095	2441	4.0%
	2)$f(n)=n-1$	$y(n)=1E+07n^{(-1)}$	3)WaitAll	0-4095	2342	WaitAll	0-4095	2441	4.0%
	4)$f(n)=0$	$y(n)=1E+07n^{(-1)}$	4)WaitAll	0-0	2342	WaitAll	0-0	2441	4.0%

to predict the communication pattern. This structure has been simplified and specified for process 0 for readability. Moreover, in this table, we show the communication volume regression equations. All these equations have the number of processes to predict as their input parameter.

At the right of Table 1, we show the real and predicted communication patterns of phase 1 and 2 for the process 0. The predicted communication pattern was predicted by means of providing the parameter 1024 (number of processes) to the General Equations.

The predicted communication pattern corresponds with the real one for both phases. The communication volume was predicted with an error of about 2% for the first phase and 3% for the second phase. For the other four phases of the application, the communication pattern was predicted without error for all of them. In the top of Table 2, we present a summary of these phases with the predicted error of communication volume. We show the maximum error of all the communications of the phase. As we can see, the communication volume was predicted with a maximum error of about 4% (phase 5).

For CG, we executed the small-scale signatures for 16, 32, 64, 128 and 256 processes. This application has 3 phases. We used the signatures from 16 to 128 processes to generate the model and the fifth signature to validate the results. We predicted the communication pattern for 4096 processes. At the right of Table 1, we show the real and predicted trace of process 0 for phase 1. Due to the similarity between the phases, we only show phase 1. At the left of Table 1, we show the generated equations of communication for this phase. Due to the type of pattern (dynamic), we used the EC1 structure to model the General Equations. For this application, the communication pattern expands, communicating with more processes as the application scales. For this reason, we also have to model an equation to predict the number of communications of the phase. The general equation calculates the number of communications of the phase and their destinations. First of all, the number of communications is predicted using the equation $\#Comm.c(n) = log_2(n)/2$, this equation calculates the number of times that the sequence Isend, Irecv and Wait repeats in the phase. Applying this equation for 4096 processes, the sequence is repeated 6 times. Then, we apply the equation $f(y)_{y=1..\#Comm.c(n)} = 2^{(y-1)}$ to calculate the destination of the sequence, obtaining a destination sequence of 1, 2, 4, 8, 16 and 32. The communication volume was predicted with an error of about 4% for all the communications of the phase.

For the other two phases of the application, the communication pattern was predicted without error for all of

them. In Table 2, we present a summary of these phases. The communication volume was predicted with a maximum error of about 6% (phase 2).

For Sweep3D, we executed the signatures for 16, 32, 64, 128 and 256 processes. This application has 4 phases. The fifth signature was used to validate the model. We predicted the communication pattern for 4096 processes. At the right of Table 1, we show the real and predicted trace of process 0 for phases 1 and 2. The generated equations are shown in the left of Table 1. Due to the type of pattern (static), we used EC2 type structure. The pattern obtained by the General Equations corresponds to a pipeline wavefront. As we can see, the predicted pattern corresponds with the real one for both phases. The communication volume was predicted with an error of about 3.2% for both phases.

For the other two phases of the application, the communication pattern was predicted without error for all of them. At the bottom of table 2, we present a summary of these two phases. As in phases 1 and 2, the communication volume was predicted with an error of about 3.2%. This is because the application sends in its messages the same volume of bytes in all its phases.

For N-Body, we executed the signatures for 16, 32, 64, 128 and 256 processes. This application has 1 phase. We used the signatures from 16 to 128 processes to generate our model and the fifth signature to validate the results. We predicted the logical trace for 4096 processes. At the left of Table 1 we show the generated equations for this phase for process 0. Due to the type of pattern (static), we used the EC2 structure to model the general equations.

The communication pattern corresponds to a pipeline. The process 'x' communicates with the process 'x+1' until the last process, which communicates with the first process. At the right of Table 1, we show the real and predicted trace of process 0 for the phase 1. The phase has two WaitAll primitives. The first WaitAll primitive waits until the Irecv releases its request, while the second WaitAll waits until the Isend releases its request. This second WaitAll is written in the application in order to synchronize the application. The communication volume was predicted with an error of about 4%.

Table 2: Summary of prediction error for the rest of phases

Phase Number	Comm. volume equation (Bytes)	Comm. volume (Error %)
Summary of phases of BT (from 3 to 6)		
Phase 3	$y(n) = 4E + 07n^{(-0.996)}$	2.7%
Phase 4	$y(n) = 4E + 07n^{(-0.997)}$	3.1%
Phase 5	$y(n) = 5E + 07n^{(-0.997)}$	3.6%
Phase 6	$y(n) = 4E + 07n^{(-0.996)}$	2.7%
Summary of phases of CG (from 2 to 3)		
Phase 2	$y(n) = 8E + 08n^{(-0.995)}$	5.8%
Phase 3	$y(n) = 8$	0%
Summary of phases of Sweep3D (from 3 to 4)		
Phase 3	$y(n) = 6E + 07n^{(-1)}$	3.2%
Phase 4	$y(n) = 6E + 07n^{(-1)}$	3.2%

6. Conclusions and future work

We propose a methodology to analyze and predict strong scalability behavior for MPI applications on a given system. It strives to use a bounded analysis time, and a reduced set of resources. The methodology has been explained focusing on presenting the scalability communication model, to predict the evolution of the communication pattern of each phase as the application scales.

As future work, we are analyzing the internal behavior of the collective MPI primitives. Internally, the collective MPI primitives make point-to-point communications to exchange information between all the processes. We are analyzing this kind of primitives, to expand the model to consider the evolution of the internal communications of the Collective MPI primitives.

Acknowledgment

This research has been supported by the MINECO (MICINN) Spain under contract TIN2011-24384

References

[1] N. Attig, P. Gibbon, and T. Lippert, "Trends in supercomputing: The european path to exascale," *Computer Physics Communications*, vol. 182, no. 9, pp. 2041 – 2046, 2011.

[2] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature for performance analysis and prediction," in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014 (Accepted).

[3] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "A tool for selecting the right target machine for parallel scientific applications," in *ICCS*, 2013, pp. 1824–1833.

[4] R. Nishtala, P. Hargrove, D. Bonachea, and K. Yelick, "Scaling communication-intensive applications on bluegene/p using one-sided communication and overlap," in *IPDPS 2009*, 2009, pp. 1–12.

[5] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "Analysis of scalability: A parallel model approachâĂİ," in *CLUSTER*, 2014, pp. 294–295.

[6] X. Wu and F. Mueller, "Scalaextrap: Trace-based communication extrapolation for spmd programs," *ACM Trans. Program. Lang. Syst. Article 5*, vol. 34, no. 1, 2012.

[7] J. Zhai, T. Sheng, J. He, W. Chen, and W. Zheng, "Fact: fast communication trace collection for parallel applications through program slicing," in *SC*, 2009.

[8] B. J. Barnes, J. Garren, D. K. Lowenthal, J. Reeves, B. R. de Supinski, M. Schulz, and B. Rountree, "Using focused regression for accurate time-constrained scaling of scientific applications," in *IPDPS*, 2010, pp. 1–12.

[9] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Euro-Par*, 2005, pp. 196–205.

[10] D. Bailey, E. Barszcz, J. Barton, and D. Browning, "The NAS Parallel Benchmarks," *International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 66–73, Jan 1991.

[11] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional," *Journal of High Performance Computing Applications*, vol. 14, pp. 330–346, Jan 2000.

Reusability of DDS Information-Model for Distributed VRE

Hassan Haidar*
Lab-STICC-CERV ENIB
Brest, France
haidar@enib.fr

Ali Kalakech
Lebanese University
Beirut, Lebanon
akalakech@ul.edu.lb

Ali Hamie
Lab-CRITC AUL
Beirut, Lebanon
hamie@aul.edu.lb

Ronan Querrec
Lab-STICC-CERV ENIB
Brest, France
querrec@enib.fr

***Abstract*— Virtual Reality Environments (VRE) which simulate reality and thus present a safer learning environment, are increasingly being adopted to simulate complex systems. Such systems make the process of engineering virtual environments a complex task, especially due to the abundance of dynamic data types like behaviors. In parallel, distribution services have become essential following advances in telecommunications and the subsequent demand on mobile technologies. Hence, middleware enables technologies to provide such services to existing and newly-developed applications. Data Distribution Service (DDS) is a middleware standard for real-time applications based on a peer-to-peer architecture. DDS requires awareness about the type of distributed data which is achieved by defining an information-model. Consequently, distributing VRE using DDS complicates the development process of its information-model in order to meet the requirements of complex data types. In this paper, we propose a generic information-model in order to facilitate the structuring of domain models and ensure their reusability.**

Keywords— Middleware; DDS; IDL; Distrributed Virtual Reality Environment; Meta-Model; UML;

I. INTRODUCTION

Recent advances in telecommunication and mobile technologies are ever more leading to the continued connection of different elements of society (people, applications, data, and objects) in one big network [1]. This technology is evolving every day, even connecting objects to public networks over the internet making those objects programmable, intelligent and capable of interacting with users in the community as part of the concept of the "Internet of Everything (IoE)". These innovations are creating new opportunities in different fields such as industry, biomedicine, education and e-Learning. Mobility of both users and applications has also brought about such advances, and geographical distances have been drastically reduced due to distribution capabilities.

On the other hand the demand for virtual reality environments, such as artificial computer-generated environments, has increased over the past couple of years. This is due to their ability to depict a wide variety of domains and to offer new possibilities for experimentation. They present an alternative to real-life testing and training by introducing new types of investigation for complex systems in fields like industry, avionics, medicine and the military [2] [3].

Applying the advances in networking and telecommunication to virtual reality increases the need for "Distributed Virtual Environments (DVE)". DVE offers the possibility of running the simulated environment on multiple computers connected over a network. Using multiple nodes to share the same environment can facilitate user interaction as well as enabling simultaneous actions from different users, allowing them to cooperate and collaborate interactively to complete an action [4]. As stated in [5], "*each user uses his own computer to have individual interaction capabilities or to meet others if they are not located at the same geographical place*".

In this regard, middleware enables those technologies which are catching the attention of computer scientist for their ability to provide distribution services to both newly-developed and existing applications without the need for re-engineering. This fact is an important metaphor for VRE as its implementation is intricate as it simulates such complex systems. Moreover, the increasing number of users for such environments has led to a change in infrastructure: from a client/server (C/S) centralized-basis to decentralized peer-to-peer (P2P) [6] [7]. Such change is mainly due to the growing demand on scalable architectures.

In order to satisfy this growing demand whilst also considering scalability, the Object Management Group (OMG) released DDS middleware as a standard for distributing large-scale and real-time applications [8]. It is based on a pure P2P design without the need for central servers or special node roles, nor for any broker to distribute data from the source to its destination. DDS is also categorized as data-centric middleware [9] that requires awareness about distributed data. This awareness is gained through engineering an information-model that contains a representation of data types to be distributed. It can be represented by an Interface Definition Language (IDL) file. The same IDL file can be compiled to different languages (C, C++, and Java). Besides its advantages, the information model in the IDL file represents a domain-specific model where data is structured according to the needs of each application. Thus distributing VRE, which describe complex systems, using DDS requires a re-engineering of the IDL file as per each environment. Knowing that each virtual reality (VR) application has been specified by a domain-expert, each application therefore requires intervention from an expert to engineer its corresponding IDL file, which is a major

limitation. This limitation hinder the reusability of the IDL structure and thus limits the reusability of DDS middleware. However, the author in [10] considers reusability to be one of the main common aspects that must be considered when designing an architecture for distributed VRE.

In this paper, our aim is to propose a generic algorithm for DDS information-model in order to automatically distribute virtual environments without the need for expert intervention. These algorithms use a generic data structure in the IDL file based on the Unified Modeling Language (UML) meta-model, which facilitates the engineering process and makes the model reusable.

As DDS middleware is data centric, the distributed data has to be explicit. For this reason, throughout our work, we use the MASCARET framework [11] (Multi-Agent Systems to simulate Collaborative, Adaptive and Realistic Environments for Training). The principle of MASCARET is to use a meta-model based on UML to develop the different VRE. This means that each VRE is defined by a UML model and then automatically executed. The corresponding model is therefore explicit during the simulation. This principle will enable us to link the explicit definition of the VRE with the DDS information-model. Moreover, we suggest using the explicit knowledge about the data to be distributed, which can be provided by such environments, to define both generic algorithms and certain distribution strategies.

The rest of this paper is organized as follows: Section II gives overview of the DDS standard and its workflow in order to design a DVE. Section III highlights the role of the abstract layer and proposes similar abstraction for the DDS information-model. Section IV details the generic information-model and its concept of reusability. Section V offers a practical example, using our suggested model to distribute a simplified VRE. Section VI concludes this paper, showing the result we reach through our proposed solution and discusses new strategies for controlling the distribution of VRE using semantics.

II. Data Distribution Service and VRE

A. DDS Overview

DDS is an OMG standard released in 2004 for distributing real-time applications [8]. It is categorized as an indirect-communication paradigm that uses a publish-subscribe architecture [12]. It standardizes both an API and a wire-protocol. It specifies two levels of interfaces: DCPS (Data-Centric Publish-Subscribe) as a communication model interface, and DLRL (Data-Local Reconstruction Layer) as an optional higher level interface to simplify integration into the application layer.

DDS publishers (pub) and subscribers (sub) use data-writers and data-readers to write and read updates. Writing and reading occurs through a logical global data space (GDS) representing the data bus that is accessed through applications from heterogeneous platforms. Its indirect-communication paradigm allows for decoupling in both space and time. This means pub and sub identities do not need to be known to each other as there is no server registration. They also do not need to exist at the same time, as data can be retained for late joiners.

A distinguishing characteristic of DDS is the information-model that specifies the type of information exchanged over GDS. This model is defined through an IDL file to represent domain-specific types where a specific data structure is defined for each application. This structure is based on Topics and Keys. The Topic represents the unit of information type while the Key represents the unique instance identifier of this Topic. Such models also offer a type-specific safety feature which identifies errors in the development cycle of the API. Data structured in the IDL file is independent of the programming language. Thanks to the compilation process undergone by this file, it is possible to generate DDS API files in different languages such as C, C++, CS and Java.

Furthermore, DDS presents a rich set of QoS in order to ensure simulations are distributed in real-time. Several aspects can be configured, e.g. durability, history, latency budget, ownership, reliability. They can also be configured at different DDS entity levels, e.g. topics, publisher, subscriber, data-writer.

This paper focuses on the engineering process of the information-model within the IDL file which is required to operate the DCPS interface of DDS, a topic that will be fully covered in Section III.

B. DDS Workflow used for VRE

Although using middleware helps to provide existing applications with distribution services without the need for re-engineering, the fact that it is a data-centric technology creates another complication. This complication can be illustrated in the complexity of modeling an information-model for complex systems. Upon distributing behavior-rich VRE environments with DDS, a specific IDL file needs to be engineered for each environment. When we look more closely at the engineering workflow for a VRE with DDS on the modelling level, we note the following:

- **VRE Level:** VR experts write the code required in order to define the VR application model
- **Middleware Level:** Define the Information Model required for this environment in order to grant DDS with awareness about data types exchanged. This can be done through the IDL file

The result of this workflow is two models for each environment. In the next section, it will be shown that the complexity of the first model is solved through an abstraction layer, while the second complexity remains valid. Thus, in our work, we also provide a solution to ease the engineering process of the second model while ensuring its reusability.

III. Abstraction Proposal

In this section, we first highlight the conceptual UML meta-model that is used as an abstract layer to VRE models. Then we propose a similar abstraction layer for releasing a generic information-model for DDS, which is the major contribution of our work.

A. Abstract layer for VRE

The model-driven approach facilitates the development of VRE by representing them in human-oriented, intelligent, and semantically rich environments rather than simple computer-generated ones. This knowledge approach, along with artificial intelligence, introduced what are called Intelligent Virtual Environments (IVEs) [13].

In the scope of virtual reality and intelligent virtual environments, objects are characterized by having not only geometrical properties, or what is called 3D element parameters, but also behavioral properties [14]. The development workflow of such behavior-rich environments leads to certain complications and it is time consuming to develop domain models for each application. For that, a higher level of abstraction was proposed to produce a model-driven approach based on UML meta-models for VRE [11].

In this regard, conceptual models are being adopted in virtual reality platform architecture to facilitate the creation of complex models. The UML meta-model is a model-based approach proposed by previous work in virtual reality to define semantics. It provides a knowledge-driven perspective to the contents of the virtual environment including the ontology of the domain, behavior of the entities, structure of the environment, and users' and agents' interactions and activities. Environments using this methodology are described as environments rich in "Knowledge" about the instantiated applications models. In the field of multi-agent systems, which overlaps with Intelligent Virtual Environments (IVEs), different authors have chosen to ground their meta-models in UML, like in [15] [16] [17], and their work can be partially reused or extended. In our work, we use the MASCARET framework whose meta-model is also based on UML.

The common line of interest we find with this approach is that instead of rewriting an IDL file, we propose to use such conceptual models to structure a generic information-model. The meta-model contains all types of data that might be required to distribute a virtual reality environment, including a representation of both the static and dynamic parts of the model. In the next sub-section, we propose mapping the knowledge represented by the framework meta-model along with the awareness required by DDS represented by the information-model (IDL). The main contribution we have achieved is to intelligently distribute VRE while facilitating the engineering process of the IDL file and ensuring its reusability.

B. Abstract layer for IDL

In terms of the methodology for VRE modeling, it can be interesting to adopt a similar approach to designing IDL files.

Some attempts have been made to automatically generate the IDL from UML profiles or XML [18], but such attempts require special tools like OpenCCM or TUPI and sometimes the UML model needs to be refined. In addition, several problems arise in recognizing all UML associations, enumerations, organization of data within the IDL, etc. Moreover, the data inside the IDL is not completely identical to the UML model. For this reason, it can be difficult to automatically generate the IDL file from the UML model. Thus, designing a "generic" information-model in the IDL file for distributing VRE requires a generic data structure compliant with the different concepts of virtual reality environments so as to contribute to the different data types.

Although the DDS information-model is a domain-specific structure, the solution we propose is to engineer the IDL file according to the abstract layer of VRE. In other words, we structure data in the IDL file as per the meta-model level rather than structuring it as per each environment model. As long as application models are instantiated from the meta-model, then the proposed generic IDL structure contributes to data types found in the application model.

The MASCARET meta-model, which is based on UML, contains high level information corresponding to environment semantics. In order to do so, the proposed IDL structure provides DDS with generic knowledge about data types found in virtual reality and as a result provides with DDS the ability to distribute it. This knowledge includes both the static and dynamic parts of the environments. This step makes the IDL easier to engineer while simultaneously enabling re-usability of the IDL with multiple VREs.

The next section examines the link between this meta-model and the DDS information-model in order to devise our proposed model.

IV. GENERTIC IDL FILE

We generally classify four main concepts presented by the MASCARET meta-model in engineering of VRE: "3D elements", "Properties", "Behaviors" (Operation, state machine, etc.), and "Events". For each concept, we provide the corresponding sub-part of the meta-model and we associate it with the right DDS information-model. The resulting IDL file is then engineered in a generic manner according to the meta-model. As VRE models are instantiated from the meta-model, the same IDL file can be reused by multiple VRE as it contributes to data types required for each domain-specific application model. This signifies the awareness required for DDS to distribute VRE.

A. 3D-Element

For virtual reality environments, 3D representation is of course used with objects that can be graphically represented. In MASCARET, these 3D objects are called "Entities". Fig.1 provides an extract of the Entity concept of the MASCARET meta-model.

All Entities have a Position, an Orientation (with referential point) and a shape (types of topological specification). As its name indicates, Position (pos) provides the position of the entity in terms of the x-axis, the y-axis and the z-axis. Orientation (orient) provides the orientation of the entity according to quaternions [19]. Both are of the type vector3. Shape geometrically presents entities by allocating color, scale, etc. [20]. The corresponding information-model mapped with the model in Fig.1 is as follows:

Generic IDL Structure for 3D-Element

```
struct vector3    {
    double x,y,z; };
```

```
struct Entity3D   {
    string entityName;
    vector3 pos;
    vector3 orient;
    string shape;  };
#pragma keylist Entity3D entityName
```

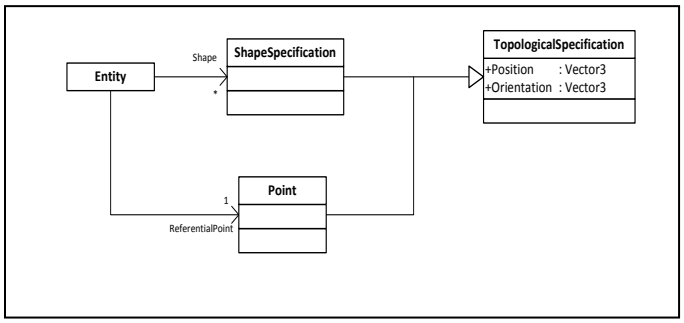


Fig. 1. Extract of the entity class of UML meta-model

B. Property

The next MASCARET concept we address after "3D-element", is "Property". "Property" has a name and a "Slot" which contains the value of a structural feature of a class instance. This concept is commonly used in VR applications to complement the physical appearance in 3D as previously mentioned.

Below are two simplified figures provided as an extract of the "Property" concept of the UML meta-model. Fig. 2 shows the inheritance of Property and Class from the Element class and thus can be identified by a name. A Property represents all the attributes of a given class, from primitive types to complex types (Class). Fig. 3 shows the instance level of a Property. "Class" is used to describe some classes of the domain-specific model. The "InstanceSpecification" class is an instance of class model where the type of instance is based on its classifier and the attributes are called slots. The "ValueSpecification" represents the different values (int., string, etc.). The corresponding information-model, mapped with the models in Fig.2 and Fig 3 is as follows:

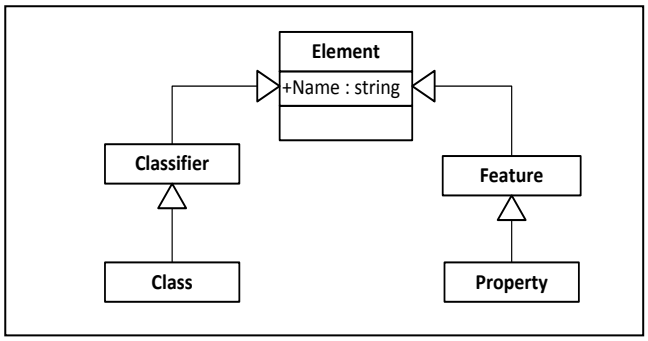


Fig. 2. Extract of the Property Concept of MASCARET meta-model at the class level

Generic IDL structure for Property

```
struct Property        {
  string entityName;
  string propertyName;
  string propertyValue;};
#pragma keylist Property entityName
propertyName
```

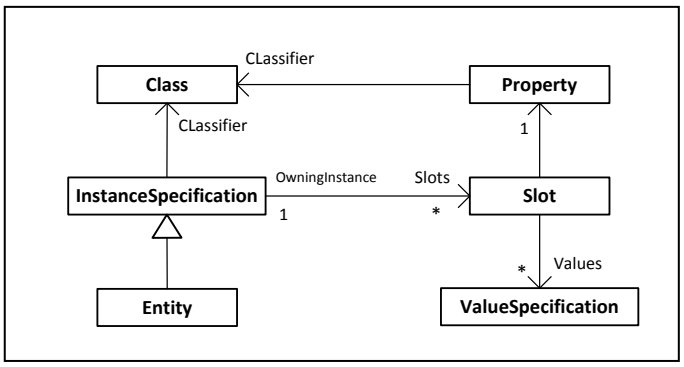


Fig. 3. Extract of the Property Concept of MASCARET meta-model at the instance level

C. Behaviors

The third MASCARET concept we address is "Behavior". This concept describes the dynamic part of the virtual reality environment. In this section, we define the generic information-model of DDS that is capable of distributing the dynamic part of the virtual environment.

The description of a behavior is static and does not change during run time while the execution of a behavior, represented by "Behavior Execution", can be executed several times during the simulation. Behavior Execution is a kind of execution specification based on a sequence of conditions and actions which is used to explore the dynamic nature of behaviors. It therefore describes a specific behavior and makes it possible to introspect the features of entities. It is also considered to be explicit knowledge with a start and a stop featuring a timestamp, result, etc. Without this execution, it is not possible to formalize the effect of behaviors and the modification of the structural properties of entities. Due to this fact, we find it interesting to distribute data about both "Behavior execution" and its context as it makes it possible to share the same execution of behaviors in distributed environments.

The execution of a behavior can be formalized by four kinds of behavior type: Operation, OpaqueBehavior, Activity and StateMachine, (Fig.4). Thus creating an information model for distributing behavior concept of MACARET meta-model is actually destined to distribute the execution of the different behavior types formalized.

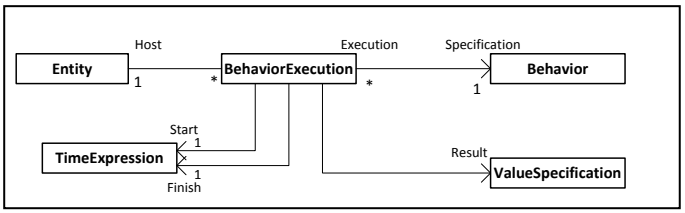


Fig. 4. Extract of the Behavior Concept of UML meta-model

The information-model mapped with the model in Fig.4 is as follows:

Generic IDL structure for Behavior Execution

```
struct Parameter        {
      string name;
      string value;     };
struct BehaviorEx       {
      string entityName;
      string behavior;
      Double start;
      Double stop;
      Parameter parameter;    };
#pragma keylist BehaviorEx entityName
behavior parameter.name
```

1) In order to distribute an **Operation**, we exchange messages about the "behavior execution" and mention: the entity realizing this operation (entityName), the name of the operation (Behavior), the start, and the stop of it. Actions, which are also behaviors carried out as a consequence of this operation, are distributed using the same "behavior execution" message.

2) **Opaque behaviors** are an exception because the behavior is 'opaque'. This means it can be invoked online but it cannot be introspected. We can know if this behavior started or stopped and in which context, but we cannot know the sequence of actions executed as a result of this behavior. This also includes messages about the behavior.

3) The execution of an **Activity** triggers the execution of a sequence of actions belonging to that activity. But actions are also behaviors. Therefore, to distribute information about both the activity and the actions, we can exchange information about the "behavior execution".

4) **StateMachines** describe interactions as asynchronous reactive models used for modelling the reaction of an entity to events. To distribute information about this type of behavior, we use the same "behavior execution" message to indicate the start of a state machine. However, a state machine can have multiple transitions from one state to another during run-time, therefore simply sending the start of a state machine behavior is not sufficient. Hence, when a state machine changes state, we need to send this change. Consequently, we add a "parameter" structure to the informational-model to represent the transitions between States of the State Machine. In this case, the Stop attribute, used with other behavior executions, is not significant as an entity cannot be in multiple states simultaneously. The effect of a new state might be to carry out a new behavior or make a new transition or a "Do" function. In all that, the same "behavior execution" message can be exchanged.

D. Events

Events are another concept of UML which are used in MASCARET to represent the lowest information level that produces a consequence at a particular point in time. As found in [21], events are atomic occurrences without duration.

Events are a subclass of the Element class and thus can be identified by their names. Upon issuing an event, it enters a pool of events until it is processed. A consumed event is no longer available for processing. The effect of an event differs according to its type or subclass. As shown in Fig.5, there are four fixed types of Events [22].

- **Call Event:** Invokes Operation (Behavior Execution)
- **Signal Event:** A signal generated by Actions that are part of an Activity (Behavior Execution)
- **Change Event:** An implicit event that occurs when an explicit Boolean expression becomes true. Usually this type of events uses the keyword "When" followed by a Boolean expression
- **Time Event:** An event that occurs after that an amount of time has elapsed. Example: "After 2 seconds call function drop()"

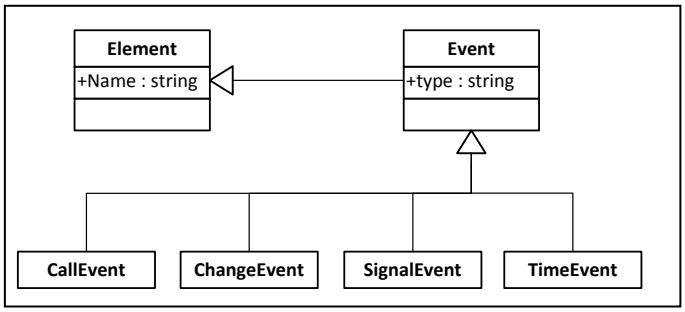


Fig. 5. Extract of Event Concept of MASCARET meta-model

The corresponding information-model mapped with the model in Fig.5 is as follows:

Generic IDL structure for Events

```
struct CallEvent        {
      string caEventName;
      string type;
      string opName;    };
#pragma keylist CallEvent caEventName
struct SignalEvent      {
      string sEventName;
      string type;
      string parameterName;
      string parameterValue; };
#pragma keylist SignalEvent sEventName
struct ChangeEvent      {
      string chEventName;
      string type;
      string changeExpression;};
#pragma keylist ChangeEvent chEventName
struct TimeEvent        {
      string tEventName;
      string type;
      string timeExpression; };
#pragma keylist TimeEvent tEventName
```

E. Other Data Types

In Addition to the different concepts we have covered in this section, there are some other types of data, such as Agent communication and Multimedia, which exist in VRE and which also need to be distributed.

For Agent Communication, in [23] a layered model was introduced which separates low-level data transport issues from high-level semantic interoperability aspects. Such modeling aims to independently optimize agent communication at different levels. Thus, our proposed method here is based on using DDS as the underlying protocol to transport data types according to FIPA[1] structure.

Regarding the multimedia types (e.g audio, video), it was found in [24] that DDS is able to transport these types of media data. Three topics have been created for multimedia types: one for audio signal, the second for audio stream (RTP), and the third for video stream.

In this paper, we merely point out the feasibility of using DDS for such data types but we will not explicitly define their corresponding information-model. For both Agent communication and Multimedia, the defined IDL structure will be generic as the type of data used is fixed, standardized, and general.

V. APPLICATION: CLOCK EXAMPLE

In order to evaluate this generic information-model, we used a simplified VRE based on the MASCARET meta-model to represent a simple example about a programmable plug. The purpose of creating this application is to provide its users with the technical knowledge they require in order to use the plug in reality. Fig.6 features a snapshot of this application along with a sample state machine diagram.

We grouped the different parts of the generic information-model above in one IDL file and compiled it using an IDL compiler to generate DDS API files in the required programming language. In our case, we used SimD as a compiler [25] and C++ as a language. We integrate this API as a plugin in MASCARET so we are able to publish and subscribe to the different DDS topics we declared in the generic IDL file above.

In this paper, we present two scenarios for distributing data about two different DDS topics and show the typical distribution of the "3D representation" and "properties" of this application without the need to develop a specific IDL file. Both scenarios are based on having two different users with their own machines (two different nodes) connected over a WiFi network to operate the Clock application running on their local side.

A. Scenario 3D-Element

For this scenario we have chosen the topic of the 3D-element so we can demonstrate how each user's operations in the distributed environment are synchronized on both nodes.

- User1 on node1 presses the Hour button and, in his local environment (node2), user2 is able to visualize the effect of user1's action on his own environment (on node1)
- User2 on node2 presses the Minute button while user1 on node1 should be able to visualize that the position of this button has changed

In terms of DDS, when user1 presses the hour button, a message is published from node1 about this update. This is represented by the entityName "ButtonHour" and the new position values. The GDS will transmit this message to the subscriber in node2 that will read this update and make the necessary changes, according to the new values received. Then, node2 can visualize the position change of the button as a result of the action of user1. The same scenario is applied for user2's action made on "Button Minute". As a result, the 3D-element topic is generically used for distributing this type of data in VRE regardless of the application-specific model and without the intervention of a computer expert.

B. Scenario Property

For simplification purposes, this scenario continues that of the previous 3D-Element showing the distribution of the Properties of buttons pressed. User1's operation with "Button Hour" increased its value by one. So in addition to the distribution of the position change, using DDS it is also possible to publish the new value for the property "hour". The exchanged message is defined by the property name "hour" and the slot value, which is the new value. User2's node2 detected the position change update and the new value for the hour property, thus applying the same values locally in order to ensure both environments remain synchronized. The same procedure is repeated when user2 operates the "Button Minute".

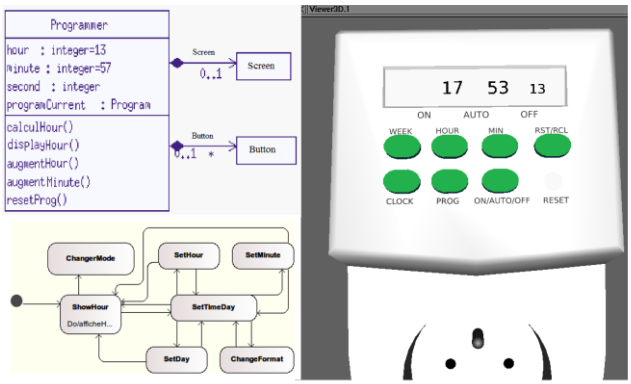


Fig. 6. A scene from MASCARET simplified example

VI. CONCLUSION AND DISCUSSION

In this paper, we proposed a novel abstract layer for the DDS information-model. Our generic information-model (covered in section IV) provides DDS with the awareness, and consequently the capability, to distribute static and dynamic data types of VRE that are modeled based on MASCARET meta-model. Considering that MASCARET application models are instantiated from the meta-model, the information-model we proposed is generic and can be reused by multiple

[1] Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00061/SC00061G.pdf

applications without requiring an intervention from a computer expert when distributing a new VRE. Reusability, which is one of the main key factors for distributing VRE, emphasizes the advantage of using DDS for distributing VRE describing complex systems. Moreover, the safety perspective of the information-model simplifies the development process, especially in terms of behavior data types, and helps avoid development errors.

From a high level perspective, DDS remains a low-level communication API that lacks the intelligence required to control what type of data to distribute and when. It is true that through our generic model, DDS has the capability to distribute data about the different concepts of the MASCARET meta-model. However, sometimes different types can be used to reach the same result. As an example to what we are trying to clarify here, we take the second scenario above where we showed the distribution of new properties using the generic DDS topic "Property". Here we see that we can also use the "behavior execution" of the operation "press" so we can make both environments perform the same function. The consequence of this operation execution is the simultaneous increase of the property value in both environments.

Thus, the question remains as to which topic to select and when. From here, we can conclude the need for generic rules and strategies capable of controlling the distribution process at a high level. To deliver such practice with pure peer-to-peer architecture like DDS, we suggest using explicit knowledge to identify data according to the user's interest and need. This requires both a dynamic filtering mechanism and a dynamic criterion to filter the data.

In this regard, DDS makes it possible to filter data in different ways. Some are static while others, like query-condition, are dynamic and can be modified during the simulation. In parallel, and in order to set an intelligent criterion based on the high-level knowledge of VRE, we propose to use the behavioral knowledge presented by the MASCARET meta-model. The coupling between Query-Condition (DDS) and the awareness about behavior (MASCARET), makes it possible to select data types as per the context of behaviors and during run time. This requires sharing environment knowledge between distributed environments. Our future work will focus on the semantic criteria that offer clues as to how to design global and generic strategies for intelligent distribution of VRE.

REFERENCES

[1] H. Kopetz, "[Real-Time Systems : Design Principles for Distributed Embedded Applications," *second edition Springer,* 2011.

[2] N. Marion, C. Septseault, A. Boudinot and R. Querrec, "GASPAR: Aviation management on aircraft carrier using virtual reality," in *In Proceedings of Cyberworlds*, Hanovre (Allemagne), 2007.

[3] F. Le Corre, C. Fauvel, C. Hoareau, R. Querrec and C. Buche, "CHRYSAOR: an Agent-Based Intelligent Tutoring System in Virtual Environment," in *International Conference on Virtual Learning*, Brasov: Romania, 2012.

[4] S. Guleyupoglu, ""Distributed Collaborative Virtual Reality Framework for System Prototyping and Training".," in *Paper presented at the RTO IST Symposium on "New Information Processing Techniques for Military Systems"*, Istanbul, Turkey, 2000.

[5] C. Fleury, T. Duval, V. Gouranton and B. Arnaldi, "A New Adaptive Data Distribution Model for Consistency Maintenance in Collaborative Virtual Environments," in *Virtual Reality Conference of EuroVR - EGVE - VEC*, 2010.

[6] S.-Y. Hu, J.-F. Chen and T.-H. Chen, "VON: A Scalable Peer-to-Peer Network for Virtual Environments," *Network, IEEE, 20(4):22–31, 2006.,* vol. 20, no. 4, pp. 22-31, 2006.

[7] N. MATSUMOTO, Y. KAWAHARA, H. MORIKAWA and T. AOYAMA, "A scalable and low delay communication scheme for networked virtual environments," in *Global Telecommunications Conference Workshop (GlobeCom)*, 2004.

[8] "DDS, OMG Standard," 2007. http://www.omg.org/spec/DDS.

[9] S. Schneider, "What's the Difference between Message Centric and Data Centric Middleware?," 2012.

[10] S. Torki, "Entités autonomes en environnements virtuels distribué," Toulouse, France, 2008.

[11] P. Chevaillier, T.-H. Trinh, M. Barange, P. DeLoor, F. Devillers, J. Soler and R. Querrec, "Semantic Modelling of Virtual Environments Using MASCARET," in *Proceedings of SEARIS*, Singapore, 2011.

[12] G. Coulouris, J. Dollimore, T. Kindberg and G. Blair, Distributed Systems: Concepts and Design (5th Edition), Addison-Wesley, 2011.

[13] R. Aylett and M. Cavazza., "Intelligent virtual environments - A stateof-the-art report," in *STAR Proceeding of Eurographics*, Manchester, UK, 2001.

[14] J. Odell, M. Nodine and R. Levy, "A metamodel for agents, roles, and groups," in *Agent-Oriented Software Engineering V*, New York, NY, USA, Springer, 2005, pp. 78-92.

[15] J. L. Lugrin and M. Cavazza, "Making sense of virtual environments: Action representation, grounding and common sense," in *ACM Intelligent User Interfaces*, Honolulu, Hawaii,, 2007.

[16] G. Beydoun, G. Low, B. Henderson-Sellers and H. Moura, "Faml: A generic metamodel for mas development," *IEEE Transactions on Software Engineering,* vol. 35, no. 6, p. 841–863, 2009.

[17] H. Van Dyke Parunak and J. J. Odell, "Representing Social Structures in UML," in *Agent-Oriented Software Engineering II*, vol. 2222, Springer Berlin Heidelberg, 2002, pp. 1-16.

[18] T. Nascimento, T. Batista and N. Cacho, "TUPI: Transformation from PIM to IDL," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, vol. 2888, Catania, Sicily, Italy, Springer Berlin Heidelberg, 2003, pp. 1439-1453.

[19] L. Perumal, "Quaternion and Its Application in Rotation Using Sets of Regions," *International Journal of Engineering and Technology Innovation,* vol. 1, pp. 35-52, September 2011.

[20] T. Tutenel, R. Bidarra, R. M. Smelik and K. J. D. Kraker, "The role of semantics in games and simulations," *Computers in Entertainment (CIE),* December 2008.

[21] M. A. Jackson, System Development, Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1983.

[22] J. Offutt and A. Abdurazik, "Generating tests from UML specifications," in *Proceedings of the 2nd international conference on The unified modeling language: beyond the standard*, Fort Collins, CO, USA, 1999.

[23] H. Helin, "Supporting Nomadic Agent-based Applications in the FIPA Agent Architecture," Finland, 2003.

[24] E. DENIZ, "VideoOverDDS", MilSoft, Turkey, Brussels, Belgium, 2007.

[25] SimD, simd-cxx, https://code.google.com/p/simd-cxx/, 2011.

Multi-dimensional Interval Test

Qing Zhang
Dalian Shipping College, Dalian, China

Abstract - *In this paper, we propose a sophisticated technique of data dependence analysis for distributed memory parallel environments that is used for converting sequential code into a parallel form targeted for a particular architecture. Two-dimensional arrays with subscripts formed by induction variable in real programs appear quite frequently [10]. We test if there are integer-valued solutions for two-dimensional arrays with subscripts formed by induction variable. It is demonstrated that without direction vectors there are integer-valued solutions for two-dimensional arrays with subscripts formed by induction variable. Furthermore, it is also shown that under a specific direction vector* $\vec{\theta} = (=_1, \cdots, =_d)$ *there are integer-valued solutions for two-dimensional arrays with subscripts formed by induction variable. Finally, we point out that for other direction vectors there are no integer-valued solutions for two-dimensional arrays with subscripts formed by induction variable.*

Keywords: Parallelizing compilers, Data dependence analysis, Loop parallelization, I Test, Parallel code debugging.

1 Introduction

In automatic parallelization and parallelizing compilers, achieving a good data dependence analysis is a critical issue in order to reduce the communication overhead and to exploit parallelism of applications as much as possible. It is essential to develop a new analysis technique for data dependence. This motivates us that we are not only to develop traditional dependence tests for parallelizing transformation but also to develop new techniques for multi-dimensional induction variables that frequently occur in nested loops.

In this paper, we propose a sophisticated technique of data dependence analysis for distributed memory parallel environments that is used for converting sequential code into a parallel form. Our approach is to test dependence if there are *integer-valued* solutions for two-dimensional arrays with subscripts formed by induction variable. Without direction vectors, there are integer-valued solutions for two-dimensional arrays with subscripts formed by induction variable. Furthermore, it is also shown that under a specific direction vector $\vec{\theta} = (=_1, \cdots, =_d)$ there are integer-valued solutions for two-dimensional arrays with subscripts formed by induction variable.

There are several well-known data dependence analysis algorithms applicable for *one-dimensional* arrays under *constant* bounds or *variable* bounds: the GCD test [6, 9, 12, 13], Banerjee's method [6, 12, 13], the I test and the direction vector I test [2, 16, 24, 25, 26], the extension of the I test [14], the extension of the direction vector I test [20] and the interval reduction test [15]. There are also several well-known data dependence analysis algorithms applicable for *multi-dimensional coupled* arrays under *constant* bounds or *variable* bounds: the generalized GCD test [6, 12, 13], the Lambda test [3], the multi-dimensional I test [28], the multi-dimensional direction vector I test, the Power test [7] and the Omega test [4]. Also, there are several well-known data dependence analysis algorithms applicable for arrays with linear subscripts with symbolic coefficients or with non-linear subscripts under *symbolic* bounds: the infinity Banerjee test [11, 12], the Range test [17], the infinity Lambda test [23] and the access range test [18, 19].

Shen et al. pointed out that two-dimensional arrays with subscripts formed by induction variable occur quite frequently in real programs [10]. A *d*-nested loop accessing

a two-dimensional array with subscripts formed by induction variable is shown in Figure 1.

An induction variable is a scalar integer variable, which is used in a loop to simulate do-variables: it is incremented or decremented by a constant in each iteration. Every induction variable can be replaced by a linear function of the loop's index-variables. This transformation is called induction variable substitution. Since the variable *K* in Figure 1 is an induction variable, it can be replaced by

$$z\times((I_1-1)\times(\prod_{P_1=2}^{d}U_{P_1})+(I_2-1)\times(\prod_{P_2=3}^{d}U_{P_2})+\cdots+I_d),$$

where *d* is the number of common loops and *z* is one integer variable in Figure 1. Therefore, the code in Figure 1 is transformed into the code in Figure 2 after finishing the processing of induction variable substitution for the variable *K*.

We treat the loop iteration variables referenced in S_1 as being different variables from those referenced in S_2, subject to common loop bound, because dependence between S_1 and S_2 in Figure 2 may arise in different iterations of the common loops. Therefore, when we analyze dependence arising from a statement pair nested in *d* common loops, the problem will involve *n* unique variables (where *n* = 2*d*). Furthermore, variables X_{2k-1} and X_{2k} $(1\le k\le d)$ are different instances of the same loop iteration variable, and $L_{2k-1}=L_{2k}$, $U_{2k-1}=U_{2k}$, where $L_{2k-1}, L_{2k}, U_{2k-1}$ and U_{2k} are lower bounds and upper bounds for X_{2k-1} and X_{2k}, respectively.

```
K = 0
FOR  I_1 = 1 TO U_1
        ⋮
  FOR  I_d = 1 TO U_d
          K = K + z
S_1 :        A(K + C, K + C) = ⋯
S_2 :        ⋯ = A(K + C, K + C)⋯
             ⋮
  ENDFOR
             ⋮
ENDFOR
```

Figure 1: Example of a nested loop with induction variable.

The problem of determining if there is dependence for the array *A* between S_1 and S_2 in Figure 2 can be reduced to the problem of checking whether a system of two linear equations with *n* unknown variables has an integer solution simultaneously, which satisfies the constraints for each variable in the system. Assume that two linear equations in a system are written as

$$z\times((X_1-X_2)\times(U_2\times\cdots\times U_d)+(X_3-X_4)\times(U_3\times\cdots\times U_d)+\cdots+(X_{2j-1}-X_{2j})\times(U_{j+1}\times\cdots\times U_d)+\cdots+(X_{2d-3}-X_{2d-2})\times U_d+(X_{2d-1}-X_{2d}))=0 \quad (1\text{–}0)$$

$$z\times((X_1-X_2)\times(U_2\times\cdots\times U_d)+(X_3-X_4)\times(U_3\times\cdots\times U_d)+\cdots+(X_{2j-1}-X_{2j})\times(U_{j+1}\times\cdots\times U_d)+\cdots+(X_{2d-3}-X_{2d-2})\times U_d+(X_{2d-1}-X_{2d}))=0,$$

where each U_k $(1\le k\le d)$ is an integer variable and is one upper bound for the k^{th} loop nest. Because *z* is the greatest common divisor for all of the coefficients in the left-hand side of (1–0), all of the coefficients in (1–0) are divided by *z* and (1–0) is rewritten as

$$(X_1-X_2)\times(U_2\times\cdots\times U_d)+(X_3-X_4)\times(U_3\times\cdots\times U_d)+\cdots+(X_{2j-1}-X_{2j})\times(U_{j+1}\times\cdots\times U_d)+\cdots+(X_{2d-3}-X_{2d-2})\times U_d+(X_{2d-1}-X_{2d})=0 \quad (1\text{–}1)$$

$$(X_1-X_2)\times(U_2\times\cdots\times U_d)+(X_3-X_4)\times(U_3\times\cdots\times U_d)+\cdots+(X_{2j-1}-X_{2j})\times(U_{j+1}\times\cdots\times U_d)+\cdots+(X_{2d-3}-X_{2d-2})\times U_d+(X_{2d-1}-X_{2d})=0.$$

It is postulated that the constraints to each variable in (1–1) are represented as

$$1 \le X_{2k-1} \text{ and } X_{2k} \le U_k,$$

(1–2)

where $1 \le k \le d$.

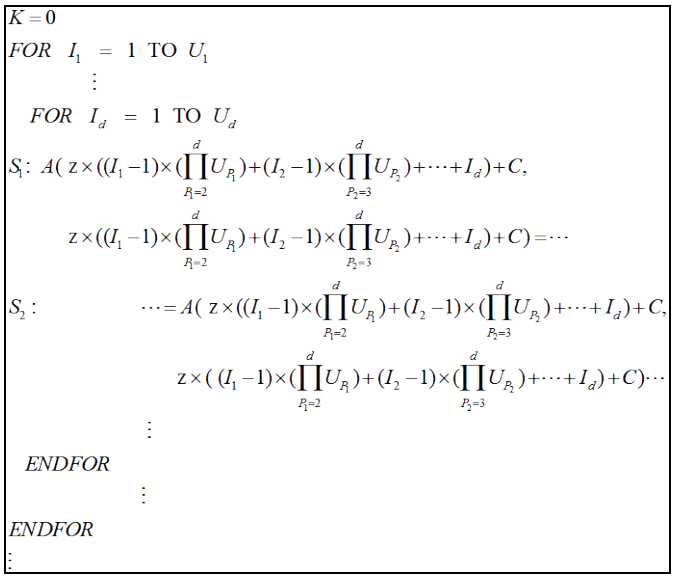


Figure 2: The result of example nested loop after finishing the processing of induction variable substitution.

2 Determining Integer-valued Solutions for Two-dimensional Arrays with Symbolic Coefficients Under Symbolic Bounds

Given a data dependence problem of two-dimensional arrays with *symbolic* coefficients under symbolic bounds, we want to find that (1) under what conditions there are *integer-valued* solutions and (2) under what conditions there is *no integer-valued* solution. In this section, we first discuss the case without direction vectors.

2.1 Determine Integer-valued Solutions for Two-dimensional Arrays with Symbolic Coefficients Under Symbolic Bounds Without Direction Vector

A system of linear equations (1–1) with *symbolic* coefficients is deduced from determining whether there exists dependence for the array *A* between S_1 and S_2 in Figure 2. A system of linear inequalities (1–2) is derived from lower and upper bounds of loop index variables in Figure 2. If there is no *integer-valued* solution for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2), then there is no dependence. Otherwise, there is dependence for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2). Lemma 2–1 is presented to demonstrate that there are integer-valued solutions for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2).

Lemma 2–1: There are *integer-valued* solutions for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2).

Proof: Omitted due to the limitation of space.

We now use the following example to explain how Lemma 2–1 is applied. Consider the do-loop in Figure 3. The front-end of a parallelizing compiler can recognize that the variable *K* is one induction variable and the subscript of the array *A* is formed by the induction variable *K*. The equations of the data dependence for the subscript of the array *A* are described below.

$$(X_1 - X_2) \times N + (X_3 - X_4) = 0$$
$$(X_1 - X_2) \times N + (X_3 - X_4) = 0.$$

(Ex1-1)

The I test in [2] is first used to separately test each linear equation in (Ex1–1). If one of the equations is indicated to be no *integer-valued* solution, then there is no integer-valued solution for all of the equations. Otherwise, the multi-dimensional I test in [28] is next applied to simultaneously check linear equations in (Ex1–1). From the I test, the first linear equation in (Ex1–1) with *symbolic* coefficients can be rewritten as the following interval equation:

$$(X_1 - X_2) \times N + (X_3 - X_4) = [0,\ 0].$$

(Ex1–2)

According to the I test, the term $-X_4$ in the interval equation (Ex1–2) is moved to the right-hand side to gain the new interval equation:

$$(X_1 - X_2) \times N + X_3 = [1,\ N].$$

(Ex1-3)

From the I test, the term X_3 in the interval equation (Ex1–3) is moved to the right-hand side to gain the new interval equation:

$$(X_1 - X_2) \times N = [1 - N,\ N - 1].$$

(Ex1-4)

Because *N* is the greatest common divisor for all of the coefficients in the left-hand side of the interval equation (Ex1–4), all of the coefficients in the interval equation (Ex1–4) are divided by *N* and the new interval equation is obtained:

$$(X_1 - X_2) = [\left\lceil \frac{1-N}{N} \right\rceil, \left\lfloor \frac{N-1}{N} \right\rfloor] = [0, 0].$$

(Ex1–5)

Repeat the processing of the steps above until the term in left-hand side of the interval equation is reduced to zero item. Therefore, we will obtain the new interval equation $0 = [1 - M, M - 1]$. Because $M \geq 1$, thus $1 - M \leq 0 \leq M - 1$. Therefore, there are *integer-valued* solutions for the first linear equation in (1–1) with *symbolic* coefficients. Similar processing is used to test the second linear equation in (Ex1–1). Conclusively, there are *integer-valued* solutions for the second linear equation. Assume that $\Omega_{1,j}$ is denoted as the coefficient of the j^{th} variable in the first linear equation and $\Omega_{2,j}$ the coefficient of the j^{th} variable in the second linear equation. A set of canonical solutions produced by the multi-dimensional I test is denoted as: $\{(\Omega_{2,j}, -\Omega_{1,j})\}$.

According to the multi-dimensional I test, every canonical solution yields one new interval equation. Because the sum of the coefficient of every variable for each new interval equation is 0, each interval equation is reduced to 0 = [0, 0]. Because $0 \leq 0 \leq 0$ is always true, we can derive that there are *integer-valued* solutions for linear equations in (Ex1–1) with *symbolic* coefficients.

2.2 Determine Integer-valued Solutions for Two-dimensional Arrays with Symbolic Coefficients Under Symbolic Bounds with Direction Vector

From Lemma 2–1, it is very clear that there are integer-valued solutions for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2). Next, under a specific given *direction vector* and the limits of (1–2), whether there are *integer-valued* solutions for linear equations in (1–1) with *symbolic* coefficients will be discussed. The following lemmas are proposed to show that (1) under what kind of a specific direction vector there are *no integer-valued* solutions and (2) under what kind of a specific direction vector there are *integer-valued* solutions.

Lemma 2–2: There are *no integer-valued* solutions for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2) and a specific direction vector $(<, *, \cdots, *)_d$, where *d* is the number of common loops and "*" is any one of $\{<, =, >\}$.

Proof: Omitted due to the limitation of space.

Lemma 2–3: There are *no integer-valued* solutions for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2) and a specific direction vector $(>, *, \cdots, *)_d$, where *d* is the number of common loops and "*" is any one of $\{<, =, >\}$.

Proof: Similar to Lemma 2–2. ■

Lemma 2–4: There are *no integer-valued* solutions for linear equations in (1–1) with *symbolic* coefficients under the limits of (1–2) and a specific direction vector $(=, \theta_2, \cdots, \theta_{d-1}, \theta_d)_d$, where *d* is the number of common loops and θ_k is any element of {<, =, >} for $2 \leq k \leq d - 1$ and θ_d is any element of {<, >}.

Proof: Similar to Lemma 2–2. ■

Lemma 2–5: There are *integer-valued* solutions for linear equations in (1–1) with *symbolic* coefficients under the

limits of (1–2) and a specific direction vector $(=,=,\cdots,=)_d$, where *d* is the number of common loops.

Proof: Similar to Lemma 2**–**2. ■

```
K=0
DO J = 1, M
   DO I =1, N
      K = K + 1
   S:  A(K+1, K+1) = B(K, K)
   ENDDO
ENDDO
```

Figure 3: A Fortran do-loop.

We now explain how Lemma 2–2 to Lemma 2–5 is applied to the do-loop in Figure 3. Consider the do-loop in Figure 3. The front-end of a parallelizing compiler can recognize that the variable *K* is one induction variable and the subscript of the array *A* is formed by the induction variable *K*. The equations of the data dependence for the subscript of the array *A* under any given direction vector (<, *) are described below.

$$(X_1 - X_2)\times N + (X_3 - X_4) = 0$$
$$(X_1 - X_2)\times N + (X_3 - X_4) = 0.$$

(Ex2-1)

From Lemma 2–2, suppose that $\Omega_{1,j}$ is denoted as the sum of the coefficients for the first and second variables in the first linear equation and $\Omega_{2,j}$ the sum of the coefficients to the first and second variables in the second linear equation. Because $\Omega_{1,j}$ and $\Omega_{2,j}$ are both 0, a set of canonical solutions produced by the multi-dimensional direction vector I test in [31] is denoted as: $\{(1,1)\}$. According to the multi-dimensional direction vector I test, the canonical solution, (1, 1) yields one new interval equation as follows:

$$2\times((X_1 - X_2)\times N + (X_3 - X_4)) = [0, 0].$$

(Ex2–2)

Because 2 is the greatest common divisor for all of the coefficients in the left-hand side of the interval equation (Ex2–2), all of the coefficients in the interval equation (Ex2–2) are divided by 2 and the new interval equation is obtained:

$$(X_1 - X_2)\times N + (X_3 - X_4) = [0, 0].$$

(Ex2–3)

According to the I test, the term $-X_4$ in the interval equation (Ex2–3) is moved to the right-hand side to gain the new interval equation:

$$(X_1 - X_2)\times N + X_3 = [1, N].$$

(Ex2–4)

From the I test, the term X_3 in the interval equation (Ex2–4) is moved to the right-hand side to gain the new interval equation:

$$(X_1 - X_2)\times N = [1 - N, N - 1].$$

(Ex2–5)

Because *N* is the greatest common divisor for all of the coefficients in the left-hand side of the interval equation (Ex2–5), all of the coefficients in the interval equation (Ex2–5) are divided by *N* and the new interval equation is obtained:

$$(X_1 - X_2) = [\left\lceil \frac{1-N}{N} \right\rceil, \left\lfloor \frac{N-1}{N} \right\rfloor] = [0, 0].$$

(Ex2–6)

Because the direction vector is "<" for the terms X_1 and X_2, according to the direction vector I test [8], the two terms X_1 and X_2 are moved to the right-hand side. Therefore, we obtain the new interval equation $0 = [1, M-1]$. Because $1 \le 0 \le M-1$ is false, thus there are *no integer-valued* solutions for linear equations in (Ex2–1) with *symbolic* coefficients under a specific direction vector (<, *). Similarly, for dealing with other

specific direction vector, Lemmas 2–3 to 2–5 can be applied for dealing with other specific direction vector. Hence, from Lemma 3–5, it is obtained that there are integer-valued solutions for the array *A* under a specific direction vector $(=,=)$. From Lemma 2–2 to Lemma 2–4, consequently, there is no integer-valued solution for the array *A* under other specific direction vectors. The front end of a parallelizing compiler derives that there only exists loop-independence output dependence for the do-loop and the do-loop can be executed in parallel mode.

2.3 Extending Symbolic Subscript Formed by Induction Variable

We can easily extend the expression "*K*+*C*" for the array *A* in Figure 1 to "*a***K*+*C*", where *a* is an integer variable and is not equal to zero. Since the variable *K* in Figure 1 is one induction variable, it can be replaced by

$$z\times((I_1-1)\times(\prod_{P_1=2}^{d}U_{P_1})+(I_2-1)\times(\prod_{P_2=3}^{d}U_{P_2})+\cdots+I_d).$$

Therefore, the code for the extended expression of the array *A* in Figure 1 is transformed into the code in Figure 4 after finishing the processing of induction variable substitution for the variable *K*. The problem of determining whether there exists one dependence for the array *A* between S_1 and S_2 in Figure 4 is actually equal to that of checking whether there are integer-valued solutions for a system of *new* linear equations in (2–11) under the constraints of (1–2). It is assumed that the system of new linear equations in (2–11) is written as

$$\begin{aligned}&a\times z\times((X_1-X_2)\times(U_2\times\cdots\times U_d)+(X_3-X_4)\times(U_3\times\cdots\times U_d)+\cdots\\&+(X_{2j-1}-X_{2j})\times(U_{j+1}\times\cdots\times U_d)+\cdots\\&+(X_{2d-3}-X_{2d-2})\times U_d+(X_{2d-1}-X_{2d}))=0 \text{ and}\end{aligned}$$

(2–11)

$$\begin{aligned}&a\times z\times((X_1-X_2)\times(U_2\times\cdots\times U_d)+(X_3-X_4)\times(U_3\times\cdots\times U_d)+\cdots\\&+(X_{2j-1}-X_{2j})\times(U_{j+1}\times\cdots\times U_d)+\cdots\\&+(X_{2d-3}-X_{2d-2})\times U_d+(X_{2d-1}-X_{2d}))=0,\end{aligned}$$

where each U_k is an integer variable and is one upper bound for the k^{th} loop for $1\le k\le d$. Because $a\times z$ is not equal to zero, all of the coefficients in (3–11) are divided by $a\times z$ and (3–11) is exactly equal to (1–1). Therefore, the following theorems can be applied to deal with whether there is dependence for the array *A* in Figure 4.

Theorem 2–1: There are *integer-valued* solutions for linear equations in (2–11) with *symbolic* coefficients under the limits of (1–2) and a specific direction vector $(=,=,\cdots,=)_d$, where *d* is the number of common loops.

Proof: Similar to Lemma 2–2. ■

Theorem 3–2: There are no *integer-valued* solutions for linear equations in (2–11) with *symbolic* coefficients under the limits of (1–2) and other given direction vectors.

Proof: Similar to Lemma 2–2. ■

3 Experimental Results

We have tested our method and have performed experiments by hand on the codes abstracted from three numerical packages: Parallel loop, Vector loop and Perfect Benchmark [5, 8, 25]. 10 pairs of two-dimensional array references with different direction vectors are observed to have subscripts formed *induction variable*. The proposed method is only applied to test those two-dimensional arrays with subscripts formed *induction variable*. It is found in the experiment that there is no case to satisfy the condition of the proposed method for Vector loop, TRFD and OCSI. It is very clear from the result shown in Table 1 that the proposed method could properly solve whether there are *integer-valued* solutions for two-dimensional arrays with subscripts formed induction variable.

We also implemented the Omega test and the Power test based on [4, 7] to compare their effects with that of the proposed method. The Omega test and Power test are applied to solve the same 10 pairs of two-dimensional arrays with symbolic coefficients. Clearly, from the result shown in Table 2, the Omega test and the Power test could not be applied for determining whether there are *integer-valued* solutions for two-dimensional arrays with *symbolic* coefficients. Therefore, the results shown in Table 1 and Table 2 indicate that the precision of the proposed method

is slightly superior to that of the Omega test and the Power test.

Table 1. The result of solving whether there are integer-valued solutions for two-dimensional arrays with subscripts formed induction variable.

Benchmark	The number of arrays tested	The number of integer-valued solution	The number of no integer-valued solutions
Parallel loop	10	2	8
Vector loop	0	0	0
TRFD	0	0	0
OCSI	0	0	0

$$
\begin{aligned}
&K = 0\\
&FOR\ \ I_1 = 1\ \text{TO}\ U_1\\
&\quad\ \vdots\\
&\quad FOR\ \ I_d = 1\ \text{TO}\ U_d\\
&S_1:\ A(a\times z\times((I_1-1)\times(\prod_{P_1=2}^{d}U_{P_1})+(I_2-1)\times(\prod_{P_2=3}^{d}U_{P_2})+\cdots+I_d)+C,\\
&\qquad a\times z\times((I_1-1)\times(\prod_{P_1=2}^{d}U_{P_1})+(I_2-1)\times(\prod_{P_2=3}^{d}U_{P_2})+\cdots+I_d)+C)=\cdots\\
&S_2:\qquad \cdots=A(a\times z\times((I_1-1)\times(\prod_{P_1=2}^{d}U_{P_1})+(I_2-1)\times(\prod_{P_2=3}^{d}U_{P_2})+\cdots+I_d)+C,\\
&\qquad\qquad a\times z\times((I_1-1)\times(\prod_{P_1=2}^{d}U_{P_1})+(I_2-1)\times(\prod_{P_2=3}^{d}U_{P_2})+\cdots+I_d)+C)\cdots\\
&\qquad\ \vdots\\
&\ ENDFOR\\
&\qquad\ \vdots\\
&ENDFOR\\
&\vdots
\end{aligned}
$$

Figure 4: The result for the extended expression of the array *A* in Figure 1 after induction variable substitution.

Table 2. The result of the Omega and Power tests for solving whether there are integer-valued solutions for 10 pairs of two-dimensional arrays with *symbolic* coefficients.

Dependence Method	The number of arrays tested	The number of *integer-valued* solution	The number of *no integer-valued* solutions
The Omega Test	10	–	–
The Power Test	10	–	–

("–" represents that the Omega and Power tests cannot deal with them.)

The study in [4] stated that (1) the cost of scanning array subscripts and loop bounds to build a dependence problem is typically 2 to 4 times of the copying cost (the cost of building a system of dependence equations) for the problem, and (2) the dependence analysis cost for more than half of *simple* arrays tested is typically 2 to 4 times of the copying cost, but the dependence analysis cost for other simple arrays and all of the *regular*, *convex* and *complex* arrays tested is more than 4 times of the copying cost. Based on such results we can figure out that, for simple arrays, the analysis cost of data dependence for a parallelizing/vectorizing compiler occupies generally about 29% to 57% of total compiling time. But, for complex arrays, the analysis cost of dependence testing takes more than 57% of total compiling time. Therefore, enhancing on dependence testing performance may result in significant improvement on compiling performance of a parallelizing/vectorizing compiler.

4 Conclusions

The front-end of a parallelizing compiler can easily recognize induction variable that forms subscripts of two-dimensional arrays. Induction variable can be replaced by a linear function in the loop's index-variable after the front-end of a parallelizing compiler finishes the processing of induction variable substitution. The proposed method can offer data dependence analysis for arrays with references formed by induction variable. Depending on the application domains, we suggest to applying the proposed method together with the front-end

of a parallelizing compiler to provide data dependence analysis for arrays with references formed by induction variable.

5 References

[1] B. J. Smith et al., *Matrix Eigensystem Routines-Eispack Guide*, Heidelberg: Springer, 1976.

[2] Xiangyun Kong, David Klappholz and Kleanthis Psarris, "The I test," IEEE Transaction on Parallel and Distributed Systems," Vol. 2, No. 3 (July 1991), pp. 342–359.

[3] Z. Li, P.-C. Yew and C.-Q. Zhu. "An efficient data dependence analysis for parallelizing compilers," IEEE Transaction on Parallel and Distributed Systems, Vol. 1, No. 1 (January 1990), pp. 26–34.

[4] W. Pugh, "A practical algorithm for exact array dependence analysis," Communication of the ACM, 35(8) (August 1992), pp. 102–114.

[5] Dongarra J., M. Furtney, S. Reinhardt and J. Russell, "Parallel Loops – A test suite for parallelizing compilers: Description and example results," Parallel Computing 17(1991) 1247–1255.

[6] Uptal Banerjee. *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers, Norwell, Massachusetts, 1988.

[7] M. Wolfe and C.W. Tseng, "The power test for data dependence," IEEE Transaction on Parallel and Distributed Systems, Vol. 3, No. 5 (September 1992), pp. 591–601.

[8] David Levine, David Callahan and Jack Dongarra, "A comparative study of automatic vectorizing compilers," Parallel Computing 17(1991) pp.1223-1244.

[9] Vaughan and William Jeffrey, *A residuals management model of the iron and steel industry: a linear programming approach*, Mich.: Univ. Microfilms International, Ann Arbor, 1986.

[10] Z. Shen, Z. Li and P.-C. Yew, "An empirical study of Fortran programs for parallelizing compilers," IEEE Transaction on Parallel and Distributed Systems, Vol. 1, No. 3 (July 1992), pp. 356–364.

[11] Paul M. Petersen, "Evaluation of programs and parallelizing compilers using dynamic analysis techniques," PhD thesis, University of Illinois at Urbana-Champaign, January 1993.

[12] Uptal Banerjee, *Dependence Analysis*, Kluwer Academic Publishers, Norwell, Massachusetts, 1997.

[13] Uptal Banerjee, *Loop Transformations for Restructuring Compilers*: *The Foundations*, Kluwer Academic Publishers, 1993.

[14] R. Triolet, F. Irigoin and P. Feautrier, "Direct parallelization of call statements," in Proc. SIGPLAN Symp. Compiler Construction, Palo Alto, CA, 1986, 176–185.

[15] T.C. Huang and C.M. Yang, "Data Dependence Analysis for Array References," The Journal of System and Software, 52(2000) 55-65.

[16] Kleanthis Psarris, Xiangyun Kong, David Klappholz, "The direction vector i test," IEEE Transaction on Parallel and Distributed Systems, Vol. 4, No. 11 (1993) 1280–1290.

[17] W. Blume and R. Eigenmann, "Nonlinear and symbolic data dependence testing," IEEE Transaction on Parallel and Distributed Systems, Vol. 9, No. 12 (1998) 1180–1194.

[18] Yunheung Paek, "Compiling for Distributed Memory Multiprocessors Based on Access Region Analysis," PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[19] Jay Philip Hoeflinger, "Interprocedural Parallelization Using Memory Classification Analysis," PhD thesis, University of Illinois at Urbana-Champaign, 1998.

[20] Chih-Ping Chu and Weng-Long Chang, "The Extension of Direction Vector I Test," Proceedings of the 10th IASTED International Conference Parallel and Distributed Computing and Systems, Oct. 1998, Nevada, USA, pp. 484-489.

[21] Weng-Long Chang and Chih-Ping Chu, "The Infinity Lambda Test: A Multi-dimensional Version of Banerjee's Infinity Test," Parallel Computing, Vol. 26, Number 10, Aug. 2000, pp. 1275-1295.

[22] D.E. Knuth, *The Art of Computer Programming*, Vol. 2, Seminumerical Algorithms, second edtion Reading, MA: Addison-Wesley, 1981.

[23] R. Eigenmann, J. Hoeflinger, and D. Padua, "On the Automatic Parallelization of the Perfect Benchmarks," IEEE Transactions on Parallel and Distributed Systems. Vol. 9, No. 1, pp. 5-23. Jan. 1998.

[24] Kleanthis Psarris, David Klappholz, and Xiangyun Kong, "On the Accuracy of the Banerjee Test," Journal of Parallel and Distributed Computing, 12(2): 152-158, June 1991.

[25] Kleanthis Psarris and Konstantions Kyriakopoulos, "Data Dependence Testing in Practice," Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques.

[26] David Niedzielski and Kleanthis Psarris, "An Analytical Comparison of the I-Test and Omega Test," LCPC'99: Twelfth International Workshop on Languages and Compilers for Parallel Computing.

[27] R. Eigenmann, J. Hoeflinger, and D. Padua, "On the automatic parallelization of the perfect benchmarks," IEEE Transactions on Parallel and Distributed Systems. Vol. 9, No. 1, pp. 5-23. Jan. 1998.

[28] W.-L. Chang, C.-P. Chu and J.-H. Wu, "A multi-dimensional direction vector I test," The Journal of System and Software, Feb. 2001.

Developing NAND-memory SSD based Hybrid Filesystem

Jaechun No[1]
[1]College of Electronics and Information Engineering, Sejong University, Seoul, Korea

Abstract— *As the technology of NAND flash memory rapidly grows, SSD is becoming an excellent alternative for storage solutions, because of its high random I/O throughput and low power consumption. These SSD potentials have drawn great attention from IT enterprises that seek for better I/O performance. However, high SSD cost per capacity makes it less desirable to construct a large-scale storage subsystem solely composed of SSD devices. An alternative is to build a hybrid storage subsystem where both HDD and SSD devices are incorporated in an economic manner, while employing the strengths of both devices. This paper presents a hybrid file system, called HybridFS, that attempts to utilize the advantages of HDD and SSD devices, to provide a single, virtual address space by integrating both devices. HybridFS not only proposes an efficient implementation for the file management in the hybrid storage subsystem but also suggests an experimental framework for making use of the excellent features of existing file systems. Several performance evaluations were conducted to verify the effectiveness and suitability of HybridFS.*

Keywords: flash memory, hybrid system, SSD, data layout, extent

1. Introduction

The rapid growth of flash memory technology has led to the advent of SSD (Solid-State Device) in the storage platform. Due to the fact that SSD does not need the mechanical overhead, such as seek time, to locate the desire data, it has drawn great attention from IT markets that seek for improved I/O performance. These days SSD is becoming an excellent storage solution, to provide optimal I/O performance. For instance, database produces a large number of writes to store data for the purpose of update, rollback, and log. In this case, SSD offers a good opportunity to achieve desirable I/O performance.

Despite its performance potentials, the common usage of SSD is currently restricted to small-size memory devices, such as mobile equipments. The key obstacle to the widening SSD adoption to large-scale storage subsystems is its high cost per capacity ($3/GB for SSD, whereas $0.3/GB for HDD) [1]. Even though the cost of flash memory becomes decrease, the price of SSD is still much higher, compared to that of HDD. Such a high cost/capacity ratio makes it less desirable to build large-scale storage subsystems solely composed of SSD devices. An alternative storage solution is to build a hybrid storage subsystem where both SSD and HDD devices are combined in a cost-effective way, while making use of the strengths of both devices.

In this paper, we present a hybrid file system, called HybridFS, developed for exploiting the hybrid storage structure. HybridFS is capable of generating comparable performance to the file system installed on SSD devices, while offering much larger storage capacity at less cost. This is achieved by integrating vast, low-cost HDD storage space with a small portion of SSD space through several optimization schemes, to address the strengths and the shortcomings of both devices.

This paper is organized as follows: In Section 2, we discuss the design motivations and related studies. In Section 3, we analyze and compare the performance evaluation for HDD and SSD. In Section 4, we present the design of HybridFS and in Section 5, we conclude with a summary.

2. Related Studies

As pointed out by [2], [3], the major drawbacks of flash memory are write latency due to erasure per block and cleaning problems. Many flash file systems [4], [5], [6] take the out-of-place approach proposed by log-structured file system [7], to reduce the semiconductor overhead of flash memory.

The well-known characteristic of log-structured file system is its sequential, out-of-place update using logs. The logs are divided into segments, to maintain large disk free area and to avoid space fragmentation. The garbage collection is performed per segment, by copying live data out of a segment. This update approach has been adopted to most flash file systems, to minimize the block erasure overhead. JFFS and JFFS2 [6] maintain file metadata logs for the sequential writing. To mount file system, they require to scan all the logs in flash memory, which may take considerably long time to build. Also, both flash file systems were designed for embedded equipments with small-size NAND flash memory.

TFFS [5] is an optimized file system which suits for the small embedded systems with less than 4KB of RAM. TFFS is targeted for NOR devices and provides several useful functions, such as tailored API for the embedded device and concurrent transaction recoveries. ELF [4] is built for sensor nodes and keeps a log entry for each flash page. ELF tries to optimize the logging overhead by reducing the number of log entries. However, most of flash file systems have been developed for small-size flash memory and therefore is not appropriate for a large-scale data storage.

As HybridFS does with SSD devices, there are several interesting attempts to make use of storage class memories. LiFS [8] intends to store file system metadata to smaller, faster storage class memory, such as MRAM. Conquest [9] uses persistent RAM for storing small files and file system metadata. Only the remaining data of large files are stored in disk. MRAMFS [10] goes further by adding compression, to overcome the limited size of NVRAM. HeRMES [11] proposed to keep all file metadata in MRAM and to apply compression to minimize the required space for metadata. hFS [12] attempts to combine the strengths of both FFS and log-structured file system, by separating file data and file metadata into two partitions.

Besides, Agrawal et. al. [13] describes interesting SSD design issues using Samsung's NAND-flash. They also suggest several ways of obtaining improved I/O performance on SSD, by analyzing a variety of I/O traces extracted from real systems. However, there is little file system works for hybrid storage subsystems where HDD and SSD are incorporated to provide a large-scale, virtualized address space.

3. Performance Evaluation for HDD and SSD

To design a new filesystem, we were interested in which storage produces a good I/O performance on distributed-small files or sequential-big files. The first condition is the analysis of metadata access patterns and the second is the one of data access patterns.

To experiment these interesting storage issues, we tested HDD and SSD with three kinds of filesystems, using Postmark and IOzone [16]. The filesystems selected for this test are XFS, NILFS, and Ext2 [14], [15]. These three filesystems manage blocks using a quite different procedure from each other.

XFS is a journaling filesystem designed to support the fast crash recovery. To allow large I/O requests to a file, it allocates the file as contiguously as possible. This is because the size of a request to the underlying drives is limited by the range of contiguous blocks in the file being read or written. We choose XFS to test the performance of storages with sequential access patterns.

NILFS is a new log-structured filesystem for Linux. Instead of overwriting existing blocks, it continuously appends the consistent sets of modified or newly created blocks into segmented disk regions. It is noted that many random accesses are needed to read a file consisted of distributed blocks. We use NILFS to test the performance of storages with random access patterns.

Ext2 does not support any of schemes to sequentially allocate blocks. In this experiment, Ext2 filesystem is just standard for

comparing with other file systems.

The machine used for these performance measurements has a Intel Xeon 3GHz CPU, 3Gbyte of memory, 7200rpm IDE hard drives with 32-Mbytes disk cache and SATA-connected Mtron SSD. The Linux kernel version used for the measurement was 2.6.23. Both the disk block and page sizes are 4Kbytes.

3.1 Postmark Benchmark

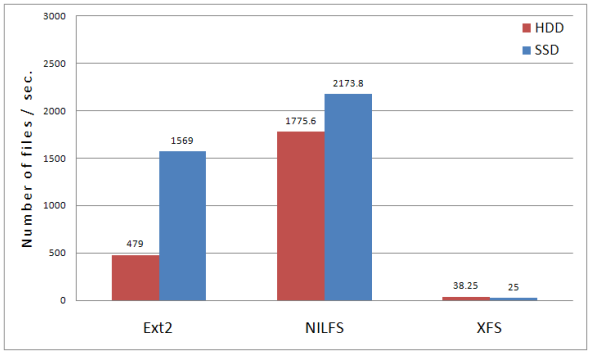


Fig. 1: I/O bandwidth for the file create operations.

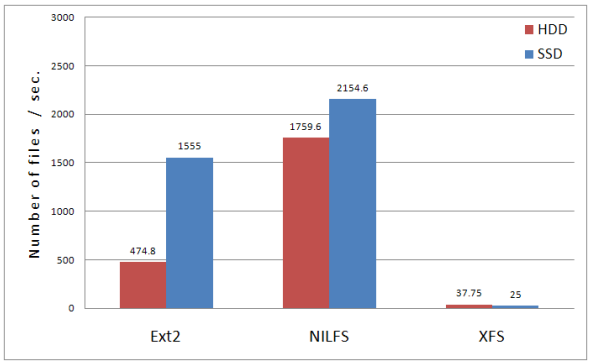


Fig. 2: I/O bandwidth for the file read operations.

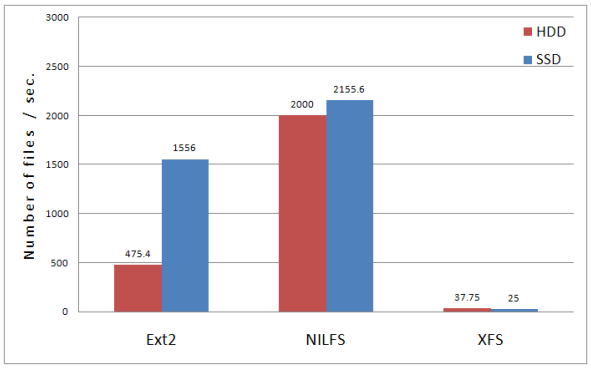


Fig. 3: I/O bandwidth for the file delete operations.

We used Postmark to evaluate the I/O performance of small-files. Postmark is a benchmark to measure the performance of the ephemeral small-files, such as electronic mail, net news and web-based commerce. It first creates a specified number of files. Then it performs a mixture of creation, deletion, read, and append operations on them. Finally, all files are deleted. We used the default setting of Post Mark v1.5 in which the block size to read and write is 512 bytes, the read/append and create/delete ratios are 5 and the file size range is 500 bytes $\sim$ 9.77 kilobytes. In this test, the number of files is set to 10000 and the number of transactions is set to 50000.

Figure 1 shows the performance for creating files. As can be seen in the Figure, Ext2 and NILFS produce better write performance with SSD than with HDD.

An Ext2 file consists of metadata and contents of it. The metadata of Ext2 filesystem consists of block bitmap, inode bitmap, super block, inode, and so on. Modifications to these metadata require to relocate a disk arm because Ext2 filesystem stores it in a fixed area for each metadata. It manages data blocks of a file by inode structures which have 12-direct block list and 3-indirect block list. Because of this design, the job access to data leads to the random access for the file size over 48Kbytes, in case of 4Kbytes of block size. These random characteristics in accessing an Ext2 file cause significant overheads in relocating the disk head to search metadata or data many times. However, with SSD, the operations being performed randomly to access a file no longer need mechanical actions. This results in much better performance in Ex2 filesystem installed on SSD than on HDD.

NILFS produces better performance than Ext2 filesystem on the whole. It is a log-structured filesystem which allocates blocks as appendix and stores inode and data as a group, thereby causing better performance than Ext2 filesystem.

XFS shows the worst I/O performance in the Figure 1, even with SSD. Presently, we do not have a clear idea why XFS shows such a worst performance with Postmark. Possibly, SSD may not be suitable for the sequential block allocation, as in XFS.

Figure 2 shows the read performance. As can be seen in the results of file create operations, SSD produces better performance than HDD in Ext2 and NILFS. As mentioned above, Ext2 needs to read data from disk using the inode structure, resulting in the disk seek time with HDD. This does not happen with SSD. In order to minimize the disk seek overhead, NILFS groups an inode and data for a file, thus resulting in much better read performance than in Ext2. This is also true with SSD.

Figure 3 shows the performance of file delete operations. To delete a file, Ext2 sets the associated data block and inode bitmaps of the file to "0". Because of this procedure, the experiment causes many random accesses. The performance degradation with HDD reflects these random accesses both in Ext2 and NILFS. However, it is noted that SSD does not suffer from these random accesses.

3.2 IOzone Benchmark

IOzone shows different performance patterns from Postmark because it generates files of sharply different sizes. Postmark generates tiny files, between 500 bytes and 1kbyte, but IOZone produces large files, between 4K and 2Gbytes. We tested SSDs using Write/Re-write, Read/Re-read and Random-write/Random-read of IOzone.

Write test measures the performance of writing a new file. When a new file is written, both the data and metadata must be stored in the disk. ***Re-write test*** measures the performance of writing a file that already exists. When a file is re-written, the I/O overhead to be required is clearly less than in the file write operation because the associated metadata already exists. ***Read test*** measures the performance of reading an existing file. ***Re-read test*** measures the performance of reading a file that was recently read. In general, the performance for re-read operations is higher than that for read operations because of caching data recently read. ***Random-read test*** and ***Random-write test*** measure the performance of reading a file or writing to a file with random access patterns. The performance of the random I/O operations can be affected by several factors, such as the size of file cache, number of disks and seek latencies.

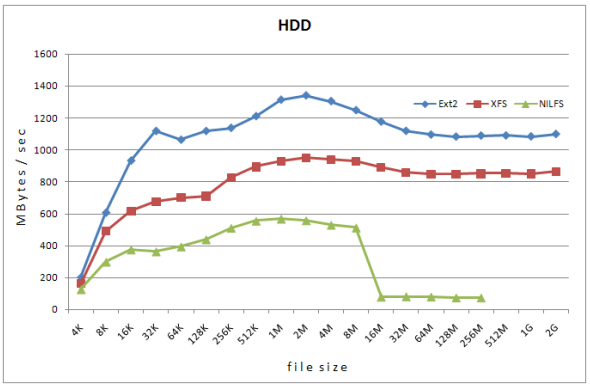


Fig. 4: Write performance on HDD

As can be seen in Figures 4 and 5, three file systems illustrate similar performance with HDD and SSD. Especially, NILFS shows almost same behavior for all the file sizes. Ext2 and XFS generate similar write performance on SSD and HDD until 64Mbytes of file size. When the file size increases above 128Mbytes, the write performance on SSD is sharply decreased. In these results, we can guess that SSD does not appropriate to write big files either with distributed blocks as in Ext2, or with sequential blocks as in XFS.

In a filesystem, many blocks are overwritten several times, for instance, an inode structure is always updated when a file

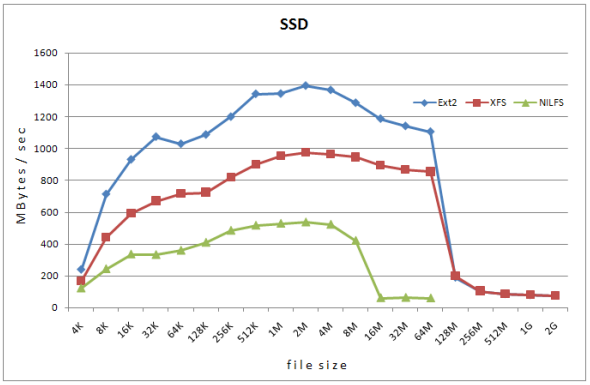


Fig. 5: Write performance on SSD

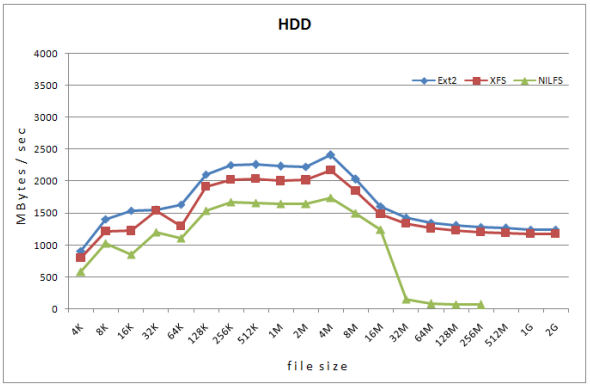


Fig. 8: Random-write performance on HDD

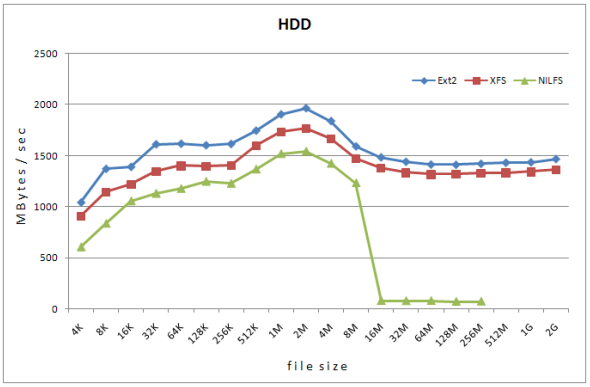


Fig. 6: Re-write performance on HDD

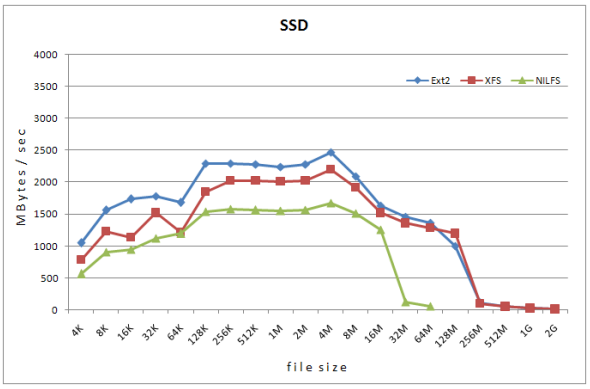


Fig. 9: Random-write performance on SSD

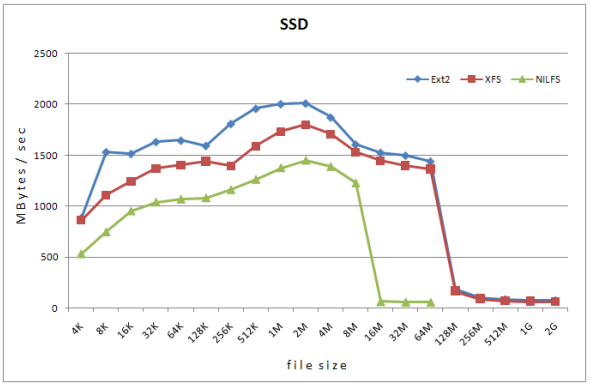


Fig. 7: Re-write performance on SSD

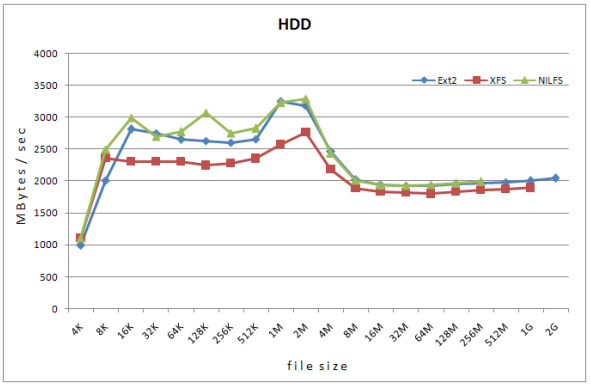


Fig. 10: Read performance on HDD

is accessed. Therefore, the re-write performance is important to design a filesystem. The Figures 6 and 7 show the re-write performance in HDD and SSD. The I/O behavior of the measurement is much similar with the write performance, but the re-write test generates better performance than the write test because the re-write operation does not need to write metadata. A similar performance decline with SSD happens in XFS and Ext2, too. SSD does not support the data modifications which are frequently requested in re-write operations. To modify data, SSD deletes the desired blocks at first and writes the modified data to them. This complicated operation is supposed to be the reason of decreasing I/O performance.

Figures 8 and 9 show the random-write performance. In these Figures, NILFS produces better performance with HDD than with SSD because, as mentioned above, SSD does not support data modification. Also, SSD uses a different block unit for the write and delete operations. These two features cause the less performance with SSD than with HDD. Furthermore, the random-write test includes the overhead to repeatedly delete and write to random locations within the file, resulting in the performance degradation in SSD.

Figures 10 and 11 show the read performance on HDD and SSD. In these Figures, the performance of NILFS shows the similar I/O behavior with Ext2, unlike in the write performance. NILFS is a log-structured filesystem in which the metadata and data are sequentially stored as a group in the same location. By this feature of NILFS, the read operation with HDD shows the similar result with SSD. The result for the sequential read access on HDD is also much similar with on SSD.

The re-read and random-read graphs show similar shape with the read operation. The re-read performance is better than the read performance, as in Figures 10 and 11. When compared the read performances with the re-read, SSD generates higher I/O bandwidth than HDD with the file size of less than 256Kbytes. The random-read performance on SSD shows better performance than HDD in general, especially with 4Kbytes and over the 8Mbytes. In this test, HDD needs seek operations to access random locations within a file, but SSD does not. This storage feature contributes to the difference of the random-read performance.

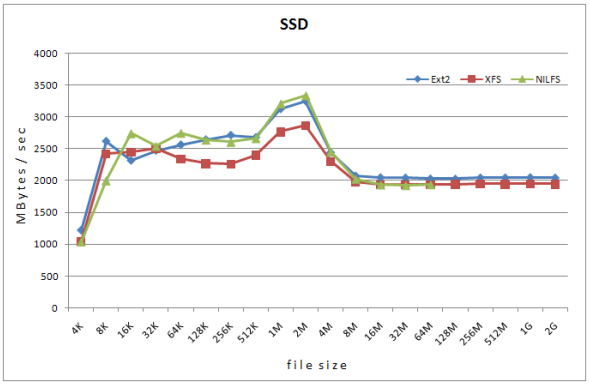


Fig. 11: Read performance on SSD

3.3 Performance Review

The test for small files - from 500 bytes to 9.77kbytes - performed in section 3.1 shows that SSD is appropriate to read, write and delete many small files with random accesses.

The other test for general files performed in section 3.2 shows that the performance with HDD is almost same with SSD for less than 64bytes of file size. With files of larger than 64bytes, SSD shows the sharp performance decline in Ext2 and XFS, except for random-read operations. It suggests that HDD generates good I/O performance with general files and is not related to the access pattern of filesystems. If a filesystem manages the data sequentially, HDD could handle a file over 4bytes more efficiently than SSD.

4. Design of a Hybrid Filesystem

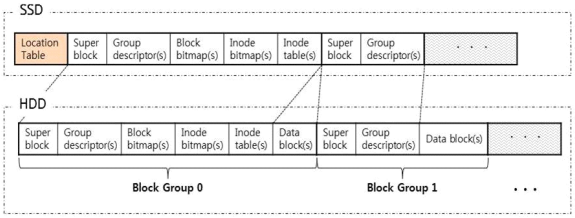


Fig. 12: A prototype of HybridFS

The key idea of HybridFS is the separation of data and metadata by storing them in a different type of storage, according to their characteristics.

As mentioned before, SSD is good for storing distributed, small files, thus HybridFS retains metadata in SSD. On the other hand, HDD is good for sequential, big files, so HybridFS is designed to store data contiguously in HDD. Figure 12 shows the disk layout of HybridFS

The HybridFS creation is started by duplicating the metadata stored in HDD into SSD, such as superblock, group descriptor table, block and inode bitmap and inode table. We added the new location field in HDD super block since both SSD and HDD partitions should know the block locations of each other. After that, using the information stored in the new location filed, HybridFS is mounted on SSD partition.

The task for using HybridFS is consisted of two steps. The first step is to format two devices for storing system metadata and real data. In the second step, we configure several metadata including superblock and also add the extra

field in the superblock to make both HDD and SSD partitions recognize the data locations of each other.

To duplicate the file system metadata to SSD partition, we first allocate SSD data block where those metadata are stored and then create the location table where each position denotes the SSD metadata location.

The HybridFS mount is performed on SSD partition by calling sys_mount(). In the function, HybridFS accesses file system metadata from SSD partition by referring to the location table, while recording the necessary system metadata to HDD partition. Figure 13 shows the location table for storing file system metadata to SSD partition. The rest steps for HybridFS are now being implemented.

5. Conclusions

As the technology of flash memory rapidly grows, SSD has drawn a great attention from IT enterprises as an attractive storage solution for fast I/O processing needs. SSD not only generates high I/O performance because of the absence of mechanical moving overhead but also provides significant power savings. However, despite its promising potentials, most SSD usages in real products have been limited to small-size memory devices, such as mobile equipments, because of its high cost per capacity. In this paper, we proposed a way of integrating SSD devices with HDD devices in a cost-effective manner, to build a large-scale, virtual address space. We tested HDD and SSD to take a good sight for developing a hybrid filesystem. The performance results obtained by using IOzone and Postmark show different shapes in Ext2, XFS and NILFS. In the evaluation, we showed that SSD is good for storing distributed, small files. On the other hand, HDD is good for keeping sequential, big files. The prototype of HybridFS is designed to take this feature by storing metadata in SSD and real data in HDD. We will further upgrade HybridFS, while combining the advantages of both HDD and SSD to generate high I/O performance.

BG	Metadata	HDD		SSD
0	Super Block	0		1
	Group Descriptor	1		2
	Block Bitmap	18		3
	Inode Bitmap	34		6
	Inode Table	50		9
1	Super Block	32768	→	2145
	Group Descriptor	32769	→	2146
	Block Bitmap	19	→	4
	Inode Bitmap	35		7
	Inode Table	762		721
2	Super Block	0		0
	Group Descriptor	0		0
...	...	...		...
...	...	...		...

Fig. 13: HybridFS location table structure

6. Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF- 2014R1A2A2A01002614).

References

[1] M. Saxena and M. Swift, "FlashVM: Virtual Memory Management on Flash," *2010 USENIX Annual Technical Conference*, Boston, MA, 2010.

[2] W. K. Josephson, L. A. Bongo, and D. Flynn, "DFS: A File System for Virtualized Flash Storage," *In Proceedings of the 8th USENIX Conference on File and Storage Technologies*, San Jose, USA, 2010.

[3] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD Lifetimes with Disk-Based Write Caches," *In Proceedings of the 8th USENIX Conference on File and Storage Technologies*, San Jose, USA, 2010.

[4] H. Dai, M. Neufeld, and R. Han, "ELF: An Efficient Log-Structured Flash File System for Micro Sensor Nodes," *SenSys'04*, Baltimore, USA, 2004.

[5] E. Gal and S. Toledo, "A Transactional Flash File System for Microcontrollers," *In Proceedings of 2005 USENIX Annual Technical Conference*, Anaheim, CA, 2005.

[6] D. Woodhouse, "JFFS: The Journaling Flash File System," *In Ottawa Linux Symposium*, 2001.

[7] M. Rosenblum and J. Ousterhout, "The Design and Implementation of a Log Structured File System," *In Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991, pp.1-15.

[8] S. Ames, N. Bobb, K. Greenan, O. Hofmann, M. W. Storer, C. Maltzahn, E. L. Miller, and S. A. Brandt, "LiFS: An Attribute-Rich File System for Storage Class Memories," *In Proceedings of the 23rd IEEE/14th NASA Goddard Conference on Mass Storage Systems and Technologies*, College Park, USA, 2006.

[9] A. Wang, G. Kuenning, P. Reiher, and G. Popek, "Conquest: Better Performance Through a Disk/Persistent-RAM Hybrid File System," *In Proceedings of the 2002 USENIX Annual Technical Conference*, Monterey, CA, 2002.

[10] N. K. Edel, D. Tuteja, E. L. Miller, and S. A. Brandt, "MRAMFS: A compressing file system for non-volatile RAM," *In Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Volendam, Netherlands, 2004.

[11] E. L. Miller, S. A. Brandt, and D. D. E. Long, "HeRMES: High-performance reliable MRAM-enabled storage," *In Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, Schloss, Germany, 2001, pp.83-87.

[12] Z. Zhang and K. Ghose, "hFS: A Hybrid File System Prototype for Improving Small File and Metadata Performance," *EuroSys'07*, Lisbon, Portugal, 2007.

[13] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," *2008 USENIX Annual Technical Conference*, 2008.

[14] R. Card, T. Ts'o, and S. Tweedie, "Design and Implementation of the Second Extended Filesystem," *In Proceedings of the First Dutch International Symposium on Linux*, 1995.

[15] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Tech, "Scalability in the XFS File System," *In Proceedings of the USENIX 1996 Technical Conference*, San Diego, USA, 1996.

[16] IOzone, Available at: http://www.iozone.org

SESSION

MPS, MATHEMATICAL MODELING AND PROBLEM SOLVING WORKSHOP

Chair(s)

Prof. Hayaru Shouno

Codebook Graph Coding of Descriptors

Tetsuya Yoshida[1] **and Yuu Yamada**[2]
[1]Graduate School of Humanities and Science, Nara Women's University, Nara, Japan
[2]Grad. School of Info. Science and Technology, Hokkaido University, Sapporo, Japan

Abstract— *This paper proposes a method called Codebook Graph Coding to improve the standard orderless Bag of Features (BoF) representation of images for whole-image categorization tasks. Inspired by the "bag of words" representation in document analysis, BoF has been widely used in image analysis. However, as in document analysis, the locations of "visual words" (features) in images are discarded in the standard BoF representation. Since the locations of visual words in images seems more important compared with document analysis, use of locational relationship may contribute to improving the performance of whole-image categorization. Instead of the proximity of descriptors and features in the feature space, the proposed method utilizes the proximity of descriptors in each image when coding images as bag of features. For each image, the proposed method first constructs a graph to represent the locational relationship of descriptors in the image. Then, the connectivity relations encoded as a set of graphs are aggregated into another graph of features. Finally, this graph is used to encode the descriptors in each image as a pooled feature. Preliminary experiments are conducted to investigate the effectiveness of the proposed method and compared with other BoF methods. Results are encouraging, and indicate that it is worth pursuing this path.*

Keywords: image categorization, descriptors, coodbook, graph

1. Introduction

Thanks to inexpensive and high resolution digital cameras, it is possible to take a lot of photos in daily life these days. However, the increasing volume of digital images makes it difficult to manage them manually. Thus, various efforts have been conducted to automatically classify or cluster them solely based on the contents. Due to the success of machine learning techniques in document analysis, use of these techniques is expected to contribute to automatic recognition of digital images. However, since many statistical machine learning techniques are based on the vector representation of data, for applying such techniques, it is necessary to represent digital images in vector representation.

The "bag of word" representation in document analysis is recently extended to "bag of features" (BoF) representation for better whole-image categorization tasks [1], [2], [3]. After extracting local keypoints from images and representing them as descriptors (e.g., SIFT descriptors [4], [5]), the descriptors are clustered into so-called "visual words" (features) so that techniques developed in document processing can be used for whole-image categorization tasks. However, as in the bag of words representation in document analysis, the location of visual words in an image and the locational relationship between visual words are discarded in BoF representation. Since the locations of visual words in images seems more important compared with document analysis, use of locational relationship may contribute to improving the performance of whole-image categorization.

Toward better whole-image categorization under the framework of BoF, this paper proposes a method called Codebook Graph Coding based on the proximity of descriptors. Instead of the proximity of descriptors and features in the feature space, the proposed method utilizes the proximity of descriptors in each image when encoding images as bag of features. For each image, the proposed method first constructs a graph to represent the locational relationship of descriptors in the image. Then, the connectivity relations in a set of graphs are aggregated into another graph of features. Finally, this graph is used to encode the descriptors in each image as a pooled feature. Experiments over scene15 and Caltech-101 datasets are conducted, and the performance of our approach is investigated through the comparison with other BoF methods. Results are encouraging, and indicate that utilization of the proximity of descriptors in each image can lead to better performance.

The rest of this paper is organized as follows. Section 2 explains related work to clarify the context of this research. Section 3 explains the details of our approach. Section 4 reports the evaluation of our approach and a comparison with other methods. Section 5 summarizes our contributions and suggests future directions.

2. Related Work

2.1 Preliminaries

A bold normal uppercase letter is used for a matrix, and a bold italic lowercase letter for a vector. For a matrix $\mathbf{A}$, $\mathbf{A}_{ij}$ stands for an element in $\mathbf{A}$, and $\mathbf{A}^T$ stands for its transposition. When a matrix $\mathbf{A}$ is not singular, its inverse matrix is denoted as $\mathbf{A}^{-1}$. A vector with p ones $(1, \cdots, 1)^T$ is denoted as $\mathbf{1}_p$.

2.2 Bag of Features (BoF) Representation

Bag of Features representation is often used in image analysis to represent a 2-dimensional image as a vector based

on the frequencies of visual features [1], [2], [3]. SIFT [4], [5] and SURF [6] are often used for extracting local keypoints from images and representing them as vectors (called descriptors). Usually, extracted descriptors from images are clustered, and the centroids in the clusters are treated as "visual words" (features) in the images. The set of visual words is called a codebook. Then, codes of the descriptors are constructed based on the codebook, and each image is represented as a vector based on the frequencies of features in the image.

Suppose each descriptor is represented as a p-dimensional vector, and n descriptors are extracted from an image. Let the descriptors be represented as a matrix $\mathbf{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n] \in \mathbb{R}^{p \times n}$. Also, suppose the codebook for a set of images is represented as a matrix $\mathbf{B} = [\boldsymbol{b}_1, \cdots, \boldsymbol{b}_m] \in \mathbb{R}^{p \times m}$, where each $\boldsymbol{b}_j \in \mathbb{R}^p$ corresponds to a feature.

The standard BoF representation is based on hard vector quantization (VQ) [7] coding with respect to the codebook $\mathbf{B}$. Codes for the descriptors under VQ coding is determined based on the following constrained least square optimization:

$$\mathbf{C}^* = \arg\min_{\mathbf{C}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \mathbf{B}\boldsymbol{c}_i\|^2 \quad (1)$$

$$\text{s.t.} \, \|\boldsymbol{c}_i\|_{\ell_0} = 1, \|\boldsymbol{c}_i\|_{\ell_1} = 1, \boldsymbol{c}_i \geq \mathbf{0}, \forall i$$

where $\boldsymbol{c}_i \in \mathbb{R}^m$ is the code for descriptor $\boldsymbol{x}_i$, and $\|\cdot\|$ denotes the standard Euclidean norm. The matrix $\mathbf{C}^* = [\boldsymbol{c}_1, \cdots, \boldsymbol{c}_n] \in \mathbb{R}^{m \times n}$ denotes the corresponding codes for the descriptors $\mathbf{X}$. The constraint $\|\boldsymbol{c}_i\|_{\ell_0} = 1$ means that there will be only one non-zero element in each code $\boldsymbol{c}_i$, which corresponds to the quantization id of $\boldsymbol{x}_i$. The non-negative, ℓ_1 constraint $\|\boldsymbol{c}_i\|_{\ell_1} = 1, \boldsymbol{c}_i \geq \mathbf{0}$ means that the coding weight for each descriptor is one.

In practice, the single non-zero element (quantization id) for each descriptor is usually found by the nearest neighbor search in the p-dimensional Euclidean space (which is called the feature space). After constructing the matrix $\mathbf{C}^*$, taking the row sum of $\mathbf{C}^*$ (i.e., $\mathbf{C}^*\mathbf{1}_n$) results in an m-dimensional vector, which is the sum-pooled representation of the image under the standard BoF representation.

2.3 Extensions of BoF

As in the bag of words in document analysis, the locations of visual words in an image and their locational relationship are discarded in the standard orderless BoF representation. Spatial Pyramid Matching (SPM) [8] tries to tackle this problem by defining a similarity of a pair of images based on an approximate global geometric correspondence of features in images. SPM partitions an image into increasingly fine sub-regions, and histograms of features are computed in each sub-region. Finally, the histograms are concatenated into a (high-dimensional) vector.

Since sub-regions correspond to a global locations in an image, the concatenated vector represents an approximate global locations of features to some extent. It is reported that the defined similarity in SPM can improve the performance of Support Vector Machine (SVM) [9]. However, the dimension of concatenated vector increases as the number of levels (sub-divisions) increases.

With VQ coding in eq.(1), only one feature is assigned to a descriptor in the standard BoF. On the other hand, Locality-constrained Linear Coding (LLC) assigns several features to a descriptor based on the proximity of descriptors and features in the feature space [10]. LLC uses the locality constraint to project each descriptor into its local coordinate system. Theoretically, LLC finds the codes of descriptors which minimizes the following objective function:

$$h(\mathbf{C}) = \sum_{i=1}^{n} \|\boldsymbol{x}_i - \mathbf{B}\boldsymbol{c}_i\|^2 + \lambda \|\boldsymbol{d}_i \odot \boldsymbol{c}_i\| \quad (2)$$

$$\text{s.t.} \mathbf{1}_m^T \boldsymbol{c}_i = 1, \forall i$$

where $\odot$ denotes Hadamard product (i.e., element-wise multiplication) of vectors. The vector $\boldsymbol{d}_i \in \mathbb{R}^m$ is defined as follows:

$$\boldsymbol{d}_i = \exp\left(-\frac{\text{dist}(\boldsymbol{x}_\text{i}, \mathbf{B})}{\sigma}\right) \quad (3)$$

where $\text{dist}(\boldsymbol{x}_\text{i}, \mathbf{B}) = [\text{dist}(\boldsymbol{x}_\text{i}, \boldsymbol{b}_1), \cdots, \text{dist}(\boldsymbol{x}_\text{i}, \boldsymbol{b}_\text{m})]^\text{T}$, and $\text{dist}(\boldsymbol{x}_\text{i}, \boldsymbol{b}_\text{j})$ is the Euclidean distance in the feature space between descriptor $\boldsymbol{x}_i$ and feature $\boldsymbol{b}_j$. A parameter σ adjusts the weight decay with respect to locality in eq.(3).

In practice, instead of solving the optimization problem to minimize the function in eq.(2), for faster computation of codes, approximation of LLC is actually conducted in [10]. For each descriptor $\boldsymbol{x}_i$, k-nearest neighbor features of $\boldsymbol{x}_i$ in the feature space are treated as its local codebook $\mathbf{B}_i$, and the code $\boldsymbol{c}_i^*$ for the descriptor is defined as $\boldsymbol{c}_i^* = \arg\min_{\boldsymbol{c}_i} \|\boldsymbol{x}_i - \mathbf{B}_i\boldsymbol{c}\|^2$ s.t. $\mathbf{1}_k^T \boldsymbol{c}_i = 1$. Finally, the code $\boldsymbol{c}_i$ is defined by padding zeros to other non-selected features in $\boldsymbol{c}_i^*$.

3. Codebook Graph Coding

Fig. 1 shows the overview of our approach. As in the standard BoF, descriptors are extracted from each image, and extracted descriptors are clustered to define features. In addition, a descriptor graph is constructed based on the proximity of descriptors in each image. Then, another graph called codebook graph is constructed by aggregating the connectivity relations in the set of descriptor graphs. Finally, codes for descriptors are defined based on the transition probability in the codebook graph as well as the nearest neighbor of each descriptor. The details are explained in the following subsections.

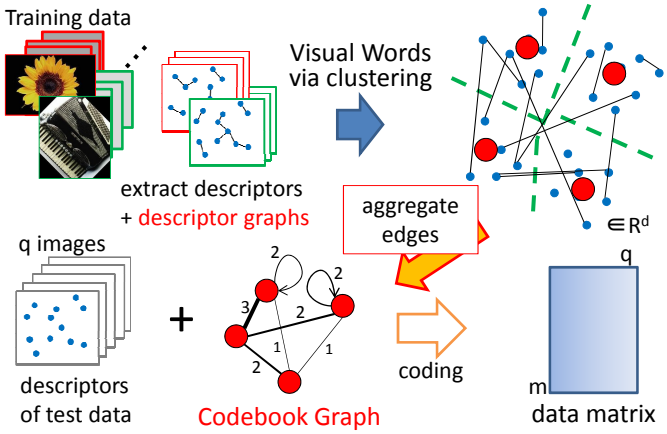


Fig. 1: Overview of our approach

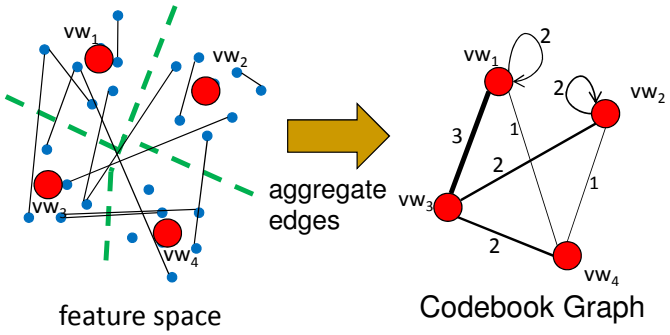


Fig. 2: Code Book Graph

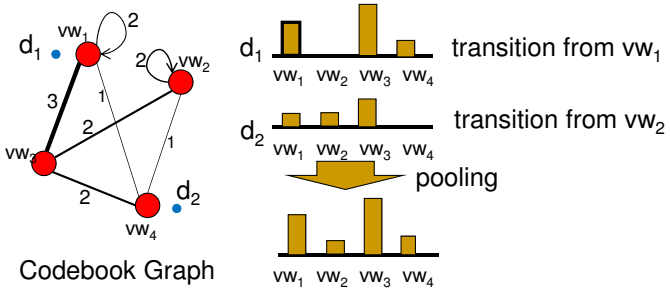


Fig. 3: Coding using a Code Book Graph

3.1 Descriptor Graph

In our approach, the locational relationship of descriptors in an image is represented as a graph (which is called descriptor graph). Nodes in a descriptor graph corresponds to descriptors in the image, and edges are defined based on the proximity of descriptors within the image. Various proximity graphs are defined based on the proximity of nodes in the literature. Among them, Nearest Neighbor Graph (NNG) [11], Relative Neighborhood Graph (RNG) [12], and Gabriel Graph (GG) [13] are used in our approach.

Locational relationship of descriptors is also used in SPM [8] to some extent. However, since absolute locations of descriptors in the given images are used, the constructed BoF representation is not robust with respect to translation, rotation and scaling. On the other hand, since relative relations of descriptors in an image is represented as a descriptor graph, it would be robust to these transformations. Furthermore, SPM represents an approximate global geometric correspondence of descriptors, but local relationship of descriptors is represented based on their proximities in our approach.

3.2 Codebook Graph

By aggregating the connectivity relations in the set of descriptor graphs, another graph called codebook graph is defined to represent the relation of features based on the locational relationship of descriptors for a set of images. Suppose descriptors are extracted from the images and clustered into m features. Hard clustering of descriptors (e.g., k-means) corresponds to defining a function $f(\cdot)$ from the given descriptors to the defined features. Also, suppose a set of descriptor graphs $\mathcal{G}$, where each $G_l(V_l, E_l) \in \mathcal{G}$ corresponds to the descriptor graph for the -th image, is already constructed. Here, V_l denotes the nodes, and E_l denotes the edges in G_l.

Construction of codebook graph is illustrated in Fig. 2. In the codebook graph, each feature (visual word) is represented as a node. The edge weight between a pair of nodes vw_i and vw_j (with feature IDs i and j) in the codebook graph is defined as:

$$w_{ij} = \sum_{G_l \in \mathcal{G}, v_s \in V_l, v_t \in V_l} |\{(v_s, v_t) \in E_l | f(v_s) = i \wedge f(v_t) = j\}| \quad (4)$$

where v_s and v_t are vertices in G_l, and $|\{\cdot\}|$ denotes the cardinality of the set. Intuitively, weight w_{ij} in eq.(4) corresponds to the number of edges between the sub-region dominated by i-th feature vw_i and the sub-region dominated by j-th feature vw_j in the feature space.

3.3 Codebook Graph Coding

Codes of descriptors under the proposed Codebook Graph Coding (CGC) are defined based on the transition probability in the codebook graph as well as the nearest neighbor of each descriptor. As in the standard BoF, the nearest feature for each descriptor is first determined. Then, the transition probability from the feature to all features in the codebook is used when defining the code for the descriptor. Finally, the codes of the descriptors in an image are pooled together to define the pooled feature of the image. These processes are summarized in Fig. 3.

For a given image, let $\mathbf{C}_0$ denotes the codes of descriptors based on VQ coding in eq.(1). The edge weights defined in eq.(4) can be represented as a square matrix $\mathbf{W} \in \mathbb{R}^{m \times m}$. Also, by defining the degree vector $\boldsymbol{d} = \mathbf{W}\mathbf{1}_m$, the degree

matrix of the codebook graph is defined as a diagonal matrix $\mathbf{D} = \mathrm{diag}(\boldsymbol{d})$ [1]. Then, based on the transition provability matrix, which is defined as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$, the codes for the descriptors by CGC are define as:

$$\begin{aligned} \mathbf{C} &= \mathbf{P}\mathbf{C}_0 \\ &= \mathrm{diag}(\mathbf{W}\mathbf{1}_{\mathrm{m}})^{-1}\mathbf{W}\mathbf{C}_0 \end{aligned} \quad (5)$$

Finally, the defined codes are pooled together to get the corresponding pooled feature of the image. The standard pooled feature in BoF is based on sum pooling [8], which is defined as $\boldsymbol{c} = \mathbf{C}\mathbf{1}_n$. However, since max pooling [14], which takes the maximum value for each row in $\mathbf{C}$, is reported to contributes to improving the linear separability of the pooled features, max pooling is used in CGC.

4. Evaluation

4.1 Experimental Setting

Experiments for whole-image categorization tasks were conducted over scene15 dataset [2] and Caltech-101 dataset [3]. The former dataset contains scenery images, and the latter contains general object images. In the experiments, a specified number of images for each class (category) were selected as training data, and the remaining images were treated as test data. Since the number of images contained in each class drastically differs in these datasets, 100 images for each image were selected as test data in scene15, and 30 images were selected as test data in Caltech-101. As the quality measure of image classification, Classification Rate (CR) was used in the experiments. CR is defined as:

$$CR = \frac{\text{number of correct images}}{\text{Number of test images}} \quad (6)$$

SIFT descriptors [4], [5] were extracted from each image, and the extracted descriptors were clustered using k-means to define the codebook. After encoding the descriptors in each image, the codes were normalized under ℓ_2 norm. Finally, a pooled feature of each image was constructed from the codes using max pooling [14].

When classifying images with respect to the constructed pooled features, we used SVM [9], which is known with its high classification performance. Since the standard SVM is originally designed for two-class problem, we used One-vs-One Linear SVM to classify multiple classes in the datasets. One-vs-One SVM realizes multiple-class classification by conducting all the pairwise classification. However, when the number of classes is large (e.g., 101 classes in Caltech-101), its takes too much time since the number of combination of classifications increases. Thus, for reducing the running time in classification, we used a simple linear kernel in SVM.

[1] Each element in $\boldsymbol{d}$ is placed on the diagonal element in $\mathbf{D}$, and non-diagonal elements in $\mathbf{D}$ are set to zeros.

[2] http://www-cvr.ai.uiuc.edu/ponce_grp/data/

[3] http://www.vision.caltech.edu/Image_Datasets/Caltech101/

Table 1: Results of preliminary experiments

	1-NNG	3-NNG	5-NNG	RNG	GG	grid
CR	0.873	0.871	0.874	0.870	0.866	0.818

Table 2: characteristics of code book graphs

	1-NNG	grid
mean degree	150.46	737.19
average path length	1.85	1.27

As for comparison, experiments were conducted for 1) the standard BoF (as baseline), 2) SPM [8], 3) LLC [10], and 4) CGC in Section 3.

4.2 Preliminary Experiments

The performance of our method depends on i) sampling of descriptors from images, and ii) type of proximity graph for the descriptor graphs in Section 3.1. We first report their influences in our method.

As for the sampling of descriptors, two kinds of sampling, namely, a) sparse sampling, and b) dense sampling, have been widely used in the literature. Sparse sampling locates keypoints only for distinctive invariant locations in an image (as in the original SIFT algorithm [4], [5]). On the other hand, dense sampling uniformly samples keypoints from an image, irrelevant to the properties of keypoints. Usually keypoints are located over some grid in the image, and arranged in a matrix-like grid in the image.

For a) sparse sampling, keypoints were located by SIFT algorithm in each image, and their SIFT descriptors were connected as a descriptor graph. Five types of proximity graphs were evaluated: Nearest Neighbor Graph (NNG) [11] (the number of neighbors were set to 1, 3, 5), Relative Neighborhood Graph (RNG) [12], and Gabriel Graph (GG) [13].

As for b) dense sampling, the width of grid was set to 5 pixels, the scale in SIFT algorithm was set to 16, and the sampled SIFT descriptors were arranged as a gid graph. The codebook size (the number of features) m was set to 1024, and 10 classes from Caltech-101 were used in the experiment. The results are summarized in Table 1.

For the standard BoF representation, it is known that dense sampling or random sampling of keypoints contributes to improving the performance of classifiers, albeit the number of descriptors drastically increases [15]. On the other hand, the results in Table 1 indicate that dense sampling degrade the performance of CGC.

Our current conjecture for this observation is that, when the number of edges in the codebook graph increases, the graph gets densely connected. Since the transition probabilities from nodes (features) becomes similar to each other as the connectivity of nodes increases, the codes of descriptors become almost the same with respect to a densely connected codebook graph. To verify this, the average degree and the

Table 3: Scene15 w.r.t. CR

m	1024	2048	4096
base	0.559	0.566	0.561
SPM	0.560	0.593	0.613
LLC	0.600	0.620	0.626
CGC	0.535	0.573	0.597

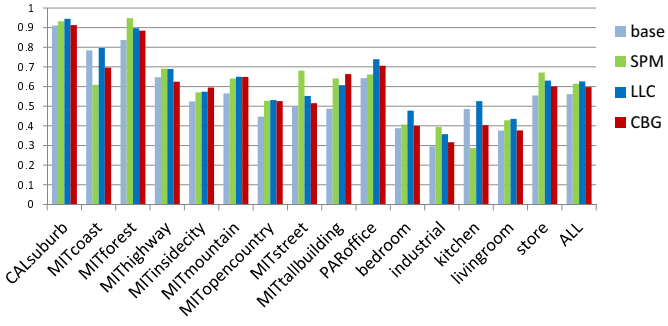


Fig. 4: Results of scene15 for each class ($m = 4096$)

average path length of 1-NNG (with the least number of edges) and of grid graph (with the largest number of edges) are shown in Table 2. The results in Table 2 indicate that almost all nodes are connected in grid graph. Thus, this might be the reason for the observation in Table 1.

Based on the above observations, in the following experiments, we used sparse sampling for extracting descriptors, and 1-NNG for descriptor graph.

4.3 Results and Discussions

The results of scene15 are summarized in Table 3. The codebook size m was set to 1024, 2048, and 4096 in the experiments. In general, class separability improves as the number of features increases. The results in Table 3 match this phenomenon. In addition, since the codebook graph gets sparsely connected as m increases, the codebook size will affect more to CGC, compared with other methods.

Unfortunately, although CGC showed comparable performance, and outperformed the standard BoF, it could not outperform other methods. The classification rate of each class in scene15 (m = 4096) is summarized in Fig. 4. Compared with the standard BoF, SPM did not outperform BoF in class "coast" and "kitchen". On the other hand, the performance degradation in CGC is less compared with SPM.

Table 4: Caltech-101 (w.r.t. CR)

m	1024
BoF	0.369
SPM	0.435
LLC	0.413
CGC	0.310

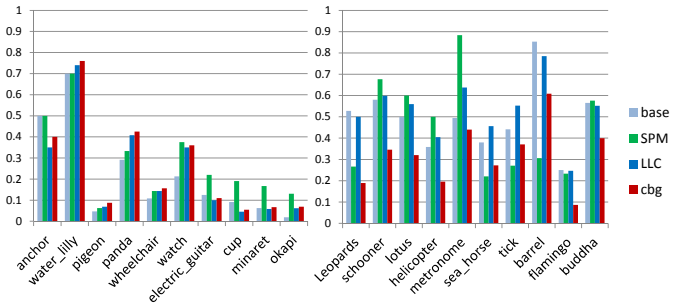


Fig. 5: Comparison with LLC in Caltech-101 (left: top 10 classes, right:worst 10 classes)

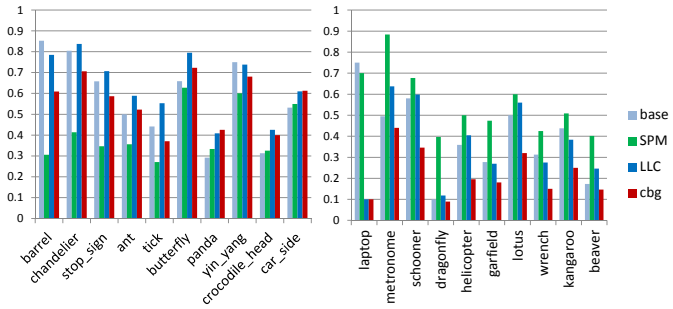


Fig. 6: Comparison with SPM in Caltech-101 (left: top 10 classes, right:worst 10 classes)

Since the number of classes is large in Caltech-101, in order to reduce the running time for classification, the codebook size m was set to 1024 for Caltech-101. The results are summarized in Table 4. Unfortunately, CGC could not outperform even BoF in terms of the overall classification rate. However, for some classes, the proposed method outperform LLC and SPM. More detailed analysis are shown in Fig. 5 and Fig. 6. Fig. 5 shows 10 classes for which CGC outperformed LLC (left hand side), and 10 classes in vise versa (right hand side). Similarly, comparison with SPM is summarized in Fig. 6. As shown in these figures, utilization of the proximity of descriptors in each image can lead to better performance in some classes, but not all classes in our current approach.

Our current conjecture for these results is as follows. Since Caltech-101 contains a lot of classes than scene15, much more descriptors were used when constructing the codebook graph. When the number of descriptors increases, the number of edges in the codebook graph increases (unless a huge number of features are used). As explained in Section 4.2, since the separability of pooled features by CGC will degrade when the codebook graph is densely connected, this may lead to performance degradation in our approach.

5. Conclusions

This paper proposed a method called Codebook Graph Coding to improve the standard orderless Bag of Features (BoF) representation of images for whole-image categorization tasks. Instead of the proximity of descriptors and features in the feature space, the proximity of descriptors in each image is used when coding images as bag of features. For each image, the proposed method first constructs a graph to represent the locational relationship of descriptors in the image. Then, the relations encoded as the graphs are aggregated into another graph of features. Finally, this graph is used to encode the descriptors in each image in BoF representation.

Preliminary experiments are conducted to investigate the effectiveness of the proposed method. By using sparse sampling for extracting descriptors from images and 1-NNG for defining descriptor graphs, our method is compared with the standard BoF, SPM and LLC. Although our method could not outperform them in terms of the overall classification rate, results indicate that reflecting the proximity of descriptors in each image can lead to performance improvement in image categorization tasks. We plan to conduct more in-depth analysis of our approach, especially the structure of codebook graph, and extend it based on the analysis in near future.

Acknowledgment

This work is partially supported by the grant-in-aid for scientific research (No. 24300049) funded by MEXT in Japan.

References

[1] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on Statistical Learning in Computer Vision*, ser. ECCV'04, 2004, pp. 1–22.

[2] F. Jurie and B. Triggs, "Creating efficient codebooks for visual recognition," in *Proc. of the 10th IEEE International Conference on Computer Vision*, 2005, pp. 604–610.

[3] A. Bosch, X. Muòoz, and R. Martí, "Which is the best way to organize/classify images by content?" *Image and Vision Computing*, vol. 25, no. 6, pp. 778–791, 2007.

[4] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision*, ser. ICCV '99, vol. 2. Washington, DC, USA: IEEE Computer Society, 1999, pp. 1150–1157. [Online]. Available: http://dl.acm.org/citation.cfm?id=850924.851523

[5] ——, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Proceedings of the 9th European Conference on Computer Vision*, ser. ECCV'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 404–417.

[7] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Springer, 1991.

[8] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 2169–2178, 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1641019

[9] N. Cristianini and J. Shawe-Taylor, Eds., *An Introduction to Suport Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[10] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong, "Locality-constrained Linear Coding for image classification," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, ser. CVPR'10. IEEE, June 2010, pp. 3360–3367. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5540018

[11] D. Eppstein, M. Paterson, and F. Yao, "On nearest-neighbor graphs," *Discrete and Computational Geometry*, vol. 17, no. 3, pp. 263–282, 1997.

[12] J. Jaromczyk and G. Toussaint, "Relative neighborhood graphs and their relatives," in *Proceedings of the IEEE*, vol. 80, no. 9, 1992, pp. 1502–1517. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=163414

[13] G. K. Ruben. and S. R. R, "A new statistical approach to geographic variation analysis," *Systematic Zoology*, vol. 18, pp. 259–278, 1969.

[14] Y.-L. Boureau, J. Ponce, and Y. Lecun, "A theoretical analysis of feature pooling in visual recognition," in *27TH International Conference on Machine Learning, HAIFA, ISRAEL*, ser. ICML'10, 2010.

[15] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *9th European Conference on Computer Vision*, ser. ECCV'06. Springer, 2006, pp. 490–503.

A Personal Classification Method Using Spatial Information of Multi-channel EEG

Yu Ishikawa, Chinami Yoshida, Masami Takata, Hiroyasu Kamo, Kazuki Joe
Nara Women's University, Nara, 630-8506, JAPAN

Abstract –*Biometric authentication using various biological information is studied by many researchers. We study a feature extraction method available for personal authentication by focusing on EEG in the biological information. In addition, since electroencephalograph technology has advanced significantly in recent years, multi-channel EEG is possible to be relatively easily measured. Therefore, in this paper, as EEG features, we propose a method using a cross-correlation between electrodes obtained from the multi-channel electroencephalograph. In validations, a feature combination, which is obtained from the proposed method and time-frequency analysis, is used. A personal classification is performed by applying SVM to obtained features. Moreover, by detailed validations about the proposed method, we evaluate the possibility of the cross-correlation between electrodes as features for the personal authentication.*

Keywords: biometric authentication, brain wave, feature extraction

1 Introduction

As the spread of Internet infrastructure in recent years, various social networking services such as on-line shopping are provided for the information society. According to the increase of usability, crimes using the Internet are increasing rapidly, and the importance of authentication technologies to prevent such unauthorized access is required more than ever. While personal authentication by ID and password has been used mainly in the social networking services available at the time, it would have been forged easily against prying eyes or brute-force authentication attempts. As just described, conventional authentication technologies cannot be said to be reliable means necessarily in terms of safety, and biometric authentication is getting a lot more attention recently.

The biometric authentication refers to the personal authentication using biometric information. The biological information for the authentication includes fingerprint, iris, face, voiceprint, and handwriting, where plagiarism is difficult as compared with the traditional password-based authentication. In particular, iris or fingerprint based authentication does not provide high recognition rates but also be used in practical applications, but there are also reports that some authentication systems are forged [1]. Since the biological information is exposed to the outside at all times, it can be acquired for forgery on the basis of the biometric information. A vein based authentication system that uses in-vivo (not exposed to the outside) information is introduced while there is a report that spoofing is possible even for the vein authentication system.

In this paper, a biometric authentication using brain waves as the biometric information is proposed. Electroencephalogram (EEG) is in-vivo information and superior to confidentiality because it is measured neuronal activity of a number of cerebral cortices and not measurable without wearing electroencephalograph.

Studies utilizing brain waves for authentication are already underway by various researchers. For example, EEGs of forty examinees during open-eye-closed-eye are measured for personal identification to get an accuracy of 80% is reported in [2], and EEG rhythms of four examinees during closed-eye are analyzed to get an accuracy of 90% or more [3]. Other studies include personal authentication by visual evoked potential [4] and verbal recall problems and/or potential recall movement [5]. In such previous studies, auto-regressive (AR) models [2] [3] [6] or neural networks [4] [5] for feature extraction have been proposed, but their computational costs are too expensive while the authentication performances are improved more than 90%. Therefore, it is required to extract features of less computational costs.

In addition, the electroencephalograph technology has advanced significantly in recent years. Multi-channel electroencephalographs have been applied by a professional engineer and have been used in the medical field until now. Today, they have advanced to the available level on a personal and daily basis. Since electroencephalographs (Geodesic Sensor Net) or the like of which 256 channels are applied in just 15 minutes have also been developed, we are possible to easily measure a multi-channel EEG. The use of a multi-channel electroencephalograph gives spatial information which cannot be obtained by using a single-channel one. The study using correlation between electrodes which is one of spatial information has been applied extensively in the field of emotion recognition [7] [8]. By using those features, we aim to extract features with less computational costs.

Thus, in this paper, we validate whether the spatial information obtained from a multi-channel EEG is available as features for personal authentication. The study to perform personal authentication has two problems: an effective feature extraction for authentication, and threshold settings to be used in the determination. In this paper, we use the words of "personal classification" rather than "personal authentication" in order to focus on the effective feature extraction which is the first problem in brain waves authentication. It does not mention threshold settings.

The rest of the paper is organized as follows. In section 2, we introduce related works to be compared. In section 3, we propose personal classification methods. The proposed methods are validated with some experiments in section 4.

2 Related works

We introduce a study for EEG personal authentication with light computational cost using average power spectrum as feature values [9]. In this study, a method of personal authentication by EEG brain waves during virtual driving operation, which means a simple driving simulator with tracing route, is proposed. Spectral analysis by Fourier transform for the extraction of individual feature is adopted based on the fact that there are individual differences in brain wave spectrum in the α-β wave band. Exactly saying, the α and the β wave bands are divided into α1-α4 and β1-β4 regions, respectively. In each region, the average of power spectrums as individual feature is evaluated for authentication. Evaluating the authentication performance, the Equal Error Rate (EER) of the tracing route and driving simulator are 0.35 and 0.36, respectively.

Next, as a study using spectrum analysis, we introduce our study [10] that expanded the time axis and the frequency bands on the basis of the existing research [9]. We performed personal authentication from EEG with five tasks including breathing, finger, pass, song, and sport. This method uses the results of short-time Fourier transform (STFT) which is one of the time-frequency analyses as the features. We use 4-40Hz brain waves (θ to γ waves). The θ-γ wave bands are partitioned into two, three, five, and three regions of θ1-θ2, α1-α3, β1-β5, and γ1-γ3, respectively. The average power spectrum for each region by time is calculated to be used as feature values. Using all of five tasks, the EER achieves the best 0.03. In this paper, we use these two types of features to compare the proposed method described in section 3.

3 An individual classification method

3.1 Overview

This study aims at an accurate personal authentication using EEG. Therefore, we select valid features to identify the individual person. In this section, we describe personal classification method proposed in this paper. Steps are as follows: Step 1. EEG measurement, Step 2. Feature extraction, and Step 3. Personal classification.

In Step 1, the individual EEG is measured multiple times to create data. In Step 2, the multiple features being necessary for personal classification are extracted from the data in Step 1. Finally, Step 3 performs personal classification based on the features extracted by Step 2 to validate whether each feature is applicable to EEG-based personal authentication.

3.2 Feature extraction method

As features to be used for personal classification based on EEG, we propose the following five types (Feature 1-5) using a cross-correlation coefficient between electrodes.

Feature 1. Cross-correlation coefficient between electrodes of measured data

Feature 2. Cross-correlation coefficient between electrodes of measured data after preprocessing

Feature 3. Cross-correlation coefficient between electrodes of measured data after preprocessing by time

Feature 4. Cross-correlation coefficient between electrodes of power spectrum after frequency analysis

Feature 5. Cross-correlation coefficient between electrodes of power spectrum after time-frequency analysis

Feature 1 is calculated using the cross-correlation coefficient between electrodes of measured original data as the features. The cross-correlation coefficient X between electrode l and m is represented by the following formula.

$$X_{lm} = \frac{\sum_{i=1}^{n}(l_i - \bar{l})(m_i - \bar{m})}{\sqrt{\sum_{i=1}^{n}(l_i - \bar{l})^2}\sqrt{\sum_{i=1}^{n}(m_i - \bar{m})^2}}$$

where n is the measured data length.

In Feature 2 and 3, a preprocessing is performed on the measured original data. A band-pass filter is applied as the preprocessing. The measured EEG includes artifacts produced by biological phenomenon such as myopotential, eye movement and heartbeat or by environment such as AC Interference. For that reason, the original waveform is reconstructed to remove low-frequency and high-frequency band which is not included in the EEG. Feature 2 is calculated using the cross-correlation coefficient between electrodes from reconstructed data as with Feature 1. Since artifacts are removed from the features, it is able to use the cross-correlation just consisting of EEG frequency bands. Feature 3 is calculated using the cross-correlation coefficient between electrodes for each short data which is cut out from the reconstructed data using a window function. By this

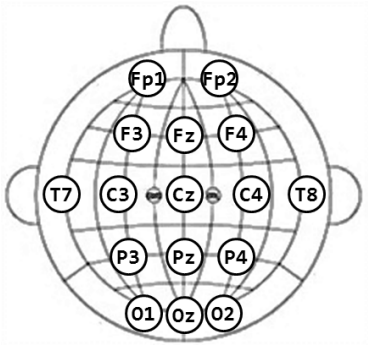


Figure 1 BioSemi electrode position

approach, the feature in consideration of the time change is obtained.

Feature 4 and 5 are obtained from the cross-correlation coefficient between electrodes of power spectrum after frequency analysis. Fast Fourier Transform (FFT) is applied as a frequency analysis. Only EEG frequency bands are cut out from the data after FFT to be used. This process is to remove artifacts as with the above-mentioned preprocessing. Feature 4 is calculated using the cross-correlation coefficient between electrodes of power spectrum after FFT. Feature 5 is calculated using the cross-correlation coefficient between electrodes for each short data after FFT which is cut out from the original data using a window function. By this approach, it is able to consider the time change as with Feature 3.

3.3 Classification method

We apply support vector machine (SVM) for the personal classification. Using margin maximization, SVM is able to construct a classifier with high identification performance for unlearned data. In this paper, we use a multi-class SVM to classify EEG of over three persons. The multi-class SVM has two methods: One-vs-ALL SVM and One-vs-One SVM. Although One-vs-ALL SVM is implemented easily and has less computational cost, class separation becomes difficult when the number of classes is large. In addition, as used for the personal authentication, this method causes a bias of the user and other data. Therefore, in this paper, we perform the classification by using the One-vs-One SVM. A linear kernel which is commonly used is applied as the SVM kernel. Cross-validation is used to validate the method.

4 Validation

4.1 Measurement of data

We use the BioSemi as the multi-channel electroencephalograph, of which the maximum sampling rate is 2,048Hz and the maximum electrode number is 256. A bipolar lead method is used for deriving the reference electrode. In this paper, we use a BioSemi with the maximum sampling rate of 2,048Hz and 16 electrodes. The electrodes are placed as shown in Figure.1 based on International 10–20 system.

Examinees are 26 healthy women in their 20s. The measurement of ten seconds is performed 50 times per examinee. They take a break of five seconds during each measurement and remove the electrodes once in five times. The number of all data is 1,300 (multiplying 50 times with 26 examinees) in total. It is performed with sitting and resting.

In the k-fold cross-validation, k is defined as 5, 3 and 2. In the case of k=5, 3 and 2, the combination of (Training set size, Test set size) is (1040, 260), (867, 433), and (650, 650), respectively. The training set and the test set are chosen randomly. The results of classification rate are obtained from average of k times. All the results are shown by percentage.

4.2 Validation 1: optimal feature values

4.2.1 Validation methods

In Validation 1, we compare classification rates between the two features described in section 2 and the five features described in section 3.3. Therefore, the comparison features are the following seven types. Note that Feature 1-5 is the same in section 3.3.

Feature 6. Power spectrum after frequency analysis

Feature 7. Power spectrum after time-frequency analysis

In the proposed method, the frequency range of 4-40Hz is used for the frequency analysis and the preprocessing by band-pass filter. This is a general EEG frequency band to be used in EEG analysis. When a window function is applied, the number and the size of windows are 16 and 0.5 seconds, respectively. The classification data length is eight seconds which are a power of two for the use of FFT. The number of used electrodes is all of 16 channels.

4.2.2 Validation results

Table 1 shows the results of Validation 1. The classification success rates with cross-validation of k=5, 3 and 2 are displayed by percentage.

First, as this result, the classification rate of Feature 1 is a 30% level, which is a significantly low classification rate in comparison with the other features. Since the using frequency bands are 0-1024Hz, the frequency other than EEG are included in Feature 1. However, as the features applying the preprocessing to Feature 1, Feature 2 using band-pass filter and Feature 3 using Fourier transform are over 70%. Other features give relatively good classification rate, too. This is due to noise rejection by using frequency bands of only 4-40Hz. From the above, the noise rejection using the frequency analysis or a band-pass filter is needed.

Table 1 Results of Validation 1 and 2

Validation	Feature	k=5	k=3	k=2
Validation 1	1	35.5	32.9	31
	2	80.1	78.2	74.3
	3	94.3	93.3	88.9
	4	82.2	78.8	75.1
	5	77.9	76.1	71.8
	6	92.3	91.4	89.8
	7	96.4	95.7	93.5
Validation 2	8	98.3	97.9	96.9

We focus on Feature 1-5 of the proposed method. Feature 1 is excluded from the consideration because it includes a great deal of noise as previously described. The classification rate of Feature 3 is the best and only over 90% among other four Features. Next, the classification rate is high in the order of Feature 4, 2, and 5. In comparison between Feature 2 and 3, applying the cross-correlation coefficient between electrodes to original data, Feature 2 with a window function is better classification rate considering the time change. In comparison between Feature 4 and 5, applying the cross-correlation coefficient between electrodes to data after frequency analysis, however, the classification rate is better without any window function. In the case of using cross-correlation coefficient between electrodes as the features, the time-series data has a personal difference than the power spectrum. Note that we should consider the time change by using the window function.

We focus on the Feature 6 and 7 in the existing method. Both classification success rates have over 90% although Feature 7 considering the change by time as features obtains higher classification rate.

The methods which obtain over 90% classification rate among all methods are good in the order of Feature 3, 6, and 7. Feature 3 is a little inferior classification rate rather than Feature 7, but the classification rate itself is very high. Therefore, the feature using the cross-correlation coefficient between electrodes is effective as a feature for the personal authentication.

4.3 Validation 2: validate a new feature values

The result of Validation 1 shows that Feature 3 and 7 have good classification rates. Thus, we propose Feature 8 using two types of Feature 3 and 7. First, Feature 8 performs a noise rejection by applying a band-pass filter to measured data. Next, it cuts out the short data by using a window function to the reconstruction data with calculating the cross-correlation coefficient between electrodes. The results obtained by these processes are defined as the features. At the same time, it performs a time-frequency analysis using the STFT to measured data with calculating the average power spectrum of each window by time. The results are also defined as the features. Feature 8 is validated by giving the two types of features to SVM.

As with Validation 1, frequency bands are 4-40Hz, and the number and the size of windows are 16 and 0.5 seconds, respectively. The classification data length is eight seconds, and the number of electrodes is 16. The validation results are shown in Table 1. Feature 8 obtains the best classification rate among the methods of Feature 1-7. In particular, it exceeds 98% in the case of k=5, and the number of errors is less than five in 260 data.

4.4 Validation 3: optimal parameters

4.4.1 Validation methods

In Validation 3, we perform the validation of parameters in Feature 8, which is proposed in section 4.3 and obtains the best classification rate. The following values are validated as parameters.

- Validate the optimal frequency bands (Validation 3a)
- Validate the optimal number and size of window (Validation 3b)
- Validate in the case of using the optimum values (Validation 3c)
- Validate the data length and the number of window (Validation 3d)

In Validation 3a, the frequency bands are changed to use band-pass filter and STFT. In Validation 1 and 2, 4-40Hz is used, but five types of frequency bands are compared as follows in Validation 3a: θ (4-8Hz), α (8-14Hz), β (14-26Hz), γ (26-40Hz), α+β+γ (8–40H) and all (4-40Hz). In this validation, the number and size of windows is unified in 16 and 0.5 seconds.

In Validation 3b, the number and size of windows is changed. Changing the number and size of windows into (1, 8), (2, 4), (4, 2), (8, 1), (16, 0.5), we calculate the classification rate. The feature, of which the number and size of the windows is one and eight seconds, is consistent with one of not using the window. Moreover, when window size is within 0.5 seconds, frequency resolution is 4Hz. Therefore, the average power spectrum of the proposed method cannot be calculated, the minimum value of the window size is defined as 0.5 seconds. The frequency bands of band-pass filter are 4-40Hz.

In Validation 3c, the personal classification rate is calculated in the case of using the optimum parameter which is obtained from the results of Validation 3a and 3b.

In Validation 3d, the change of classification rate by changing classification data length is compared to Feature 7. Since measured data is 8 seconds, the cases of 1, 2 and 4

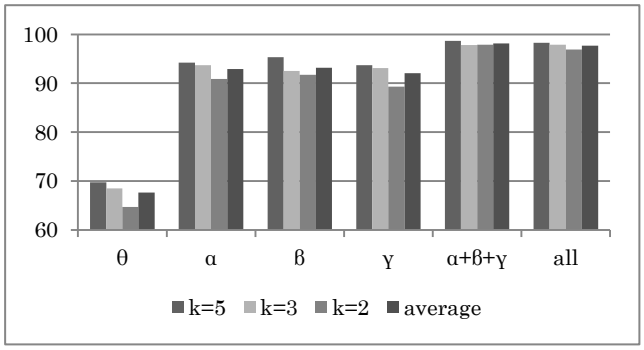


Figure 2 Results of Validation 3a

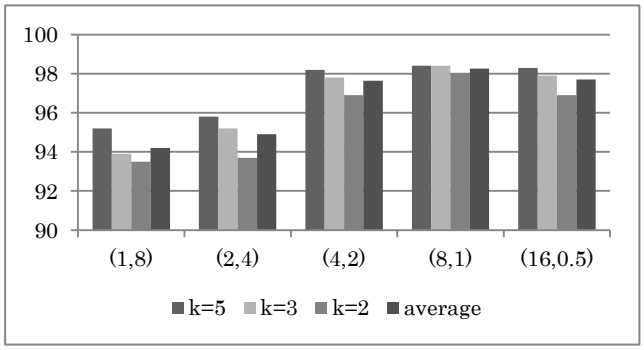


Figure 3 Results of Validation 3b

Table 2 Results of Validation 3c

Feature	k=5	k=3	k=2
8 (After parameter adjustment)	98.6	98.3	98.2

Table 3 Results of Validation 3d

data length(s)	Windows	Feature 8	Feature 7	Difference
8	**(1,8)**	95.2	93.6	1.6
	(2,4)	95.8	95.3	0.5
	(4,2)	98.2	96.5	1.7
	(8,1)	98.4	96.9	1.5
	(16,0.5)	98.3	96.4	1.9
4	**(1,4)**	92.4	92.1	0.3
	(2,2)	93.8	93.6	0.2
	(4,1)	97.1	93.9	3.2
	(8,0.5)	96.8	93.4	3.4
2	**(1,2)**	92.2	88.5	3.7
	(2,1)	91	89.2	1.8
	(4,0.5)	93	87.8	5.2
1	**(1,1)**	86	81.5	4.5
	(2,0.5)	84.5	80.6	3.9

seconds are compared; which is equal to or less than 8 seconds and data length having a power of two. In addition, the cases of changing the number of windows are compared, respectively. As the used frequency bands, the optimum results obtained from Validation 3a are adopted.

4.4.2 Validation results

Figure 2 shows the results of Validation 3a. Compared to the average values of k=5, 3, 2 of each frequency band, the classification rate is high in the order of θ, γ, α and β. In the case of using only θ wave bands, classification rate decreases remarkably. That is because the noise is easily mixed into the low-frequency. The result of using the α, β and γ of frequency bands except for θ bands with the low classification rate is (α+β+γ) of Figure 2. The better results is provided in k=5, 2 in comparison with (all) using all frequency bands. From the results, as the optimum frequency, we use α+β+γ wave bands except for θ bands with the low classification rate.

Next, Figure 3 shows the results of Validation 3b. When the number and size of windows are (4, 2), (8, 1) and (16, 0.5), the classification rate is good at the same level. In the case of (8, 1), especially, the average classification rate is the best 98.3%. In the case of (1, 8), the features quantity is less because the number of windows is one. Therefore, the classification rate becomes low. Namely, as the optimum number and size of windows, we adopt (8, 1).

In Validation 3c, the following values, the optimum parameters obtained from Validation 3a and 3b, are used. Those indicate that the optimum frequency bands are 8-40Hz using α, β and γ and the optimum number and size of windows are (8, 1). Table 2 shows the results of classification using those parameters. As the results, we obtained the best classification rate 98.6%.

The results of Validation 3d are as Table 3. Table 3 shows only the results of 5-fold cross-validation. In all cases, Feature 8 is higher classification rate than Feature 7. In addition, the longer the data length is, the better the classification results are. The case of the window size of a second is the best classification rate except the results of Feature 8 using data length of two seconds and window of (4, 0.5). From the results, the size of windows is more important than the number of windows. Next, comparing the difference between Feature 8 and 7, the difference tends to increase as the data length is longer. In other words, Feature 8 obtains high classification rate from even short data length.

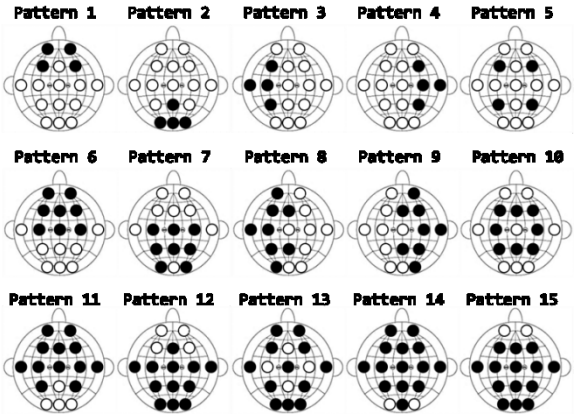


Figure 4 Electrode position pattern for Validation 4b

4.5 Validation 4: Validate the combination of electrode positions

4.5.1 Validation methods

While the classification rate is validated with fixing the number of electrodes to 16 in Validation 1-3, the following validations are performed to investigate the number and the position of electrodes for the personal classification in Validation 4.

- Validate the combination of electrodes (Validation 4a)
- Validate the number and the position pattern of electrodes (Validation 4b)

We use the same parameters as Validation 3c obtaining the best classification rate in past Validations. The data length is 8 seconds, the frequency bands are 8-40Hz, and the number and the size of windows is (8, 1). 5-fold cross-validation is performed in this condition.

In Validation 4a, the classification rate is calculated in the case of using two channels, and the combination of active electrodes is validated for personal authentication. When all 16 channels are used, the combination of two channels is 120 kinds. We validate the combination of electrodes of the top 10%, 20% and 30% classification rate in them.

In Validation 4b, we validate the classification rate in the case of changing the number and the position pattern of electrodes. The number of using electrodes is changed to 4, 8, 12 and 14. Moreover, the electrode position patterns are validated in the case of collected electrodes on the front, rear, left, right, and center side. Figure 4 shows the placement of the electrode position patterns. The validation patterns include 15 patterns in total. Black filled electrodes in the layout diagram are the used electrodes. The first, second and third line in Figure 4 show the case of using 4, 8, 12 and 14 electrodes, respectively.

Table 4 Results of Validation 4a

ch	Fp2	F4	Fz	F3	T7	C3	Cz	C4	T8	P4	Pz	P3	O1	Oz	O2
Fp1	58	68	71	66	68	67	69	72	70	78	69	79	84	83	84
Fp2		68	70	67	67	65	68	69	68	78	70	82	84	82	83
F4			69	68	74	70	70	74	73	82	71	82	87	86	86
Fz				70	78	74	71	78	77	82	71	85	89	87	88
F3					74	71	71	74	73	82	74	83	87	88	88
T7						70	72	77	72	83	75	82	85	85	86
C3							69	75	70	84	77	84	88	87	87
Cz								73	71	82	71	83	86	86	86
C4									74	82	74	84	88	87	86
T8			Top 10%							81	72	82	83	84	85
P4			Top 20%								76	86	88	88	86
Pz			Top 30%									77	85	82	83
P3													87	86	86
O1														83	86
Oz															83

4.5.2 Validation results

In Validation 4a, we validate the combination of electrodes. The results are shown in Table.4. The classification rate is rounded off to the nearest integer. Since the average classification rate of the combination of 120 kinds is 78.1%, the personal classification can work up to some extent by using 2 electrodes. However, the selection of the used electrodes is important. The combinations of obtaining the minimum and the maximum classification rates are (Fp1, Fp2) at 57% and (Fz, O1) at 89.1%, respectively. It is clear that there are large differences in the classification rate by using electrodes. Fp1 and Fp2 are the most susceptible to noise caused by eye movement because they are the closest electrodes to eyes. Therefore, they are not suitable for the personal classification. Since the classification rate is obtained close to 90% by using just 2 channels in the combination of Fz and O1, they are the combination of highly effective electrodes for authentication. Next, we focus on the electrodes of the top 30% classification rates. They almost include the combination of O1, Oz and O2 in the occipital region. Note that the combination with Fp2 is excluded. Focusing on the top 20%, the combination comprising Fp1, T8, and Pz are excluded in addition to Fp2. In the top 10%, T7 and Cz are excluded. Namely, the electrodes which are not included in the top 10% are the frontal region (Fp1, Fp2), the temporal region (T7, T8) and the central region (Pz, Cz). In particular, as the combination with O1, Oz, and O2, Fz and F3 are higher classification rates with all belonging to the top 10%. From this result, we consider that it is better for a longer distance between 2 electrodes. The classification rate is 94.3% in the case of using those 5 electrodes (O1, Oz, O2, Fz, F3).

Table 5 shows the results of Validation 4b. First, we validate the number of electrodes. When the number of electrodes is 4, 8, 12, and 14, the average classification rate of each pattern is 89.8%, 96.2%, 98.2%, and 98.4%, respectively. The classification rate is higher every time the number of used electrodes increases. Next, we focus on the electrodes

Table 5 Results of Validation 4b

pattern	channel number	position	k=5
1	4	front	81.3
2		rear	91.8
3		left	92.3
4		right	90.9
5		center	92.5
6	8	front	94.5
7		rear	97.6
8		left	96.4
9		right	96.4
10		center	96.2
11	12	front	98.3
12		rear	98.4
13		right	97.9
14	14	front	98.3
15		rear	98.5

position pattern. In comparison with the front side of the position pattern, the classification rate of the rear side is higher. The difference between the front and the rear side of the classification rate reduces as the number of used electrodes increases. In fact, since the many artifacts by eye movement appear to the front region around the electrodes, their electrodes give an influence on the personal classification even after noise rejection by a band-pass filter. Therefore, the classification in the occipital region with less noise shows good results. When the number of electrodes is small, the classification rate is greatly influenced by the electrode position. There is no much difference when the number of used electrodes is 12, 14, and 16. From these results, we conclude that 12 channels are enough for the number of electrodes to perform the personal classification.

5 Conclusions

In this paper, we validated the possibility of a personal classification using a cross-correlation coefficient between electrodes as a feature. As the results, it is possible to more accurately perform the personal classification by combining the cross-correlation between electrodes than the classification method using just time-frequency analysis. In addition, since the noise in the electrode position is low in the occipital region, a high classification rate is obtained. If you want to authenticate with a smaller number of electrodes, it is preferable to use electrodes in the occipital region than the frontal region. The data used for this validation was relatively short measurement interval. We think that some changes appear to the classification rate by changing the interval of data measurement. In future, the influence to appear to EEG should be inspected by changing the interval of data measurement such as one hour, six hour, 12 hour, one day and one week. Based on this, we aim to extract the available features to the personal authentication which is not affected by the measurement interval.

6 References

[1] T.Matsumoto, H.Matsumoto, K.Yamada and S.Hoshino, Impact of artificial "Gummy" fingers on fingerprint systems, vol. 4677, Proc. SPIE, 2002, pp. 275-289.

[2] R. B. Paranjape, J. Mahovsky, L. Benedicent and Z. Koles, The Electroencephalogram as a Biometrics, vol. 2, Proc. of 2001 Canadian Conference on Electrical and, 2001, pp. 1363-1366.

[3] M. Poulos, M. Rangoussi, V. Chrissikopoulos and A. Evangelou, Parametric person identification from the EEG using computational geometry, vol. 2, Proc. of the 6th IEEE Int. Conf. on Electronics, Circuits and Systems, 1999, pp. 1005-1008.

[4] R.Palaniappan and D.P.Mandic, Biometrics from brain electrical activity: A machine learning approach, vol. 29 no.4, IEEE Trans. Pattrn Anal. Mach Intell, 2007, pp. 738-742.

[5] S. Marcel and J. R. Millan, Pearson Authentication Using Brainwaves (EEG) and Maximum A Posteriori Model Adaption, vol. 2, IEEE Trans. on Pattern Analysis and Machine Intelligence, 2007, pp. 743-748.

[6] A.Riera, A.Soria-Frish, M.Caparrini, C.Grau and G.Ruffini, Unobtrusive biometrics based on electroencephalogram analysis, vol. 2008, EURASHIP J.Advances in Signal Processing, 2008, pp. 1-8.

[7] R. Horlings, D. Datcu , L. J. M. Rothkrantz, Emotion recognition using brain activity, In Proceedings of the 9th international conference on computer systems and technologies and workshop for PhD students in computing ACM, 2008, p. 6.

[8] K. Schaaff and T. Schultz, Towards emotion recognition from electroencephalographic signals, Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on, 2009, pp. 1-6.

[9] I. Nakanishi, H. Fukuda and S. Li, Biometric Verification Using Brain Waves toward On-Demand User Management Systems Performance differences between divided regions in $\alpha - \beta$ wave band, Proc. of the 6th International Conference on Security of Information and Networks, 2013, pp. 131-135.

[10] Y. Ishikawa, C. Yoshida, M. Takata and K. Joe, Validation of EEG Personal Authentication with Multi-channels and Multi-tasks, vol. 2, In Proceedings of 2014 International Conference on Parallel and Distributed Processing Techniques and Applications, 2014, pp. 182-188.

A Method to Maintain the Field Coverage by Static and Mobile Sensor Nodes Using Wireless Charging

Yuki Tsuchiya
Graduate School of Science
Osaka Prefecture University
Sakai, Osaka, Japan

Ryo Katsuma
Graduate School of Science
Osaka Prefecture University
Sakai, Osaka, Japan

***Abstract*—: In wireless sensor networks (WSNs), sensor nodes periodically sense, record, and transmit environmental information. WSNs require long lifetime and adequate field coverage, which can be problematic under certain conditions. Several studies have addressed these problems using energy harvesting, wireless charging, or mobile sensor nodes. In particular, mobile nodes are effective for adequate field coverage: however, appropriate node movement is critical. In addition, mobile nodes with wireless charging devices can charge the batteries of other nodes. We formulate the problem to extend lifetime and maintain field coverage by determining the positions of mobile nodes that can cover the field effectively and charge the batteries of other nodes simultaneously. We propose a coverage algorithm that can cover a field with a minimal number of nodes and a movement algorithm that determines an efficient mobile node movement schedule to charge static nodes. Simulation results, confirm that the energy charging system used by the proposed method can extend WSN lifetime up to 66%.**

I. Introduction

Wireless Sensor Networks (WSNs) are networks constructed with many sensor nodes equipped with wireless communication devices. In WSNs, sensor nodes monitor environmental data, such as temperature and humidity, and capture images. The data is sent to a sink node via multi-hop communication. No communication infrastructure is required for WSNs because each sensor node can perform both sensing and communication. The use of WSNs is expected for a variety of applications, such as environmental monitoring, ecological surveys, and marine guard [1] [2].

A major problem with WSNs is the coverage problem. A sensor node can only sense environmental information within a limited circular sensing range. As a result, node deployment must cover the entire field of interest . Node deployment cost can be reduced by placing nodes at appropriate positions. Several studies have attempted to address this issue by adjusting the position of mobile sensor nodes [13][14]. However, efficient movement scheduling of mobile nodes is required because mobile nodes consume significant amounts of energy when moving.

An other major problem of WSNs is the lifetime extension problem. A WSN is constructed by many sensor nodes that operate with limited battery power. These nodes must be recharged when the power is depleted. Conventionally, wired power supplies are employed to recharge batteries. However, wired power systems cannot be used in general WSNs; thus, operating such WSNs for extended periods is challenging. For example, it is difficult to recharge WSN node batteries when measuring temperature in a large agricultural field; significant time and costs are required to retrieve, recharge, and redeploy such sensor nodes. Energy harvesting (EH), which converts natural energy (e.g., light and heat) into electricity, has been considered to address this problem. Solar-power and wind-power generation are familiar examples of EH. The strongest point of EH is the ability to charge batteries in several locations, which is otherwise commercially unviable when conventional power charging techniques are employed. In EH, retrieving, recharging, and redeploying sensor nodes are not necessary: thus, it is expected that EH will allow WSNs to operate semi-permanently.

However, energy generation by EH is unstable and sometimes insufficient. Thus, wireless charging technology used for cellular phones and a variety of small devices is our focus. In a wireless charging system, two independent devices are used to send and receive electric power. A node with a sending device consumes energy to charge the battery of a node with a receiving device. It has been reported that electromagnetic induction type wireless charging systems are size and have greater than 80% power efficiency[3].

When mobile nodes have sending devices and static nodes have receiving devices, determining the loci of the mobile nodes when repairing the field coverage and charging batteries of static nodes becomes problematic. We have attempted to solve this problem. We propose a method to allow some mobile nodes with sending devices and solar panels to cover the field, and the other mobile nodes charge the exhausted batteries of the static nodes.

The proposed algorithm is comprised of a coverage algorithm and a movement algorithm. When a field is covered by static as well as mobile sensor nodes, the coverage algorithm determines the positions of a minimal number of mobile nodes that can cover the field by determining enclosed regions that are not covered by sensor nodes. The movement algorithm determines the destination points of free mobile nodes. The movement algorithm reduces energy consumption when moving by performing cascade movement [17] in order to avoid moving over long distances.

Our simulation results show that the proposed method extends WSN lifetime up to 66% for a 100×100 [m^2] field with 20 - 115 nodes. We have confirmed that the coverage algorithm can repair field coverage with a sufficiently small number of nodes. We have also confirmed that employing the

proposed method in a honeycomb structure is more effective than random deployment.

The remainder of this paper is organized as follows. We describe related work in Section II. Assumptions and the problem formulation are discussed in Section III. The proposed method is presented in Section IV. A performance evaluation is discussed in Section V, and conclusions are presented in Section VI.

II. RELATED WORK

One of the major problems with WSNs is lifetime extension because sensor nodes operate with limited battery power. Recently, many studies have examined using EH technology have to address this problem [4] [8]. In addition, several studies have explored charging node batteries using robots [15] [5].

Brunelli et al. proposed EH technology that uses an efficient and small solar battery module [4]. Their study showed that maximum power point tracking, which can maximize energy generation, allows sensor node batteries to be charged under a variety of solar intensity conditions. The efficiency of power conversion despite the small solar battery was reported to be up to 80%, which is sufficient power generation to charge a sensor node battery. Peng et al. proposed a system that determines charging sequences to be executed by the mobile charger to extend the lifetime of a WSN [5]. However, these studies did not consider field coverage because mobile robots only move for charging.

Another significant problem with WSNs is coverage. For environmental monitoring, it is necessary to cover the target field completely with the minimal number of sensor nodes to enable sensing at any point within the field.

Bai et al. proposed two different deployment patterns, i.e., the diamond pattern and the double-strip pattern. They compared the number of nodes required to achieve field coverage with four-connectivity using these patterns to that of an existing deployment pattern [6]. Other studies have used mobile nodes to cover the field automatically. Wang et al. proposed a field coverage method using mobile nodes that initially move only once to save energy by considering energy balance [7]. However, it is difficult to adapt such methods to changing environments. Eto et al. proposed a method to cover an agricultural field by moving mobile nodes dynamically [8]. Their method avoids battery depletion by predicting the amount of solar energy generation. However, that study targeted WSNs with low sensing frequency where field coverage is achieved by a set of mobile nodes visiting all points in the field at least once during duty cycle. Thus the field is not always covered.

Existing studies have attempted to solve the lifetime extension problem using EH or wireless charging, or the coverage problem by mobile sensor nodes. Therefore, to address gaps in field coverage and charge static node batteries simultaneously, a new method to determine the efficient loci of mobile sensor nodes is required.

We propose a method that determines the position of each mobile node in order to extend the lifetime of a WSN constructed with static nodes using wireless charging receivers and mobile nodes with wireless charging senders and solar panels.

III. ASSUMPTIONS AND PROBLEM AND FORMULATION

In this section, we describe our assumptions and the problem formulation.

A. Target WSNs

We target WSNs constructed with a single sink node and static and mobile sensor nodes. Here, a sink node is denoted B. Static and mobile sensor nodes have batteries and devices for sensing and radio communication. The capacity of the battery is denoted E. The residual battery power of node s at time t is denoted $s.energy(t)$. The sensing range and communication range are denoted R_s and R_c, respectively. Both types of sensor nodes sense environmental data every duty cycle I (a sink collects the environmental information from all sensor nodes every I minutes). We assume that a mobile sensor node can move to specified destination, charge its battery using an equipped solar panel and send electric energy to a static sensor node using the equipped wireless charging system's sending device. The speed of a mobile node is constant value v. Note that a static sensor node can charge using only its wireless charging receiving device. If the distance between the sending and receiving devices is less than d_c, a static node can charge its battery from a mobile node.

Here, a field with height F_{height} and width $Fwidth$ is given, in which N_{static} static sensor nodes are initially deployed. Then, N_{mobile} mobile sensor nodes are deployed to cover the field completely and charge the batteries of the static nodes. The position of sensor node s at time t is denoted $s.pos(t)$. The destination point of mobile node p at time t is denoted $s.dest(t)$. A sensor node can know its own position and the positions of other nodes using GPS and wireless communication.

B. Definitions

A sensor node has finite battery power, and it is not feasible to replace batteries physically when they are exhausted. Battery power is consumed by sensing, moving, wireless communication, and supplying electric power. On the other hand, battery power can be charged by solar power generation and the electric power supplied by a mobile node.

The power for consumed sensing x[bit] data $Sens(x)$ is expressed by Formula(1).

$$Sens(x) = E_{elec}x + E_{sens} \tag{1}$$

Here, E_{sens} and E_{elec} are constant values that represent the power required for sensing and information processing, respectively.

Consumed powers $Trans(x, d)$ and $Recep(x)$ required to transmit x[bit] for d[m] and receive x[bit] are expressed by Formulas (2) and (3), respectively [11].

$$Trans(x, d) = E_{elec} \times x + \varepsilon_{amp} \times x \times d^2 \tag{2}$$

$$Recep(x) = E_{elec} \times x \tag{3}$$

Here, ϵ_{amp} is a constant value that represents the power required for signal amplification.

Consumed power $Move(d)$ required to move d [m] is expressed by Formula (4) [16].

$$Move(d) = E_{move}d \tag{4}$$

Here, E_{move} is a constant value that represents the power required to move a node 1 [m].

Using the wireless charging system, mobile sensor node p consumes its energy to charge the battery of static node q. Consumed power $C_{send}(y)$ required to send the electric power for y [sec] is expressed by Formula (5).

$$C_{send}(y) = \epsilon_{wc}y \tag{5}$$

Here, ϵ_{wc} is a constant value that represents the energy consumption of the wireless charging system.

The energy of charged by the wireless charging system $C_{recep}(y)$for y [sec] is expressed by Formula (6).

$$C_{recep}(y) = \theta\epsilon_{wc}y \tag{6}$$

Here, θ represents the charging efficiency ($0 \leq \theta \leq 1$). If $\theta = 1$, the energy consumption of a mobile node equals the charged energy of a static node.

A mobile node can charge its battery by solar power generation. The amount of generated solar power depends on the intensity of solar radiation, which varies according to changing environmental conditions. We denote the intensity of solar radiation at night, during a cloudy day, and during a sunny day as c_{night}, c_{cloudy}, and c_{sunny}, respectively. The amount of solar power generated at an average solar radiation intensity $c([t, t+u])$ from time t to $t+u$ is expressed by Formula 7 [8].

$$C_{solar}(c([t, t+u])) = \epsilon_{solar}uc([t, t+u]) \tag{7}$$

Here, ϵ_{solar} is the charged energy amount per unit time coefficient. c([t, t+u]) is expressed by Formula (8).

$$c([t, t+u]) = \frac{ic_{night} + jc_{cloudy} + kc_{sunny}}{i+j+k} \tag{8}$$

Here, i, j, and k are the rates of night, cloudy, and sunny periods during $[t, t+u]$, respectively.

C. Problem formulation

The inputs to the target problem are target field ($F_{width} \times F_{height}$), the positions of the sink node and static nodes, the initial battery power of each sensor node, sensing range, and constants E_{sens}, E_{elec}, ϵ_{amp}, ϵ_{wc}, and ϵ_{solar}. The outputs are the position of each mobile node for each time t and the node pairs for battery power supply at each time t. We refer to these node positions as the node schedule.

Note that the field must always be covered by sensor nodes. This condition is expressed by Formula (9).

$$\forall pos \in Field, \forall t \in Time, cover(pos, t) \geq 1 \tag{9}$$

Here, $Cover(pos, t)$ is the number of nodes covering the point pos at time t. $Time$ and $Field$ represent set of the WSN termination time and set of the field, respectively.

If node s with position $s.pos(t)$ at time t moves to s's destination point $s.dest(t)$, the residual battery power of s should be greater than the energy consumed by moving. This constraint is expressed by Formula (10).

$$\forall t \in Time, s.energy(t) - Move(|s.dest(t) - s.pos(t)|) + C_{solar}(c([t, t + \frac{|s.dest(t) - s.pos(t)|}{v}])) > 0 \tag{10}$$

Here,$|s.dest(t) - s.pos(t)|$ represent the moving distance.

If mobile node p charges the battery power of static node q at time t for u_t seconds, the residual battery power of p should be greater than the energy consumed for charging.

$$\forall t \in Time, s.energy(t) - C_{send}(u_t) + C_{solar}(c([t, t+u_t]) > 0 \tag{11}$$

Our purpose is to determine a node schedule that maximizes WSN operation time T. The objective function is shown in Formula (12).

$$maximize(T) \tag{12}$$

IV. Proposed method

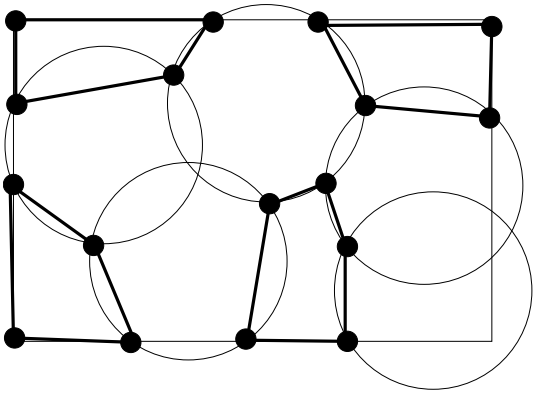
In this section, we describe the proposed method to solve the target problem explained in Section III. The proposed method is comprised of two algorithms, i.e., the coverage and movement algorithms.

The coverage algorithm finds uncovered areas and determines the positions of nodes such that the uncovered areas covered filled by a small number of mobile nodes. The movement algorithm periodically determines the movement schedule of the mobile nodes.

A. Coverage algorithm

First, the polygon-finding algorithm included in the coverage algorithm finds all areas that are not covered by static nodes. Next, the coverage algorithm determines efficient positions for the mobile nodes that will cover the uncovered areas. Finally, the mobile nodes are deployed to each position.

To find each **uncovered polygon** that inscribes an uncovered area, the polygon-finding algorithm finds all **uncovered intersections** as shown in Fig. 1. An uncovered intersection is an intersection that is not covered by sensor nodes and is made up of two elements, i.e., edges of the target field and sensing range circles of the sensor nodes. Note that an uncovered polygon, whose vertices are uncovered intersections, exists if and only if an uncovered area exists.



Circles are the sensing region of static nodes.
Black dots are uncovered intersections.
Uncovered polygons whose vertices are only
uncovered intersections are drawn by bold lines.

Fig. 1. Ucovered polygons and intersections

The coverage algorithm minimizes uncovered polygons by placing nodes at appropriate positions in the polygons sequentially until the polygons no longer exist. Here, we denote a set of uncovered polygon vertices found by the polygon-finding algorithm as $Q = \{q_0, q_1, q_2, \cdots\}$, which q_i and q_{i+1} are adjoined vertices. Two sensor nodes a and b that construct vertex q_i is denoted by $q_i.s_1$ and $q_i.s_2$, respectively. Note that $q_i.s_2$ and $q_{i+1}.s_1$ represent the same sensor node b. Similarly, $q_{i-1}.s_2$ and $q_i.s_1$ represent the same sensor node a. If the distance between q_i and q_{i+2} is within diameter $2R_s$ of the sensing range circle, the candidate position of a node that covers q_i, q_{i+1}, and q_{i+2} exists. This position is the circumcenter of the triangle whose vertices are q_i, q_{i+1}, and q_{i+2}. If no such node position includes these three vertices, the candidate position is the center of the circle that inscribes q_i and q_{i+1}. The coverage algorithm places the mobile node at one of these candidate positions such that the area that is newly covered by the mobile node is maximized. Here, if the mobile node placed at the candidate position divides the uncovered polygon into two or more polygons, such a position is eliminated as a candidate. Although one of destination position of optimal deployment divides the uncovered polygon into two or more polygons, our algorithm puts the node at the position when the destination position no longer divides the uncovered polygon. In order to simplify the problem, our algorithm makes the outer side destination have priority.

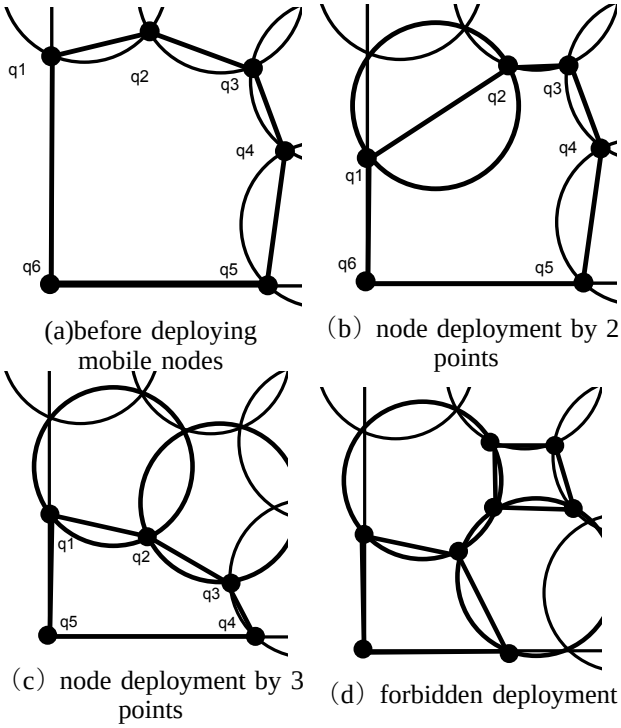


Fig. 2. A method to cover polygon Q

For example, uncovered polygon Q is constructed by uncovered intersections q_1, q_2, q_3, q_4, q_5, and q_6 as shown in Fig. 2 (a). The mobile node is placed at the center of the circle that inscribes q_1 and q_2, because the area whose size is newly covered by the mobile node is maximized as shown in Fig. 2 (b). Then, the distance between q_2 and q_4 in Fig. 2 (b) is less than $2R_s$, and the mobile node is placed at the circumcenter of the triangle whose vertices are q_2, q_3, and q_4 (Fig. 2 (c)). On the other hand, the mobile node is not placed at a position such that the uncovered polygon becomes divided as shown in Fig. 2 (d).

1) Coverage algorithm: The output of the coverage algorithm is a set of node positions P. Here, Q is an uncovered polygon. $Q.m$ is the number of vertices of Q. q_i is the i-th element of Q, and $dist(q_i, q_j)$ is the distance between q_i and q_j.

1) Find uncovered polygon Q by the polygon-finding algorithm. If no polygon is found, the algorithm terminates.
2) If m is 0, return to Step 1.
3) Find i such that $dist(q_i,\ q_{i+2}) < 2R_s$; otherwise proceed to Step 4. A mobile node is placed at the circumcenter of the triangle whose vertices are q_i, q_{i+1}, and q_{i+2} such that the area that is newly covered is maximized. The coordinate of this mobile node is added to P. Proceed to Step 6.
4) Find i such that $dist(q_i, q_{i+1}) < 2R_s$, otherwise proceed to Step 5. A mobile node is placed at the center of the circle that inscribes q_i and q_{i+1} such that the area that is newly covered is maximized. The coordinate of this mobile node is added to P. Proceed

to Step 6.

5) A mobile node is placed such that the sensing region of the node includes an element of Q and the area that is newly covered is maximized. The coordinate of this mobile node is added to P. Proceed to Step 6.
6) The vertices newly covered by a mobile node are eliminated from Q. The uncovered vertices newly created by a mobile node are added to Q. Return to Step 2.

2) Polygon-finding algorithm: The polygon-finding algorithm finds the polygons that are created by the uncovered intersections. The polygon-finding algorithm outputs the uncovered polygon Q. Here, P is a set of uncovered intersections.

1) $i = 0$. If $P = \emptyset$, the algorithm terminates as no polygon is found.
2) If $q_i.s2$ is an edge of the field, proceed to Step 4, else go to Step 3.
3) Investigate the element of P to way that is not covered by $q_i.s1$ in the sensing range circle. The point that is found first is q_{i+1}. Proceed to Step 5.
4) Investigate the element of P to way that is not covered by $q_i.s1$ on the edge of the field. The point that is detected first is q_{i+1}.
5) If $q_{i+1}.s1$ is not $q_i.s2$, exchange $q_{i+1}.s1$ with $q_{i+1}.s2$.
6) If q_{i+1} is p_0, proceed to Step 7, else $i = i + 1$ and return to Step 2.
7) All followed points except q_{i+1} are added to Q. The algorithm terminates with $P = P - Q$.

B. Movement algorithm

The movement algorithm allows field coverage to be maintained for long periods. We set two modes for each mobile node, i.e., a coverage mode and a free mode. A coverage mode mobile node moves to the candidate position calculated by the coverage algorithm. Note the number of coverage mode nodes is fixed. The other nodes operate in free mode. If the residual battery power of a coverage mode node becomes low, the free mode node moves to the position of the coverage mode node and becomes a coverage mode node. Note that free mode nodes do not perform sensing.

In the movement algorithm, nodes communicate residual battery power and power-generation efficiency information to each other, and the sink node forecasts node s whose residual battery power will be depleted rapidly. If s is a free mode node, s charges its battery by solar power at its current position. If s is a static node or a coverage mode node, it seeks another mobile node n_1 with sufficient battery power. If mobile node n_1 is a free mode node, then it moves to the position of node s and performs wireless charging or changes n_1's mode to coverage and s's mode to free: otherwise, s seeks the nearest free mode node n_2. E^c_{n1} and E^c_{n2} are the estimated residual battery power of n_{n1} and n_{n2} when n_2 move to the position of n_1 and n_{n1} moves to the position of s (this type of movement is referred to as **cascade movement** [17]. E^d_{n1} and E^d_{n2} are the estimated residual battery power of n_{n1} and n_{n2} when n_2 moves directly to the position of s. The movement algorithm compares the minimum values of the estimated residual battery power E^c_{n1} and E^c_{n2} to E^d_1 and E^d_2 by these two movement types and selects the movement type with greater estimated residual battery power.

The movement algorithm runs when a node whose residual battery power is less than $\frac{L}{l}$ exists. Here, the average residual battery power among all nodes is denoted L.

1) Movement algorithm:

1) If node s whose residual battery power is a less than $\frac{L}{l}$ is free mode node, then s's battery is charged by solar energy generation and the algorithm terminates, else seek the nearest mobile node n_1 whose battery power is greater than L.
2) If n_1 is a free mode node, then move n_1 to the position of s and proceed to Step 5, else seek mobile node n_2 that has more than L battery power and is the nearest node from s.
3) Estimate the residual battery power E^d_{n1} and E^d_{n2} by moving n_2 to the position of s directly, and estimate E^c_1 and E^c_2 by moving n_2 to n_1's position and n_1 to the position of s (cascade movement).
4) Select the movement of the greater minimum value of E^c_1 and E^c_2, or E^d_1 and E^d_2. Mobile nodes are then moved.
5) If node s is a static node, the moved mobile node performs wireless charging, else s and the moved mobile node exchange modes.

V. Performance evaluation

Here, we describe simulation results of an evaluation of the proposed method.

We performed computer simulations to measure the field coverage time that can be maintained by varying the number of sensor nodes for two patterns of static sensor node deployment. In the first scenario, static sensor nodes are scattered by a helicopter, i.e., they are deployed randomly. The mobile sensor nodes primarily move to fill gaps in field coverage. In the second scenario, static sensor nodes whose batteries have less than one-half charge are deployed in a honeycomb structure. In this scenario, the deployment is calculated and efficient, and the WSN is used for a long period. Mobile sensor nodes primarily move to charge static sensor node batteries and repair coverage holes.

To evaluate the effects of the proposed features, we compared the proposed method with a method whose battery charging mechanism has not been validated (hereafter, simple method). The common parameter settings in our simulations are shown in Table. I. The network topology is a mesh rooted at the sink node. Each sensor node sends data to the upper node that has maximum residual battery power.

A. Settings

B. Coverage algorithm evaluation

In this simulation, n static nodes were deployed initially, and the number of mobile sensor nodes required for complete field coverage was measured.

The average results of 10 trials are shown in Figure. 3. We confirmed that the number of mobile nodes decreases as

TABLE I. COMMON PARAMETER SETTINGS IN THE SIMULATIONS

Size of target field	100 × 100 $[m^2]$
Sensing radius R_s	20 [m]
Duty cycle I	15 [min]
Data size	256 [bit]
Battery capacity E	8640 [J]
Power consumption coefficient for data processing E_{elec}	50 [nJ/bit]
Power consumption coefficient for signal amplification ϵ_{amp}	100 [pJ/bit/m^2]
Power consumption coefficient for sensing E_{sens}	0.018 [J/bit]
Power consumption for moving	1680 [mW]
Efficiency of wireless charging θ	0.8

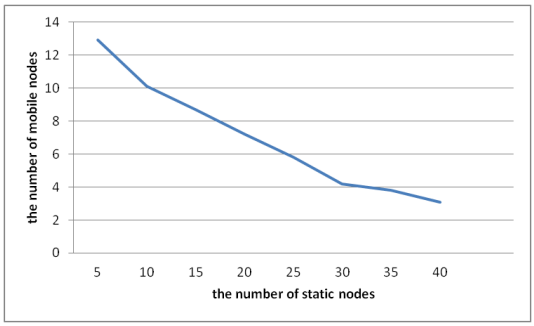


Fig. 3. The number of nodes needed by the coverage algorithm

the number of static sensor nodes increases. The coverage algorithm required 18 sensor nodes (five randomly deployed static nodes and 13 mobile nodes) to cover the 100 × 100 [m^2] field. Note that a honeycomb structure required 14 nodes to cover the same field. The coverage algorithm can cover the field with a small number of nodes.

C. Evaluation of the proposed method

In this simulation, we measured the lifetime of the WSN lby changing the number of static or mobile sensor nodes. WSN operation was terminated when battery power of at least one node was depleted.

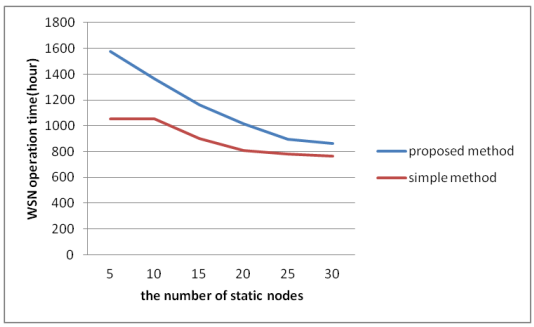


Fig. 4. WSN lifetime by fixing 50 mobile nodes

1) Changing the number of static nodes: Figure. 4 shows the WSN lifetime with 50 mobile nodes and various numbers of static nodes. We compared the proposed method with the simple method. We confirmed that the proposed method achieves longer WSN lifetime than the simple method. The energy charging system ' s moving schedule is effective to extend WSN lifetime. We also confirmed that WSN lifetime is reduced when the number of static nodes that periodically sense data is increased because the amount of energy required for data communication increases. According to Formulas (2) and (3), energy consumption increases as the amount of transmitted data increases.

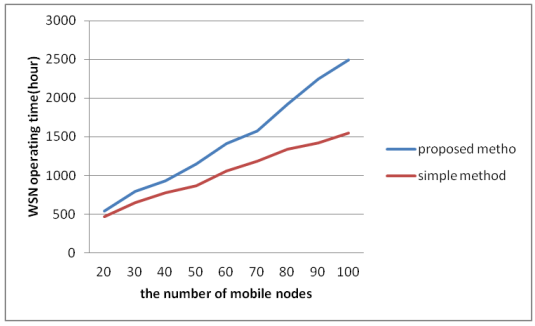


Fig. 5. WSN lifetime with 15 static nodes

2) Changing the number of mobile nodes: Figure. 4 shows the WSN lifetime with 15 static nodes and various numbers of mobile nodes. We confirmed that WSN lifetime is extended by increasing the number of mobile nodes. We confirmed that the proposed method achieves 66% longer WSN lifetime than the simple method with 100 mobile nodes. However, the difference between the proposed method and the simple method with 20 mobile nodes was not significant because the total energy generation of 100 mobile nodes is grater than 20 mobile nodes. WSN lifetime is extended with increased total energy generation.

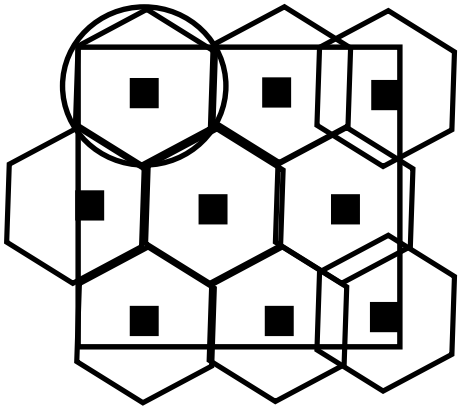


Fig. 6. Honeycomb structure

3) Honeycomb structure deployment: WSNs are sometimes required to operate according to planned node deployment. In this simulation, each static sensor node was deployed as the center point of a hexagon inscribed in a circle whose radius is R_s, as shown in Fig. 6. The honeycomb structure shown in Fig. 6 is the most efficient deployment [12]. The honeycomb structure requires 14 static nodes to cover a 100 × 100 [m^2] field, as described in Subsection V-B.

We added mobile sensor nodes to WSNs constructed with only static sensor nodes that have already consumed one-half and one-quarter if their battery power on average. The durations from initial deployment of the static nodes until one-half and one-quarter battery power was consumed was 189.4

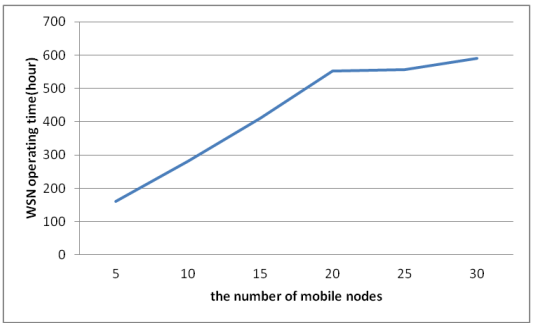


Fig. 7. Honeycomb structure (1/2 case)

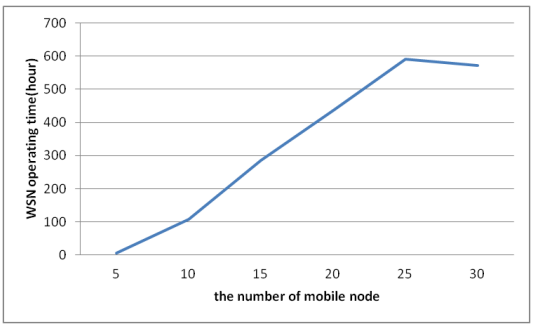


Fig. 8. Honeycomb structure (1/4 case)

[h] and 298.7 [h], respectively. Here, we describe these two cases (i.e., one-half and one-quarter battery consumption). For each case, we measured the lifetime extension achieved by adding mobile nodes.

Figures 7 and 8 show the one-half and one-quarter cases, respectively. As can be seen in both figures, the proposed method extends WSN lifetime linearly relative to the number of mobile nodes. However, the degree of lifetime extension is gradual around 550–600 [h]. With 30 mobile nodes, Figs. 5 and 7 show 793.2 [h] and 591.5 [h] lifetime extensions. In the one-half case, the WSN lifetime is $591.5 + 189.4 = 780.9$. Although no mobile node was used for 189.4 [h], the lifetime of the one-half case was similar to the randomly deployed case. In Fig. 8, the lifetime of 30 mobile nodes is less than 25 mobile nodes. The reason is the deviation of initial energy amount. Our simulations conducted 10 times were not able to eliminate the affect of unfortunate initial parameters. Thus, we confirmed that the proposed method with a honeycomb structure can extend WSN lifetime.

VI. Conclusion

We have proposed a method to extend WSN lifetime using two energy charging systems. The proposed algorithm includes a coverage algorithm and movement algorithm. The coverage algorithm determines the positions of mobile nodes that cover areas not covered by static nodes. The movement algorithm determines the destination of nodes whose batteries need to be charged by mobile nodes. Our simulation results, confirm that the proposed method with a honeycomb structure deployment can extend WSN lifetime. In future, we plan to decentralize the proposed algorithm for scalability.

References

[1] Th.Aramapatzis, J.Lygeros, and S. Manesis, : "A Survey of Applications of Wireless Sensors and Wireless Sensor Networks," *Proceedings of International Symposium on, Mediterrean Conference on Control and Automation (ISIC/MED 2005), pp. 719- 724 (June 2005).*

[2] A. Mainwaring, D. Culler, J.Polastre, R. Szewczyk, and J.Anderson; "Wireless sensor networks for habitat monitoring ," *Proceedings of the 1st ACM International Workshop pn Wireless Sensor Networks and Applications, pp.88-97 (2002).*

[3] HIDEKI AYANO, HIROSHI NAGASE, HIROMI INABA, : "A Highly Efficient Contactless Electrical Energy Transmission System," *Electrical Engineering in Japan, Vol 148, No.1, 2004.*

[4] Davide Brunelli, Luca Benini, Clemens Moster, and Lothar Thiele, : "An Efficient Solar Energy Harvester for Wireless Sensor Nodes," *Proceedings of the conference on Design, automation and test in Europe pp 104-109 (2008)*

[5] Yang Peng, Zi Li, Wensheng Zhang, and Daiji Qiao, : "Prolonging Sensor Network Lifetime Through Wireless Charging," *IEEE Real-Time Systems Symposium (RTSS), San Diego, CA, November 30 - December 3, 2010.*

[6] Xiaole Bai, Ziqiu Yun, Dong Xuan, Ten H. Lai, and Weijia Jia, : "Optimal Patterns for Four-Connectivity and Full Coverage in Wireless Sensor Networks," *Mobile Computing, IEEE Transactions on (Volume:9 , Issue: 3) pp. 435-448, March 2010*

[7] You-Chuin Wang, Wen-Chium Peng, and Yu-Chee Tseng, : "Energy-Balanced Dispatch of Mobile Sensor in a Hybrid Wireless Sensor Network," *IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 12, pp. 1836-1850, Dec. 2010*

[8] Masaru Eto, Ryo Katsuma, Morihiko Tamai, and Keiichi Yasumoto: "Efficient Coverage of Agricultural Field with Mobile Sensors by Predicting Solar Power Generation," *Proc. of IEEE Int'l. Conf. on The 29th IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 62–69, 2015.*

[9] M. Cardei, J. Wu, M. Liu, and M. Pervaiz: "Maximum network lifetime in wireless sensor networks with adjustable sensing ranges," *Proc. of IEEE Int'l. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob), 2005.*

[10] Jiong Wang, Medidi, S., Medidi, M.: "Energy-Efficient k-Coverage for Wireless Sensor Networks with Variable Sensing Radii ," *Proc. of IEEE Int'l. Conf. on Global Telecommunications Conference (GLOBECOM), pp. 1–6, 2009.*

[11] Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H. "Energy-efficient communication protocol for wireless microsensor networks," *Proc. of the 33rd Hawaii Int'l. Conf. on System Sciences (HICSS 2000)* , pp.1–10, 2000.

[12] Xiaole Bai, Ziqiu Yun, Dong Xuan, Biao Chen and Wei Zhao: "Optimal Multiple-Coverage of Sensor Networks," *Proc. of IEEE Int'l. Conf. on INFOCOM, pp. 2498–2506, 2011.*

[13] Luo, R.C., and Chen, O.: "Mobile Sensor Node Deployment and Asynchronous Power Management for Wireless Sensor Networks," *IEEE Trans. on Industrial Electronics, No. 59, Vol. 5, pp. 2377–2385, 2011.*

[14] Wang, Y. C. and Tseng, Y. C.: "Distributed Deployment Schemes for Mobile Wireless Sensor Networks to Ensure Multi-level Coverage," *IEEE Trans. on Parallel and Distributed Systems, No. 19, Vol. 9, pp. 1280–1294, 2007.*

[15] Ouadou, Mourad, Zytoune, Ouadoudi, and Aboutajdine, Driss: "Wireless charging using mobile robot for lifetime prolongation in sensor networks," *Proc. of IEEE Complex Systems (WCCS), pp. 225–230, 2014.*

[16] Rahimi, M., Shah, H., Sukhatme, G.S., Heideman, J., Estrin, D. "Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network," *Proc. of the IEEE Int'l. Conf. on Robotics and Automation (ICRA)*, pp.19–24, 2003.

[17] You-Chiun Wang, Fang-Jing Wu and Yu-Chee Tseng: "Mobility management algorithms and applications for mobile sensor networks," *J. of Wireless Communications and Mobile Computing*, Vol. 12, Issue 1, pp.7–21, 2012.

Performance Evaluation of Golub-Kahan-Lanczos Algorithm with Reorthogonalization by Classical Gram-Schmidt Algorithm and OpenMP

Masami Takata[1], Hiroyuki Ishigami[2], Kinji Kimura[2], Yuki Fujii[2], Hiroki Tanaka[2], and Yoshimasa Nakamura[2]
[1]Research Group of Information and Communication Technology for Life, Nara Women's University, Nara, Nara, JAPAN
[2]Graduate School of Informatics, Kyoto University, Kyoto, Kyoto, JAPAN

Abstract— *The Golub-Kahan-Lanczos algorithm with reorthogonalization (GKLR algorithm) is an algorithm for computing a subset of singular triplets for large-scale sparse matrices. The reorthogonalization tends to become a bottleneck of elapsed time, as the iteration number of the GKLR algorithm increases. In this paper, OpenMP-based parallel implementation of the classical Gram-Schmidt algorithm with reorthogonalization (OMP-CGS2 algorithm) is introduced. The OMP-CGS2 algorithm has the advantage of data reusability and is expected to achieve higher performance of the reorthogonalization computations on shared-memory multi-core processors with large caches than the conventional reorthogonalization algorithms. Numerical experiments on shared-memory multi-core processors show that the OMP-CGS2 algorithm accelerates the GKLR algorithm more effectively for computing a subset of singular triplets for a sparse matrix than the conventional reorthogonalization algorithms.*

Keywords: Subset computation of singular triplets, Golub-Kahan-Lanczos algorithm with reorthogonalization, Classical Gram-Schmidt algorithm with reorthogonalization, OpenMP, Shared-memory multi-core processing

1. Introduction

Let A be a real $m \times n$ matrix and $\mathrm{rank}(A) = r$ ($r \leq \min(m,\ n)$). Then A has the r singular values $\sigma_1,\ \ldots,\ \sigma_r \in \mathbb{R}$, which satisfies $\sigma_1 \geq \cdots \geq \sigma_r > 0$, and their corresponding left and right singular vectors $\boldsymbol{u}_i \in \mathbb{R}^m$, $\boldsymbol{v}_i \in \mathbb{R}^n$ ($1 \leq i \leq r$). A subset of singular triplets, i.e. the l largest singular values $\sigma_1,\ \ldots,\ \sigma_l$ and their corresponding singular vectors, is often required in low-rank matrix approximation [17] and statistical processings such as principal component analysis and the least-squares method. In such applications, the target matrix is often large and sparse, and l is often much smaller than both m and n. It is difficult to perform the computation of singular triplets directly from a large-scale sparse matrix because of the computational cost and need for large amounts of memory.

The Krylov subspace methods are better for such computations. They transform the target matrix into a significantly smaller matrix than the target matrix and the singular values of the generated matrix sufficiently approximate a subset of singular values of the target matrix. The Golub-Kahan-Lanczos (GKL) algorithm [5], [6] is one of the Krylov subspace methods and generates approximate bidiagonal matrices from the target matrix. However, the GKL algorithm usually loses the orthogonality of the Krylov subspace because of the computational error. To improve the orthogonality, let us incorporate a reorthogonalization process into the GKL algorithm. Such an algorithm is referred to as the GKL algorithm with reorthogonalization (GKLR algorithm) [1]. Note that these algorithms are generally parallelized in terms of the Basic Linear Algebra Subprograms (BLAS) [12], such as the matrix multiplications and the matrix-vector multiplications, because they are iterative algorithms. In addition, we implement the bisection algorithm and the inverse iteration algorithm [10], [8] for computing a subset singular triplets of the approximate matrices generated by the GKLR algorithm.

Although the GKLR algorithm is stable because of the reorthogonalization, the reorthogonalization tends to become a bottleneck in terms of the computational cost and the elapsed time as the iteration number increases. However, since the reorthogonalization of the GKLR algorithm is mainly implemented using the matrix-vector multiplications, even in parallel computing, the reorthogonalization is not effectively accelerated and then the overall elapsed time of the GKLR algorithm is not effectively reduced.

In this paper, to accelerate the reorthogonalization of the GKLR algorithm more effectively in parallel computing, we introduce a parallel implementation of the classical Gram-Schmidt algorithm with reorthogonalization (CGS2 algorithm) [3], which is parallelized using the OpenMP [13]. Hereafter, this implementation of the CGS2 algorithm is referred to as the OMP-CGS2 algorithm. This parallelization technique enables to use the cache of CPUs effectively and then the computation is expected to be accelerated more effectively than the conventional reorthogonalization

algorithms, which are parallelized in terms of the BLAS operations.

The rest of this paper is organized as follows. In Section 2, the GKLR algorithm and its implementation in this paper are described. In Section 3, a BLAS-based parallel implementation of reorthogonalization algorithms and the OMP-CGS2 algorithm are presented. Section 4 provides performance evaluations of the OMP-CGS2 algorithm on multi-core processors. We end with conclusions and future works in Section 5.

2. GKLR algorithm

This section considers the GKLR algorithm and describes the implementation of the GKLR algorithm in this paper.

2.1 GKLR algorithm

The GKL [5], [6] algorithm generates new bases $\boldsymbol{p}_k \in \mathbb{R}^n$ and $\boldsymbol{q}_k \in \mathbb{R}^m$ at the k-th iteration. The $\boldsymbol{p}_k$ is an orthonormal basis of the Krylov subspace $\mathcal{K}(A^\top A,\ \boldsymbol{p}_1,\ k)$, and the $\boldsymbol{q}_k$ is an orthonormal basis of the alternative Krylov subspace $\mathcal{K}(AA^\top, A\boldsymbol{p}_1,\ k)$. In the GKLR algorithm [1], each time a new basis is added with the expansion of the Krylov subspace, the existing orthonormal basis, and the new basis are reorthogonalized.

Algorithm 1 shows the pseudocode of the GKLR algorithm. Lines 6 and 10 show the reorthogonalization process, respectively. At the beginning of the k-th iteration for $k = 1,\ 2,\ \ldots$ in Algorithm 1, the $k \times k$ approximate matrices

$$B_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & \alpha_k \end{bmatrix} \tag{1}$$

are obtained and the following equations hold

$$AP_k = Q_k B_k, \tag{2}$$

$$A^\top Q_k = P_k B_k^\top + \beta_k \boldsymbol{p}_{k+1} \boldsymbol{e}_k^\top, \tag{3}$$

where $\boldsymbol{e}_k$ is the k-th column of the $k \times k$ identity matrix. Note that if the l largest singular values of B_k sufficiently approximate those of A, we can stop the iterations of the GKLR algorithm. On line 8 in Algorithm 1 , we check whether the l largest singular values of B_k sufficiently approximate those of A or not. Criteria for this check are discussed in Sec. 2.2.1.

Let $\sigma_j^{(k)}$, $\boldsymbol{s}_j^{(k)} \in \mathbb{R}^k$, and $\boldsymbol{t}_j^{(k)} \in \mathbb{R}^k$ $(j = 1,\ \ldots,\ k)$ be a singular value of B_k, the left singular vector, and the right singular vector corresponding to $\sigma_j^{(k)}$, respectively. If $\sigma_j^{(k)}$ approximates σ_j well, then $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ corresponds to $\boldsymbol{u}_j^{(k)}$ and $\boldsymbol{v}_j^{(k)}$ defined as the following equations, respectively:

$$\boldsymbol{u}_j^{(k)} = Q_k \boldsymbol{s}_j^{(k)},\ \boldsymbol{v}_j^{(k)} = P_k \boldsymbol{t}_j^{(k)}. \tag{4}$$

Algorithm 1 GKLR algorithm

1: Set an n-dimensional unit vector $\boldsymbol{p}_1$
2: $\boldsymbol{q} = A\boldsymbol{p}_1,\ \alpha_1 = \|\boldsymbol{q}\|_2,\ \boldsymbol{q}_1 = \boldsymbol{q}/\alpha_1$
3: $P_1 = [\boldsymbol{p}_1], Q_1 = [\boldsymbol{q}_1]$
4: **do** $k = 1,\ 2,\ \ldots$
5: $\quad \boldsymbol{p} = A^\top \boldsymbol{q}_k$
6: $\quad \tilde{\boldsymbol{p}} = \mathrm{Reorthogonalization}(P_k,\ \boldsymbol{p})$
7: $\quad \beta_k = \pm\|\tilde{\boldsymbol{p}}\|_2,\ \boldsymbol{p}_{k+1} = \tilde{\boldsymbol{p}}/\beta_k$
8: $\quad$ Check the singular values of B_k
9: $\quad \boldsymbol{q} = A\boldsymbol{p}_{k+1}$
10: $\quad \tilde{\boldsymbol{q}} = \mathrm{Reorthogonalization}(Q_k,\ \boldsymbol{q})$
11: $\quad \alpha_{k+1} = \pm\|\tilde{\boldsymbol{q}}\|_2,\ \boldsymbol{q}_{k+1} = \tilde{\boldsymbol{q}}/\alpha_{k+1}$
12: $\quad P_{k+1} = \begin{bmatrix} P_k & \boldsymbol{p}_{k+1} \end{bmatrix},\ Q_{k+1} = \begin{bmatrix} Q_k & \boldsymbol{q}_{k+1} \end{bmatrix}$
13: **end do**

In order to improve the accuracy of singular vectors, this computation is implemented to the combination with the QR factorization [11].

As seen in Algorithm 1, the GKLR algorithm must be parallelized in terms of the computations on each line. Since the computation on each line can be implemented using the BLAS operations, we parallelize the GKLR algorithm in terms of each the BLAS operations.

2.2 Implementation of GKLR algorithm

In this section, we discuss the methods to check whether the singular values of B_k approximate sufficiently those of A or not. We then introduce a stopping strategy of the GKLR algorithm and the implementation for the subset computation of singular triplets for approximate matrices in this paper.

2.2.1 Stopping strategy of GKLR algorithm

Recalling $(\sigma_j^{(k)},\ \boldsymbol{s}_j^{(k)}\ \boldsymbol{t}_j^{(k)})$, the j-th singular triplets for B_k $(j = 1,\ \ldots,\ l)$, we then have the following equations:

$$B_k \boldsymbol{t}_j^{(k)} = \sigma_j^{(k)} \boldsymbol{s}_j^{(k)},\ B_k^\top \boldsymbol{s}_j^{(k)} = \sigma_j^{(k)} \boldsymbol{t}_j^{(k)}. \tag{5}$$

Using Eqs. (2), (3), (4), and (5), we obtain

$$\begin{aligned} A^\top \boldsymbol{u}_j^{(k)} - \sigma_j^{(k)} \boldsymbol{v}_j^{(k)} &= A^\top Q_k \boldsymbol{s}_j^{(k)} - \sigma_j^{(k)} P_k \boldsymbol{t}_j^{(k)} \\ &= \left(A^\top Q_k - P_k B_k^\top\right) \boldsymbol{s}_j^{(k)} \\ &= \beta_k \boldsymbol{p}_{k+1} \boldsymbol{e}_k^\top \boldsymbol{s}_j^{(k)} \\ &= \beta_k s_j^{(k)}(k) \boldsymbol{p}_{k+1}, \end{aligned} \tag{6}$$

where $s_j^{(k)}(k)$ is the k-th element of $\boldsymbol{s}_j^{(k)}$. Thus, the following inequality holds:

$$\left\| A^\top \boldsymbol{u}_j^{(k)} - \sigma_j^{(k)} \boldsymbol{v}_j^{(k)} \right\|_2 \leq \left| \beta_k s_j^{(k)}(k) \right|. \tag{7}$$

As the results, if the right-hand side of inequality (7) is sufficiently small, then the singular value $\sigma_j^{(k)}$ of B_k can be regarded to sufficiently approximate that of A. Hence, the

Algorithm 2 Stopping strategy of GKLR algorithm

1: Compute $(\sigma_l^{(k)},\ \boldsymbol{s}_l^{(k)},\ \boldsymbol{t}_l^{(k)})$
2: **if** $\left|\beta_k s_l^{(k)}(k)\right| \leq \delta$, **then**
3: Compute $(\sigma_j^{(k)},\ \boldsymbol{s}_j^{(k)},\ \boldsymbol{t}_j^{(k)})$ for $j = 1,\ \ldots,\ l$
4: **if** $\left|\beta_k s_j^{(k)}(k)\right| \leq \delta$ for $j = 1,\ \ldots,\ l$, **then**
5: Stop the iteration of GKLR algorithm
6: **end if**
7: **end if**

following inequality can be considered as one of the stopping criteria of the GKLR algorithm:

$$\left|\beta_k s_j^{(k)}(k)\right| \leq \delta, \quad j = 1,\ \ldots,\ l, \tag{8}$$

where δ is a threshold value for stopping the iteration of the GKLR algorithm and determined arbitrarily by users. If we use this criterion based on inequality (8), we have to compute the l singular triplets for B_k, i.e. $(\sigma_j^{(k)},\ \boldsymbol{s}_j^{(k)},\ \boldsymbol{t}_j^{(k)}),\ j = 1,\ \ldots,\ l$, before checking if inequality (8) is satisfied. The computational cost of computing singular triplets for B_k is more expensive than that of checking if inequality (8). In order to reduce the total elapsed time for the GKLR algorithm, the computational cost of computing singular triplets for B_k has to be reduced. Hereafter, let k_{t} be the number of iterations where inequality (8) is satisfied for the first time.

Now let us consider the following inequality, which is one of the necessary conditions for inequality (8):

$$\left|\beta_k s_l^{(k)}(k)\right| \leq \delta. \tag{9}$$

We have only to compute the l-th largest singular triplet for B_k in order to check if inequality (9) is satisfied. Hence, from the viewpoint of the computational cost, inequality (9) is more suitable for the stopping criterion of the GKLR algorithm than inequality (8). In addition, if let k_{n} be the number of iterations where satisfy inequality (9) for the first time, it is observed that $k_{\mathrm{t}} = k_{\mathrm{n}}$ in many cases of numerical experiments. From these facts, inequality (9) can be also considered as one of the stopping criteria of the GKLR algorithm. However, since the theorems in [14] imply that the value of k_{t} depends on the distribution of singular values for the target matrix, $k_{\mathrm{t}} = k_{\mathrm{n}}$ is not always guaranteed. Thus, even if inequality (9) is satisfied, we must check if inequality (8) is also satisfied for all j.

Summarizing the above discussions, the stopping strategy for the GKLR algorithm is shown by Algorithm 2. In the experiments mentioned in Sec. 4, we set $\delta = 1.0 \times 10^{-14}$ as the stopping criterion. Note that Algorithm 2 is used on line 8 in Algorithm 1. After stopping the iteration of the GKLR algorithm, we compute the l largest singular triplets of A, i.e. $(\sigma_j,\ \boldsymbol{u}_j,\ \boldsymbol{v}_j)$ for $j = 1,\ \ldots,\ l$, using Eqs. (4).

2.2.2 Subset computation algorithms for singular triplets of approximate matrices

As mentioned in Section 2.2.1, a subset of singular triplets for the approximate matrices is required for stopping the GKLR algorithm. In this subsection, we discuss the subset computations of singular triplets of the approximate matrices on lines 1 and 3 in Algorithm 2.

The approximate matrix B_k, generated by the GKLR algorithm, is a lower bidiagonal matrix. As mentioned in [5], the singular value problem of the bidiagonal matrix can be transformed into the eigenvalue problem of the symmetric tridiagonal matrix without any computational cost. From the above fact, the singular triplets of the lower bidiagonal matrix can be obtained using the bisection algorithm and the inverse iteration algorithm (BI algorithm) for symmetric tridiagonal matrices [10], [8]. The BI algorithm enables us to compute only the required eigenpairs and is suitable for the subset computation of singular triplets in Algorithm 2. While computing l singular triplets (line 3 in Algorithm 2), we parallelize the subset computation of singular triplets as follows: The bisection algorithm is parallelized in terms of each singular value, and the inverse iteration algorithm is parallelized in terms of the BLAS operations.

3. Reorthogonalization algorithms

To improve the orthogonality of the Krylov subspace and the accuracy of the resulting singular vectors, the reorthogonalization is inevitable for the GKLR. However, the computational cost of the reorthogonalization is larger than the other processes of the GKLR, as the iteration number increases. Thus, it is important to accelerate the reorthogonalization in the GKLR.

In this section, at first, we consider three conventional reorthogonalization algorithms for the GKLR algorithm. The classical Gram-Schmidt with reorthogonalization (CGS2) algorithm [3], the modified Gram-Schmidt (MGS) algorithm [6], and the reorthogonalization algorithm using the Householder transformations in terms of the compact WY representation (cWY algorithm) [19], [9]. These algorithms are parallelized in terms of the BLAS operations in recent days. Secondly, we present the OpenMP-based parallel implementation of the CGS2 algorithm for shared-memory multi-core processors and describe the advantage of this implementation with respect to the data usability.

In the followings, we discuss the computation of $\boldsymbol{x}_i \in \mathbb{R}^m$, the reorthogonalized vector of $\boldsymbol{a}_i \in \mathbb{R}^m$ $(2 \leq i \leq n)$, where satisfies $\langle \boldsymbol{x}_i,\ \boldsymbol{x}_k \rangle = 0$ for $j \neq k$. In addition, let X_{i-1} be $X_{i-1} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_{i-1} \end{bmatrix}$ $(2 \leq i \leq n)$. Note that X_{i-1}, $\boldsymbol{x}_i$, and $\boldsymbol{a}_i$ correspond to P_k, $\tilde{\boldsymbol{p}}$, and $\boldsymbol{p}$ on line 6 in Algorithm 1, and also correspond to Q_k, $\tilde{\boldsymbol{q}}$, and $\boldsymbol{q}$ on line 10 in Algorithm 1.

Algorithm 3 CGS2 algorithm

1: **function** CGS2($X_{i-1}(=[\boldsymbol{x}_1,\ \ldots,\ \boldsymbol{x}_{i-1}])$, $\boldsymbol{a}_i$)
2: **do** $j = 1,\ 2$
3: $\boldsymbol{w} = X_{i-1}^{\top}\boldsymbol{a}_i$
4: $\boldsymbol{a}_i = \boldsymbol{a}_i - X_{i-1}\boldsymbol{w}$
5: **end do**
6: **return** $\boldsymbol{x}_i = \boldsymbol{a}_i$
7: **end function**

3.1 BLAS-based parallel implementation algorithms

3.1.1 CGS2 algorithm

The classical Gram-Schmidt (CGS) algorithm [6] is a well-known reorthogonalization algorithm. The reorthogonalization of $\boldsymbol{a}_i$ using the CGS algorithm is formulated as follows:

$$\boldsymbol{x}_i = \boldsymbol{a}_i - \sum_{k=1}^{i-1} \langle \boldsymbol{x}_k,\ \boldsymbol{a}_i \rangle \boldsymbol{x}_k. \tag{10}$$

Eq. (10) is composed of Level 1 BLAS operations, such as inner-dot products and AXPY operations. The computational cost of the CGS algorithm is about $2mk^2$ if the reorthogonalization of $\boldsymbol{a}_i$ for $i = 1,\ \ldots,\ k$ is performed. Using the matrix-vector multiplications, Eq. (10) is also replaced as

$$\boldsymbol{x}_i = \boldsymbol{a}_i - X_{i-1}X_{i-1}^{\top}\boldsymbol{a}_i. \tag{11}$$

In general, to achieve better performance, we reduce the number of data synchronizations on shared-memory multi-core processors as much as possible. The level 2 BLAS operations, such as the matrix-vector multiplications, have less data synchronization than the level 1 BLAS operations. Thus, the level 2 BLAS operations achieves better performance than the level 1 BLAS operations in parallel computing. Given this property, the CGS is conventionally implemented using matrix-vector multiplications.

However, the orthogonality of the vectors computed by the CGS algorithm deteriorates if the condition number of the original vectors is large. To improve the orthogonality, the variants of the CGS algorithm have been proposed.

The CGS algorithm with reorthogonalization (CGS2 algorithm) [3] is one of the variants. A pseudocode of the CGS2 is shown in Algorithm 3. Repeating the CGS algorithm twice, we are able to improve the orthogonality. However, the computational cost of the CGS2 is twice that of the CGS.

3.1.2 MGS algorithm

Another variant of the CGS algorithm is the modified Gram-Schmidt (MGS) algorithm. The MGS algorithm is composed of inner-dot product and AXPY operations. Then level 1 BLAS operations are mainly used. However, compared with the CGS, the MGS improves the orthogonality. Furthermore, the computational cost of the MGS is $2mk^2$ since the MGS is algebraically equivalent to the CGS.

Algorithm 4 OpenMP-based parallel implementation of CGS2 algorithm

1: **function** OMP-CGS2($X_{i-1}(=[\boldsymbol{x}_1,\ \ldots,\ \boldsymbol{x}_{i-1}])$, $\boldsymbol{a}_i$)
2: `#omp parallel private(`$j,\ s$`)`
3: **do** $j = 1,\ 2$
4: `#omp single`
5: $\boldsymbol{w} = \boldsymbol{a}_i$ ▷ Perform serially
6: `#omp end single`
7: `#omp do reduction(+:`$\boldsymbol{a}_i$`)`
8: **do** $k = 1$ to $i-1$
9: $s = -\langle \boldsymbol{x}_k,\ \boldsymbol{w} \rangle$
10: $\boldsymbol{a}_i = \boldsymbol{a}_i + s\boldsymbol{x}_k$ ▷ Array reduction
11: **end do**
12: `#omp end do`
13: **end do**
14: `#omp end parallel`
15: **return** $\boldsymbol{x}_i = \boldsymbol{a}_i$
16: **end function**

3.1.3 Compact WY algorithm

The Householder transformations [6] are also used for the reorthogonalization. However, the reorthogonalization using the Householder transformations is composed of the level 1 BLAS operations. Hence, we cannot achieve higher performance using parallel computation.

To overcome this difficulty, a reorthogonalization algorithm using the Householder transformations in terms of the compact WY representation [16] is proposed in [19]. Hereafter, this algorithm is referred to as the cWY algorithm. In this algorithm, we can rewrite the product of the Householder matrices in a simple block matrix form. Hence, the cWY can be performed mainly using the level 2 BLAS operations. This algorithm can achieve the high orthogonality theoretically and high scalability in parallel computing. In addition, the computational cost of the cWY algorithm can be reduced from $4mk^2+k^3$ to $4mk^2-k^3$ [9].

3.2 OpenMP-based parallel implementation of CGS2 algorithm

Recalling Eq. (10), the CGS and CGS2 algorithms can be parallelized in terms of the summation. Such parallel implementation is easily realized by adding OpenMP directives for shared-memory multi-core processors. From these facts, an OpenMP-based parallel implementation of the CGS2 algorithm can be represented as shown in Algorithm 4. Note that where $\boldsymbol{w}$ is a vector where preserves the original vector of $\boldsymbol{a}_i$. Hereafter, this implementation of the CGS2 algorithm is referred to as the OMP-CGS2 algorithm.

The parallel computation in terms of the summation is represented as the parallelism of do-loop as shown in line 7.

Table 1: Comparison of reorthogonalization algorithms [4]

	CGS2	MGS	cWY	OMP-CGS2
Computation	$4mk^2$	$2mk^2$	$4mk^2 - k^3$	$4mk^2$
Orthogonality	$O(\epsilon)^\dagger$	$O(\epsilon\kappa(A))$	$O(\epsilon)$	$O(\epsilon)^\dagger$
BLAS	Level 2	Level 1	Level 2	Level 1

†: Realized if the condition $O(\epsilon\kappa(A)) < 1$ is satisfied.

As the result, the inner-dot product (line 9) and the AXPY operations (line 10) in terms of the different index k are performed on each thread. In addition, the array reduction must be implemented for the summation of $\boldsymbol{a}_i$ on line 10. The array reduction in Fortran code is supported by using the `reduction` clause of OpenMP.

The advantage of this implementation is the high reusability of data. Since we compute $\boldsymbol{a}_i = \boldsymbol{a}_i + s\boldsymbol{x}_k$ (line 10) as soon as $s = -\langle \boldsymbol{x}_k,\ \boldsymbol{w} \rangle$ (line 9) is computed, the reusability of $\boldsymbol{w}$, $\boldsymbol{x}_k$, and $\boldsymbol{a}_i$ becomes higher on each thread computation. Thus, the OMP-CGS2 algorithm is expected to accelerate more effectively the reorthogonalization computation on shared-memory multi-core processors with large caches than other reorthogonalization algorithms if the vectors $\boldsymbol{w}$, $\boldsymbol{x}_k$, and $\boldsymbol{a}_i$ are stored in the L3 cache of each CPU.

3.3 Comparison of reorthogonalization algorithms

As the summary of this section, Table 1 shows that the theoretical performance of the reorthogonalization algorithms. *Computation* denotes the flops of the computational cost, *Orthogonality* indicates the bound of the norm $\|X^\top X - I\|$, and *BLAS* denotes the level of BLAS operations of which each algorithm is mainly composed. ϵ is the machine epsilon, and $\kappa(A)$ denotes the condition number of the original matrix $A = \begin{bmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_k \end{bmatrix}$.

4. Numerical experiments

In this section, we report results of numerical experiments in order to evaluate the performance of the OpenMP-based parallel implementation of the CGS2 algorithm.

4.1 Configurations of numerical experiments

In the numerical experiments, we compare the elapsed time for computing the l largest singular triplets of the same target matrix using a code of the GKLR algorithm with different l. Here, l is the number of required singular triplets; $l = 100,\ 200,\ 400,\ 800$.

We compare the elapsed time for computing subsets of singular triplets using four different codes of the GKLR algorithms. Each GKLR code is implemented with the following reorthogonalization algorithms mentioned in Section 3. **GKLR with MGS** is implemented with the MGS algorithm. **GKLR with CGS2** is implemented with the CGS2 algorithm. **GKLR with cWY** is implemented with the cWY algorithm. The reorthogonalization algorithms of the above three code are parallelized in terms of the BLAS routines. **GKLR with OMP-CGS2** is implemented with the OpenMP-based parallel implementation of the CGS2 algorithm.

Table 2: Specifications of the experimental environment

	1 node of Appro 2548X at ACCMS, Kyoto University
CPU	Intel Xeon E5-4650L@2.6 GHz, 32 cores (8 cores × 4)
	L3 cache: 20MB × 4
RAM	DDR3-1066 1.5 TB, 136.4GB/sec
Compiler	Intel C++/Fortran Compiler 14.0.2
Options	`-O3 -xHOST -ipo -no-prec-div`
	`-openmp -mcmodel=medium -shared-intel`
Software	Intel Math Kernel Library 11.1.2

In the experiments, we use three $m \times n$ real sparse matrices T_1, T_2, and T_3. All of T_1, T_2, and T_3 are set to be 256 nonzero elements, which are set to be random numbers in the range $(0, 1)$ and are randomly allocated, in each row. T_1, T_2, and T_3 are only different in the size of m and n from each other as follows: $m = 16,000$ and $n = 8,000$ for T_1. $m = 32,000$ and $n = 16,000$ for T_2. $m = 64,000$ and $n = 32,000$ for T_3. In addition, the condition number is 4.803×10^1 for T_1, 4.754×10^1 for T_2, and 4.757×10^1 for T_3, respectively.

Finally, all the experiments are run with 32 threads on a machine shown in Table 2. We use the Intel Math Kernel Library (MKL) [7] for parallelizing the level 2 and level 3 BLAS routines. The Intel MKL also provides the level 1 BLAS routines, but the implementation depends on the dimension of the target vectors and the performance of them is unstable. Thus, we use the hand-made level 1 BLAS routines, which is parallelized by using OpenMP, in the experiments.

4.2 Results of performance evaluation

Figs. 1, 2, and 3 graph the experimental results and shows the number of required singular triplets and the elapsed time for computing singular triplets of each target matrix T_1, T_2, or T_3 using the four code of the GKLR algorithm, respectively. From the figures, **GKLR with OMP-CGS2** is faster than the other code in all the cases. Thus, the OMP-CGS2 accelerates the computation of the GKLR algorithm more effectively than the other reorthogonalization algorithms.

In addition, Tables 3, 4, and 5 show the number of required singular triplets and the elapsed time spending for the reorthogonalization process in computing the singular triplets of each target matrix T_1, T_2, and T_3 using the four code of the GKLR algorithm, respectively. The tables show that the OMP-CGS2, the reorthogonalization in **GKLR with OMP-CGS2**, is at least twice faster than the CGS2 and cWY algorithms.

Note that the number of iterations at the point (k_{end}), where the GKLR algorithm stops, is the same regardless to the reorthogonalization algorithms in each of the ex-

Table 3: The number of required singular triplets (l) and the elapsed time (sec.) spending for the reorthogonalization process in computing the singular triplets of T_1 using each code of the GKLR algorithms.

# of required singular triplets	100	200	400	800
GKLR with MGS	79	176	337	1,121
GKLR with CGS2	21	57	118	315
GKLR with cWY	24	58	132	302
GKLR with OMP-CGS2	7	20	44	102

Table 4: The number of required singular triplets (l) and the elapsed time (sec.) spending for the reorthogonalization process in computing the singular triplets of T_2 using each code of the GKLR algorithms.

# of required singular triplets	100	200	400	800
GKLR with MGS	100	293	750	1,664
GKLR with CGS2	70	154	351	774
GKLR with cWY	71	166	360	796
GKLR with OMP-CGS2	25	59	151	310

Table 5: The number of required singular triplets (l) and the elapsed time (sec.) spending for the reorthogonalization process in computing the singular triplets of T_3 using each code of the GKLR algorithms.

# of required singular triplets	100	200	400	800
GKLR with MGS	261	533	1,432	2,815
GKLR with CGS2	169	372	861	1,921
GKLR with cWY	182	393	861	2,095
GKLR with OMP-CGS2	83	183	344	844

Table 6: The number of iterations at the point (k_{end}), where the GKLR algorithm stops, needed in each of the experiments. l denotes the number of required singular triplets.

l	100	200	400	800
Matrix T_1	1,000	1,600	2,400	4,000
Matrix T_2	1,300	2,000	3,200	4,800
Matrix T_3	1,600	2,400	3,600	5,600

periments. Table 6 summarizes k_{end} needed in each of the experiments.

4.3 Discussion about cache use in OMP-CGS2

As mentioned in Sec. 3.2, the high performance of OMP-CGS2 arises from the higher reusability of cache in CPU. Here, we discuss the limit size of the vectors when we perform the reorthogonalization by using OMP-CGS2. Let the number of threads in a CPU be T and the capacity of L3 cache in the CPU be C MB. The one data of the elements needs 8 bytes when we use a double-precision floating-point number.

Recalling Algorithm 4, the vectors $\boldsymbol{w}$, $\boldsymbol{a}_i$, and $\boldsymbol{x}_k$ appear at each of **do**-loop in terms of k. If all these vectors are stored in the L3 cache of CPU, we can achieve the higher performance of the reorthogonalization by using OMP-CGS2. However, $\boldsymbol{x}_k$ is not shared by different threads while

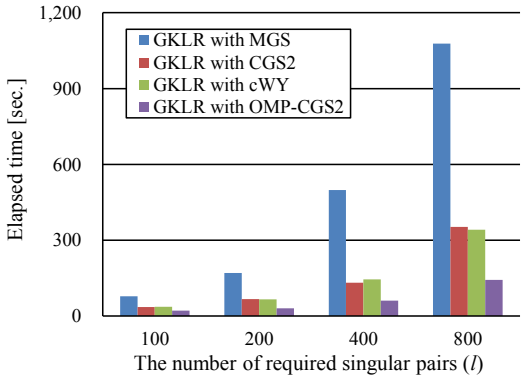


Fig. 1: The number of required singular triplets and the elapsed time for computing the l largest singular triplets of T_1 using the GKLR algorithm with different reorthogonalization implementation.

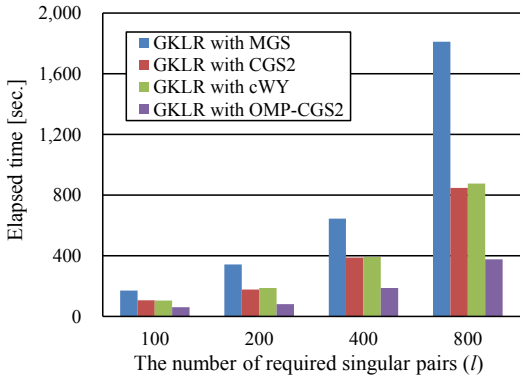


Fig. 2: The number of required singular triplets and the elapsed time for computing the l largest singular triplets of T_2 using the GKLR algorithm with different reorthogonalization implementation.

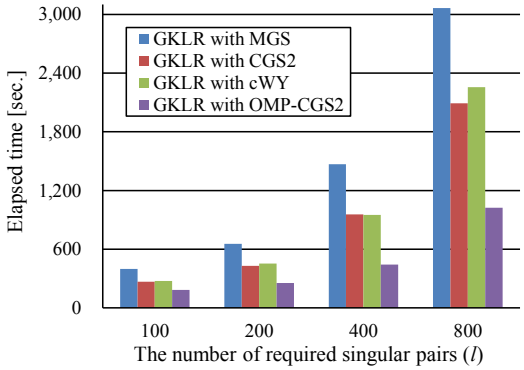


Fig. 3: The number of required singular triplets and the elapsed time for computing the l largest singular triplets of T_3 using the GKLR algorithm with different reorthogonalization implementation.

$\boldsymbol{w}$ is accessed by all computing threads. In addition, each thread should access the copy of $\boldsymbol{a}_i$ before reducing arrays.

As the results, the number of the vectors which should be stored in the cache is $(T \times 2 + 1)$.

From the above discussion, the dimension of the matrix which achieves better performance in this environment is determined by following inequality:

$$m \times (T \times 2 + 1) \times 8 \leq C \times 1024 \times 1024, \qquad (12)$$

where m is the size of the vectors $\boldsymbol{w}$, $\boldsymbol{a}_i$, and $\boldsymbol{x}_k$. Then, since $T = 8$ and $C = 20$ from the specification of the CPUs used for the performance evaluation in this paper, the following inequality holds:

$$m \leq 154202. \qquad (13)$$

Thus, under the condition (13) of the performance evaluation in the experimental environment in Table 2, the OMP-CGS2 algorithm is guaranteed to achieve the higher performance than the other reorthogonalization algorithms.

5. Conclusions and future work

In this paper, we first introduce the GKLR algorithm for computing a subset of singular triplets for target matrices. To accelerate the reorthogonalization of the GKLR algorithm on shared-memory multi-core processors more effectively, we then present the OpenMP-based parallel implementation of the CGS2 algorithm. The OpenMP-based implementation of the CGS2 algorithm has the advantage of the data reusability.

We performed numerical experiments on shared-memory multi-core processors to evaluate the performance of the GKLR algorithm with the different parallel implementations of the reorthogonalization algorithm including the OpenMP-based implementation of the CGS2 algorithm. Experimental results show that the OpenMP-based implementation of the CGS2 algorithm accelerates the GKLR algorithm more effectively for computing a subset of singular triplets for a sparse matrix than other reorthogonalization algorithms.

One of future work, is to evaluate the performance of the GKLR algorithms for larger target matrices than those we used in the performance evaluation and to extend and confirm the validity of the modeling inequality (12) depending on CPUs. The other is to apply the OpenMP-based parallel implementation of the CGS2 algorithm presented in this paper to other algorithms, such as the inverse iteration method, GMRES algorithm [15], and implicitly restarted Arnoldi and Lanczos methods [18], [2] to accelerate their reorthogonalization processes.

Acknowledgment

The authors would like to express their gratitude to reviewers of this paper for their helpful comments. In this work, we used the supercomputer of ACCMS, Kyoto University. This work was supported by JSPS KAKENHI Grant Numbers 13J02820 and 24360038.

References

[1] J. L. Barlow, "Reorthogonalization for the Golub-Kahan-Lanczos bidiagonal reduction," *Numer. Math.*, pp. 1–42, 2013.
[2] D. Calvetti, L. Reichel, and D. C. Sorensen, "An implicitly restarted lanczos method for large symmetric eigenvalue problems," *ETNA*, vol. 2, pp. 1–21, 1994.
[3] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, "Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization," *Math. Comput.*, vol. 30, no. 136, pp. 772–795, 1976.
[4] L. Giraud, J. Langou, M. Rozloẑnìk, and J. van den Eshof, "Rounding error analysis of the classical Gram-Schmidt orthogonalization process," *Numer. Math.*, vol. 101, no. 1, pp. 87–100, 2005.
[5] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *SIAM J. Numer. Anal.*, vol. 2, no. 2, pp. 205–224, 1965.
[6] G. H. Golub and C. F. van Loan, *Matrix Computations*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.
[7] Intel Math Kernel Library, "Available electronically at https://software.intel.com/en-us/intel-mkl/," 2003.
[8] I. C. F. Ipsen, "Computing an eigenvector with inverse iteration," *SIAM Review*, vol. 39, no. 2, pp. 254–291, 1997.
[9] H. Ishigami, K. Kimura, and Y. Nakamura, "On implementation and evaluation of inverse iteration algorithm with compact WY orthogonalization," *IPSJ Transactions on Mathematical Modeling and Its Applications*, vol. 6, no. 2, pp. 25–35, 2013.
[10] W. Kahan, "Accurate eigenvalues of a symmetric tridiagonal matrix," *Technical Report, Computer Science Dept. Stanford University*, no. CS41, 1966.
[11] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users's Guide*. Philadelphia, PA, USA: SIAM, 1998.
[12] Netlib, "BLAS," accssesed 2015-01-16. [Online]. Available: http://www.netlib.org/blas/
[13] OpenMP, "Available electronically at http://openmp.org/wp/," 1997.
[14] Y. Saad, "On the rates of convergence of the Lanczos and the block-Lanczos methods," *SIAM J. Numer. Anal.*, vol. 17, no. 5, pp. 687–706, 1980.
[15] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, 1986.
[16] R. Schreiber and C. van Loan, "A storage-efficient WY representation for products of Householder transformations," *SIAM J. Sci. Stat. Comput.*, vol. 10, no. 1, pp. 53–57, 1989.
[17] H. D. Simon and H. Zha, "Low-rank matrix approximation using the Lanczos bidiagonalization process with applications," *SIAM J Sci. Comput.*, vol. 21, no. 6, pp. 2257–2274, 2000.
[18] D. C. Sorensen, "Implicit application of polynomial filters in a k-step Arnoldi method," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 357–385, Jan. 1992.
[19] Y. Yamamoto and Y. Hirota, "A parallel algorithm for incremental orthogonalization based on the compact WY representation," *JSIAM Letters*, vol. 3, pp. 89–92, 2011.

Dimension Reduction Using Nonnegative Matrix Tri-Factorization in Multi-label Classification

Keigo Kimura, Mineichi Kudo and Lu Sun
Graduate School of Information Science and Technology
Hokkaido University, Sapporo, 060-0814, Japan
Email: {kkimura, mine, sunlu}@main.ist.hokudai.ac.jp

Abstract— *Multi-label classification problem has become more important in image processing and text analysis where an object often is associated with many labels at the same time. Recently, even in this problem setting dimension reduction aiming at avoiding the curse of dimensionality has gathered an attention, but it is still a challenging problem. Nonnegative Matrix Factorization (NMF) is one of promising ways for dimension reduction in unsupervised learning, and is extended from two-matrix factorization to triple-matrix factorization. In this paper, we reformulate the NMF with three factor matrices in such a way that it is solvable the problem of the combinatorial explosion of labels and incorporates the label correlation naturally in supervised learning. Experiments on web page classification datasets show the advantages of the proposed algorithm in the classification accuracy and computational time.*

Keywords: Nonnegative Matrix Factorization, Multi-label Classification, Dimension Reduction.

1. Introduction

Multi-label classification has attracted much attention in a variety of fields such as text analysis, image analysis and recommendations [1]. This is because an object often has several labels simultaneously, for example, a document may belong to *politics* and *economics*. A multi-label multi-class problem can be transformed into a set of independent single-label binary-class problems. However in that case, the relation between classes is lost.

Dimensional reduction is an essential technique in the field of machine learning and it aims to avoid *the curse of dimensionality*. The methods for dimension reduction are classified into either unsupervised or supervised method. The unsupervised methods such as Principal Component Analysis (PCA) [2] and Nonnegative Matrix Factorization (NMF) [3] reduce the dimension of the feature space ignoring the class information, while the supervised methods such as Linear Discriminant Analysis (LDA) aim to keep the class separability even in the reduced feature space. Recently, some supervised dimension reduction methods have been proposed even for multi-label classification [4]–[6]. The key idea in common is to keep the label dependency as possible in the reduced space.

Nonnegative Matrix Factorization (NMF) is one of the unsupervised dimension reduction methods and decomposes a given nonnegative matrix into a product of two lower-ranked nonnegative matrices [3]. It is reported that NMF outperforms PCA in the interpretability and even in the classification accuracy [7]. Its supervised version, NMF-LDA [8], is more advantageous in the classification accuracy. However, such supervised NMF algorithms are all only applicable to single-label classification and hard to be simply extended to multi-label classification for the difficulty to solve a set of binary-class problems in single dimension reduction scheme.

In this paper, we cope with this difficulty by proposing a multi-label NMF with the idea of tri-factorization. As seen in Fig. 1. this study is the first nonnegative supervised multi-label dimension reduction method. Our goal in this study is to find an effective representation of nonnegative data matrix with corresponding label matrix, while taking into consideration the multi-label information and the label dependency at the same time. Nonnegativity is imposed from an emprical knowledge that the nonnegativity, and the sparsity induced by the nonnegative constraint, has been to the improvement of classification in the past of study of NMF [9]–[11]. We borrow the idea of Nonnegative Matrix Tri-Factorization (NMTF) proposed by Ding *et al.* [12], one of unsupervised algorithms, which decompose a nonnegative matrix into a product of nonnegative three matrices. In this study, we decompose a data matrix into three factor matrices that have their own roles in data approximation.

1.1 Notations

We use $\mathbf{X}$ for a matrix and $\boldsymbol{x}$ for a vector. In multi-label classification, a sample $\boldsymbol{x}$ is associated with a subset $\boldsymbol{y}$ of class labels. We consider N training samples and L labels. Each sample $\boldsymbol{x}_i$ belongs to an M-dimension space and the associated label subset is represented as a binary vector $\boldsymbol{y}_i \in \{0,1\}^L$. We denote $\mathbf{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{M\times N}$ as a data matrix and $\mathbf{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N] \in \{0,1\}^{L\times N}$ as a label matrix.

1.2 Paper Organization

The rest of this paper is organized as follows. We introduce the related works in Section 2. In Section 3, we describe

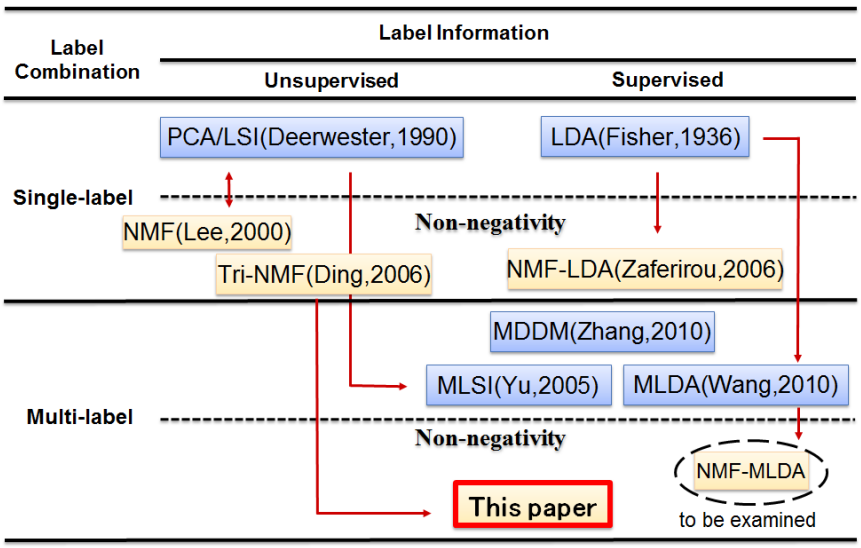


Fig. 1
THE POSITION OF THIS STUDY: A SUPERVISED MULTI-LABEL DIMENSION REDUCTION METHOD WITH NONNEGATIVE CONSTRAINT FOR THE ELEMENTS.

the proposed algorithm. Section 4 is devoted to experiments. The discussion and conclusion are stated in Section 5.

2. Related Work

According to Fig. 1, we review the methodology proposed so far.

Unsupervised Dimensionality Reduction Methods
Latent Semantic Indexing (LSI), equivalently PCA, is one of popular unsupervised dimension reduction methods and has been used in Information Retrieval [13]. LSI decomposes a matrix $\mathbf{X}$ into a product of two low-rank matrices $\mathbf{AB}$ such that $\mathbf{A}$ and $\mathbf{B}$ are low column-rank and low row-rank matrices. LSI finds the best subspace minimizing the approximation error in Frobenius norm by solving an eigen problem. Nonnegative Matrix Factorization (NMF) is another unsupervised method firstly proposed by Lee *et al.* [3]. The authors showed the advantages of NMF in comparison with LSI in text analysis and image analysis. In NMF the two factor matrices are required to be of nonnegative elements. Compared with LSI, NMF produces sparse factor matrices due to the nonnegative constraint. There are some reports saying that NMF outperforms PCA in the accuracy of classification by the virtue of the sparse representation [7], [9]–[11].

Supervised Dimensionality Reduction Methods for Single-label Classification
Linear Discriminant Analysis (LDA) is a supervised dimension reduction method for single-label multi-class classification [14]. It finds a subspace so as to maximize the ratio of between-class distance to the within-class distance. Zafeiriou *et al.* coupled NMF and LDA to produce NMF-LDA [8]. They aimed to realize both the sparsity, inheritance of NMF, and the class separability, inheritance of LDA. They constructed an objective function to achieve both in NMF-LDA.

Supervised Dimensionality Reduction Methods for Multi-label Classification
Yu *et al.* firstly conducted dimension reduction for multi-label classification problem and proposed a method called Multi-Label Informed Latent Semantic Indexing (MLSI) [4]. This method is based on LSI and decomposes data matrix $\mathbf{X}$ and label matrix $\mathbf{Y}$ into $\mathbf{AB}$ and $\mathbf{CB}$, respectively. The common matrix $\mathbf{B}$ bridges the approximation information and the label information. Zhang *et al.* proposed another method called Multi-label Dimensionality reduction via Dependence Maximization (MDDM) [5]. MDDM finds a subspace so as to maximize the dependency between the features and the associated labels in a Hilbert-Schimidt independence criterion. Wang *et al.* proposed Multi-label LDA (MLDA) as a generalization of Linear Discriminant Analysis (LDA) [6]. They redesigned the scatter matrices so as to handle multi-label setting. Other than redesigned scatter matrices, MLDA is the same as LDA. Therefore, it is straightforward to couple NMF with MLDA, such as NMF was coupled with LDA to produce NMF-MLDA, but we leave such a trial for the future work.

3. Multi-label Informed Nonnegative Matrix Tri-Factorization

We first explain the key idea of the proposed approach. In unsupervised dimension reduction, we usually consider to approximate a data point $\boldsymbol{x} \in \mathbb{R}^M$ by a linear combination of a small number of bases $\boldsymbol{u}_j \in \mathbb{R}^M$, $j = 1, 2, \ldots, J$ ($J \ll M$), as

$$\boldsymbol{x} \cong \hat{\boldsymbol{x}} = a_1\boldsymbol{u}_1 + a_2\boldsymbol{u}_2 + \cdots + a_J\boldsymbol{u}_J,$$

where $a_j \in \mathbb{R}$, $j = 1, 2, \ldots, J$ are the coefficients depending on $\boldsymbol{x}$. If samples belonging to the same class concentrate on around a representative point of the class, the number J could be identical to the number L of classes as long as single labeled samples are only considered. In multi-label problems, since such a sample $\boldsymbol{x}$ is associated with a subset of labels, the number of possible (extended) classes becomes 2^L in the same scenario. It is, therefore, infeasible to find a low-dimensional subspace, a small value of J. To cope with this problem, we take the following approach. We first assume a multi-labeled mean vector $\boldsymbol{m_y} \in \mathbb{R}^M$ is expressed by a linear combination of single-labeled mean vectors as

$$\boldsymbol{m_y} = y_1\boldsymbol{m}_1 + y_2\boldsymbol{m}_2 + \cdots + y_L\boldsymbol{m}_L,\ \boldsymbol{y} = (y_1, y_2, \ldots, y_L)^T \tag{1}$$

In addition, we consider to express the single-labeled mean vectors by J bases as

$$\boldsymbol{m}_l = s_{1l}\boldsymbol{u}_1 + s_{2l}\boldsymbol{u}_2 + \cdots + s_{Jl}\boldsymbol{u}_l, \quad l = 1, 2, \ldots, L. \tag{2}$$

From (1) and (2), we can write $\boldsymbol{m_y}$ by

$$\begin{aligned}
\boldsymbol{m_y} &= (\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_L) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_L \end{pmatrix} \\
&= (\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_J) \begin{pmatrix} s_{11} & \cdots & s_{1L} \\ \vdots & \ldots & \vdots \\ s_{J1} & \cdots & s_{JL} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_L \end{pmatrix} \\
&= \mathbf{U}\mathbf{S}\boldsymbol{y}.
\end{aligned}$$

For each pair $(\boldsymbol{x}, \boldsymbol{y})$ given as a training sample, we try to approache $\hat{\boldsymbol{x}} = \boldsymbol{m_y} = \mathbf{U}\mathbf{S}\boldsymbol{y}$ to $\boldsymbol{x}$ by choosing $\mathbf{U}$ and $\mathbf{S}$ appropriately. Once bases $\mathbf{U}$ is determined, we project $\boldsymbol{x}$ on the subspace spanned by $\mathbf{U}$ for dimension reduction.

3.1 Problem Formulation

In the following objective function to minimize, we expect that all training data $\boldsymbol{x}$ associated with $\boldsymbol{y}$ are distributed near the mean vector $\boldsymbol{m_y}$. In addition, we expect the mean vectors of frequently co-occurred classes are closely located. As a result, we find $\mathbf{U}$ and $\mathbf{S}$ minimizing

$$J(\mathbf{U}, \mathbf{S}) = \|\mathbf{X} - \mathbf{U}\mathbf{S}\mathbf{Y}\|_F^2 + \lambda \mathrm{tr}(\mathbf{S}\mathbf{L}\mathbf{S}^T), \tag{3}$$

where $\|\cdot\|_F$ denotes Frobenius norm, $\mathrm{tr}(\cdot)$ denotes the trace and λ is a positive coefficient. Here, $\mathbf{L}$ is the graph Laplacian matrix defined as

$$\mathbf{L} = \mathbf{K} - \mathbf{D},$$

where $\mathbf{K} = \mathbf{Y}^T\mathbf{Y}$, and $\mathbf{D}$ is the diagonal matrix whose lth elements is $\mathbf{D}_{ll} = \sum_{j=1}^{L} \mathbf{K}_{jl}$. The larger value of $\mathbf{K}_{ij}$ is, the more frequently class i and class j appear at the same sample.

In the first term of (3), we require that data $\boldsymbol{x}$ is close to the mean vector associated to the label $\boldsymbol{y}$ in the subspace spanned by $\mathbf{U}$. In the second term, we require that similarity between two labels is kept in $\mathbf{U}$. This is shown by

$$\begin{aligned}
\mathrm{tr}(\mathbf{S}\mathbf{L}\mathbf{S}^T) &= \mathrm{tr}(\mathbf{S}\mathbf{D}\mathbf{S}^T) - \mathrm{tr}(\mathbf{S}\mathbf{K}\mathbf{S}^T) \\
&= \sum_{l=1}^{L} \boldsymbol{s}_l^T\boldsymbol{s}_l\mathbf{D}_{ll} - \sum_{j,l=1}^{L} \boldsymbol{s}_j^T\boldsymbol{s}_l\mathbf{K}_{jl} \\
&= \frac{1}{2}\sum_{j,l=1}^{L} \|\boldsymbol{s}_j - \boldsymbol{s}_l\|^2\mathbf{K}_{jl}.
\end{aligned}$$

That is, to minimize the second term, the mean vectors $\boldsymbol{m}_j$ and $\boldsymbol{m}_l$ should be close for frequently co-occurred jth and lth classes.

3.2 Optimization

Since NMF problems are NP-hard [15], we optimize the component matrices alternatively as EM algorithm does. We use multiplicative update rules algorithm [3]. According to [3], [7], the multiplicative update rule of $\mathbf{A}$ for minimizing $\|\mathbf{X} - \mathbf{A}\mathbf{B}\|_F^2$ are expressed in general as

$$\mathbf{A} = \mathbf{A} * \frac{\nabla_A^-}{\nabla_A^+},$$

where $*$ and $/$ is the element-wise multiplication and division, respectively. Here, ∇_A^+ and ∇_A^- are the positive term and the negative term of the gradient of $\|\mathbf{X} - \mathbf{A}\mathbf{B}\|_F^2$ in $\mathbf{A}$, respectively. The gradient of (3) in $\mathbf{U}$ and $\mathbf{S}$ are calculated as follows:

$$\begin{aligned}
\nabla_U &= -2\mathbf{X}\mathbf{Y}^T\mathbf{S}^T + 2\mathbf{U}\mathbf{S}\mathbf{Y}\mathbf{Y}^T\mathbf{S}^T, \\
\nabla_S &= -2\mathbf{U}^T\mathbf{X}\mathbf{Y}^T + 2\mathbf{U}^T\mathbf{U}\mathbf{S}\mathbf{Y}\mathbf{Y}^T - 2\lambda\mathbf{S}\mathbf{K} + 2\lambda\mathbf{S}\mathbf{D}.
\end{aligned}$$

Hence, we update $\mathbf{U}$ and $\mathbf{S}$, respectively:

$$\begin{aligned}
\mathbf{U} &= \mathbf{U} * \frac{\mathbf{X}\mathbf{Y}^T\mathbf{S}^T}{\mathbf{U}\mathbf{S}\mathbf{Y}\mathbf{Y}^T\mathbf{S}^T}, \\
\mathbf{S} &= \mathbf{S} * \frac{\mathbf{U}^T\mathbf{X}\mathbf{Y}^T + \lambda\mathbf{S}\mathbf{K}}{\mathbf{U}^T\mathbf{U}\mathbf{S}\mathbf{Y}\mathbf{Y}^T + \lambda\mathbf{S}\mathbf{D}}.
\end{aligned}$$

Starting from randomly initialized $\mathbf{U}$ and $\mathbf{S}$, we update $\mathbf{U}$ and $\mathbf{S}$ alternatively until convergence. The pseudo-code of the proposed Multi-label Nonnegative Matrix Tri-Factorization (MNMTF) algorithm is shown in Algorithm 1. The non-increasing property of above these update rules can be easily found with the auxiliary function used in [16].

Algorithm 1 Multi-label Nonnegative Matrix Tri-Factorization (MNMTF)

1: **Input:** Nonnegative matrix $\mathbf{X}$ and binary label matrix $\mathbf{Y}$; Weighting parameter for label correlation λ; The number of bases J;
2: **Output:** Nonnegative matrices $\mathbf{U}$ and $\mathbf{S}$ minimizing $\|\mathbf{X} - \mathbf{USY}\|_F^2 + \lambda \mathrm{tr}(\mathbf{SLS}^T)$;
3: Initialize $\mathbf{U}$ and $\mathbf{S}$ by random positive values;
4: **repeat**
5: $\quad \mathbf{U} = \mathbf{U} * \frac{\mathbf{XY}^T\mathbf{S}^T}{\mathbf{USYY}^T\mathbf{S}^T}$.
6: $\quad \mathbf{S} = \mathbf{S} * \frac{\mathbf{U}^T\mathbf{XY}^T + \lambda\mathbf{SK}}{\mathbf{U}^T\mathbf{USYY}^T + \lambda\mathbf{SD}}$.
7: **until** Convergence criterion is met

After obtaining the subspace $\mathbf{U}$, we project all training and testing samples into the subspace spanned by $\mathbf{U}$. In the testing phase, since the bases $\mathbf{U}$ is not orthogonal and nonnegativity is required, we cannot have an analytical way to map a class-unknown sample $\boldsymbol{x}$ to $\hat{\boldsymbol{x}} = \mathbf{U}\boldsymbol{v}$. Instead we solve the following minimization problem with nonnegative constraint:

$$\|\boldsymbol{x} - \mathbf{U}\boldsymbol{v}\|^2.$$

We use this $\boldsymbol{v} \in \mathbb{R}^J$ as the new representation of original sample $\boldsymbol{x} \in \mathbb{R}^M$ in both training and test phases. In the training phase, $\boldsymbol{v}$ is given by $\boldsymbol{v} = \mathbf{S}\boldsymbol{y}$ for a pair $(\boldsymbol{x}, \boldsymbol{y})$.

3.3 Computational Complexity

All traditional algorithms such as MLSI, MDDM and MDDM solve a generalized eigen problem to obtain the subspace $\mathbf{U}$. Therefore, these algorithms need $O(M^2N)$ to form the eigen problem and need $O(M^3)$ to solve that problem with N samples of dimensionality M. On the other hand, the proposed algorithm needs $O(MNL)$ where L is the number of labels. In most cases, the number L of labels is smaller than both the dimension M of data and the number N of training samples L. Thus, the proposed algorithm is faster than these traditional algorithms in such cases.

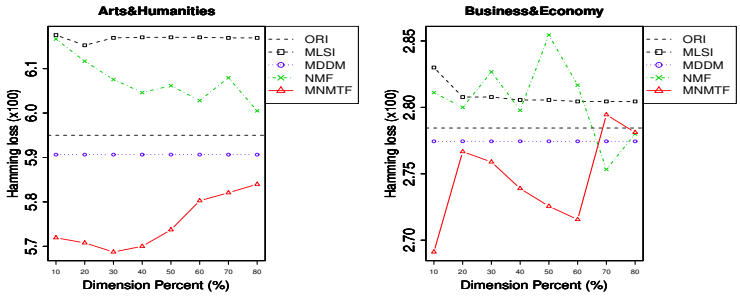
In the projection step, all traditional algorithms need $O(M(N+K)J)$ to project N training samples and K test samples. This projection is made by a simple matrix multiplication since the projection matrix is orthogonal. While, the proposed algorithm needs to solve a nonnegative least squares problem $\|\boldsymbol{x} - \mathbf{U}\boldsymbol{v}\|_F^2$. The complexity is the same $O(M(N+K)J)$, but in practice it needs several repeatations of matrix multiplications. Thus, the actual computation time is a little more than those of the traditional algorithms.

4. Experiments

We evaluated the performance of the proposed algorithm through experiments on web-page classification.

Table 1
A SUMMARY OF DATASET

Top-category	#Labels (L)	#Words (M)
Arts&Humanities	26	2315
Business&Economy	30	2192
Computers&Internet	33	3410
Education	33	2753
Entertainment	21	3200



(a) *Arts&Humanities* (b) *Business&Economy*

Fig. 2
CLASSIFICATION PERFORMANCE IN DIMENSION REDUCTION (HAMMING LOSS COMPARISON). WE OMITTED THE RESULT OF MLDA DUE TO THE WORSE PERFORMANCE.

4.1 Dataset

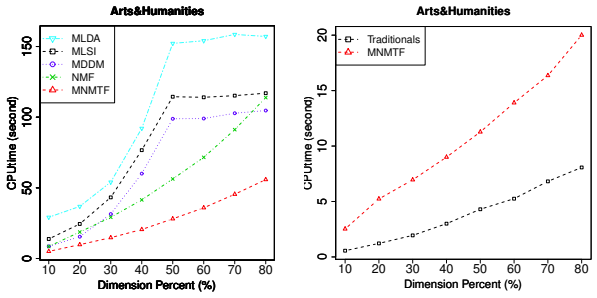
We used a yahoo web page classification dataset [17]. This dataset consists of fourteen top-categories and each of them corresponds to an independent dataset consisting of sub-categories. We consider those sub-categories as classes. Each document naturally belongs to one or more sub-categories. In this experiment, we chose five of fourteen top-categories on this experiments as shown in Table 1. We followed the setting used in [18]. For each top-category/dataset, we randomly picked up 2,000 samples for training and 3,000 for testing. Top 10% most frequently occurred words were chosen as features in such a way that we count the number of appearances of those words. Data matrix $\mathbf{X}$ represents the word appearance frequency in the documents. For more detail, see [18].

4.2 Results

We compared the proposed algorithm (MNMTF) with the other three multi-label dimension reduction methods, MLSI [4], MDDM [5], and MLDM [6]. In addition, we also compared with the original feature space (ORI) without dimension reduction and an unsupervised standard NMF without multi-label information (NMF) [3]. The weighting parameter β of MLSI β was set to the recommended value $\beta = 0.5$ [4]. In the proposed algorithm (MNMTF), we set $\lambda = 0.1$. We used a Multi-label k-Nearest Neighbor (ML-

Table 2
RESULTS ON YAHOO WEB PAGE CLASSIFICATION DATASETS. THE BOLD FIGURES SHOW THE BEST SCORE. THE SYMBOL $+/-$ LARGER/SMALLER IS BETTER IN THE CRITERION.

Dataset	Evaluation criterion	Compared methods					
		ORI	MLDA [6]	MLSI [4]	MDDM [5]	NMF [3]	MNMTF(Proposal)
Arts&Humanities	Hamming loss $(-)$	0.060	0.111	0.061	0.059	0.061	**0.058**
	One-error $(-)$	0.608	0.847	0.618	**0.565**	0.609	0.608
	Coverage $(-)$	6.269	16.815	6.468	6.395	6.364	**5.978**
	Average precision $(+)$	0.360	0.154	0.353	0.369	0.358	**0.383**
Business&Economy	Hamming loss	0.028	0.065	0.028	0.028	0.028	**0.027**
	One-error	0.119	0.596	0.123	0.126	0.122	**0.116**
	Coverage	4.023	18.598	4.053	4.125	4.030	**3.930**
	Average precision	0.394	0.193	0.393	0.391	0.394	**0.396**
Computers&Internet	Hamming loss	0.038	0.070	0.042	0.041	0.040	**0.036**
	One-error	0.421	0.702	0.429	0.404	0.413	**0.402**
	Coverage	5.343	18.193	5.606	5.607	5.491	**5.285**
	Average precision	0.389	0.193	0.382	0.388	0.388	**0.396**
Education	Hamming loss	0.040	0.067	0.042	0.041	0.040	**0.039**
	One-error	0.091	0.506	0.097	0.098	0.094	**0.090**
	Coverage	5.141	18.505	5.335	5.376	5.217	**5.084**
	Average precision	0.397	0.168	0.382	0.390	0.395	**0.406**
Entertainment	Hamming loss	0.059	0.107	0.057	0.054	0.055	**0.052**
	One-error	0.560	0.730	0.529	0.470	0.512	**0.444**
	Coverage	3.780	10.582	3.893	3.756	3.791	**3.520**
	Average precision	0.445	0.265	0.448	0.471	0.457	**0.489**



(a) *Training phase* (b) *Testing phase*

Fig. 3
CPU TIME (SECOND) CONSUMED IN TRAINING PHASE AND TESTING PHASE.

kNN) for the classifier after dimension reduction [18]. We set the number of nearest neighbors to $k = 10$ in ML-kNN (the default value in the algorithm).

In multi-label classification, several measures of performance are used at the same time instead of a single measure,*e.g.* the error rate used in single-label classification. We used four popular criteria of *hamming loss, one-error, coverage* and *average precision* [18]. We averaged the results of five-subsets in each dataset.

We varied the number of dimensionality $J = 0.1M, 0.2M, \ldots, 0.8M$. The results on "Arts&Humanities" dataset and "Business&Economy" are shown in Fig. 2. We note that NMF, MLSI and MLDA failed to improve the classification performance by reducing the dimensionality. The proposed MNMTF succeeded in total, although it is a little sensitive the value of J. We show the result with 30% dimension on Table 2. This was the best setting on "Arts&Humanities" dataset. We can see that the proposed algorithm MNMTF is almost the best in all the criteria, although the degree of improvement is only slightly more.

The time complexity shown in Fig.3 by CPU time consumed with value of J by reducing the dimensionality. The proposed algorithm is faster than the others in the training phases and slower in testing phase as the analysis predicted.

5. Conclusion

In this paper, we have proposed a supervised Nonnegative Matrix Factorization algorithm for multi-label classification problems. The key idea is to formulate a supervised multi-label problem as a factorization problem of a given data matrix into three nonnegative factor matrices one of which is a give label matrix. The results on text classification showed the advantages in classification accuracy and computational time compared with the state-of-the-art methods.

References

[1] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Dept. of Informatics, Aristotle University of Thessaloniki, Greece*, 2006.

[2] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[3] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[4] K. Yu, S. Yu, and V. Tresp, "Multi-label informed latent semantic indexing," in *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2005, pp. 258–265.

[5] Y. Zhang and Z.-H. Zhou, "Multilabel dimensionality reduction via dependence maximization," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 3, p. 14, 2010.

[6] H. Wang, C. Ding, and H. Huang, "Multi-label linear discriminant analysis," in *Computer Vision–ECCV 2010*. Springer, 2010, pp. 126–139.

[7] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari, *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. Wiley. com, 2009.

[8] S. Zafeiriou, A. Tefas, I. Buciu, and I. Pitas, "Exploiting discriminant information in nonnegative matrix factorization with application to frontal face verification," *IEEE Transactions on Neural Networks*, vol. 17, no. 3, pp. 683–695, 2006.

[9] W. Xu, X. Liu, and Y. Gong, "Document clustering based on non-negative matrix factorization," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. ACM, 2003, pp. 267–273.

[10] D. Cai, X. He, and J. Han, "Document clustering using locality preserving indexing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 12, pp. 1624–1637, 2005.

[11] D. Guillamet, B. Schiele, and J. Vitria, "Analyzing non-negative matrix factorization for image classification," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 2. IEEE, 2002, pp. 116–119.

[12] C. Ding, T. Li, W. Peng, and H. Park, "Orthogonal nonnegative matrix t-factorizations for clustering," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 126–135.

[13] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *JAsIs*, vol. 41, no. 6, pp. 391–407, 1990.

[14] B. Scholkopft and K.-R. Mullert, "Fisher discriminant analysis with kernels," in *Proceedings of the 1999 IEEE Signal Processing Society Workshop Neural Networks for Signal Processing IX, Madison, WI, USA*, 1999, pp. 23–25.

[15] S. A. Vavasis, "On the complexity of nonnegative matrix factorization," *SIAM Journal on Optimization*, vol. 20, no. 3, pp. 1364–1377, 2009.

[16] D. Cai, X. He, J. Han, and T. S. Huang, "Graph regularized nonnegative matrix factorization for data representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 8, pp. 1548–1560, 2011.

[17] N. Ueda and K. Saito, "Parametric mixture models for multi-labeled text," in *Advances in neural information processing systems*, 2002, pp. 721–728.

[18] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

A Music Composition Model with Genetic Programming –A Case Study of Chord Progression and Bassline-

Kanae Kunimatsu, Yu Ishikawa, Masami Takata, and Kazuki Joe
Graduate School of Humanities and Science, Nara Women's University, Nara-City, Nara, Japan

Abstract - *In this paper, we present the improvement of our automatic music composition model using genetic programming, which is constructed with two independent but cooperative models for chord progression and melody. The improvements are the chord progression generation model and the bassline generation model. The chord progression generation model, considering tension notes, generates chord progression suitable for jazz blues. In the music theory, tension notes that are granted for the corresponding chords are determined for some cases but it is not easy to automatically determine what to apply from such theoretical codes. In this model, by focusing on the relation between the chord's top notes with tension notes and melody tones, we attempt to automatically generate chords. Further, by adding the bassline generating part, this system is applied to Jazz improvisation.*

Keywords: Genetic Programming, chord progression, bassline

1 Introduction

In recent years, as the spread of ICTs, a large number of users want to transmit their music contents by utilizing the CGM (Consumer Generated Media). There are composer support systems with a variety of DTM (Desk Top Music) software. The composer support systems generate music with just giving initial parameters such as keys, tempo and moods while others generate music with giving interactive selections of user's favorites during the music generation processes.

Since good music compositions require professional knowledge and experience, it is difficult for ordinal persons to compose music. To make them easily compose music, automatic music composition systems have been proposed. As methods of automatic music composition, a probabilistic method and an evolutionary computation method have been proposed. In this paper, we develop an automatic music generation system with the evolutionary computation that does not require professional knowledge or experience. Evolutionary computation is an engineering model with the mechanism of biological evolution to be applied to combinatorial optimization problem, machine learning and system analysis. Considering the automatic composition is basically an optimization problem of searching an optimal combination from the combinations of large number of musical notes, the evolutionary computation is suitable for the automatic composition.

As an automatic composition system using evolutionary computation, we have proposed the automatic music composition system with genetic programming (GP) [1]. It utilizes the genetic structure of GP, namely a tree structure, to associate the tree depth level with the length of a music note. In this framework, any length of music notes can be expressed for melody line and chord progression. In [1], we narrow down the music genre to blues and adopt a seventh chord in a chord progression part. However, the generated chords do not contain tension notes, which are absolutely imperative for expressing various sounds such as Jazz. Actual Jazz improvisation is performed with voicing, namely omitting chord tones or adding tension notes, to generate more various sounds. In this paper, we improve the chord progression generation model by adding chords that include tension notes and voicing the chords. Thereby, we improve the chord progression model so that the generated chords include tension chords and look like a real Jazz performance. Furthermore, in order to apply this system to Jazz improvisation, we propose the third model to generate bassline.

This rest of the paper is constructed as follows. In section 2, we explain tension notes and voicing. In section 3, we introduce the music composition model with genetic programming. In section 4, we improve the chord progression model and propose the bassline model. In section 5, we present some experiments to validate the model.

2 Characteristics of Blues Music

In this section, first we explain chord classification and blues chord progression. Next, we describe tension chord and voicing.

2.1 Chord classification

A chord is categorized into three functions.

- Tonic (T) : the basic note: beginning and ending
- Dominant (D) : notes for returning to Tonic
- Subdominant (SD) : notes for procession and connection: easy to transfer to Dominant

As above described, there is the basic chord progression of T-SD-D-T. T-D-T and T-SD-T are also natural progressions. In

Table 1: A musical scale plotted with I to VII.

	T	D	SD
Primary triad	I	V(7)	IV
Secondary triad	VI	VII	II

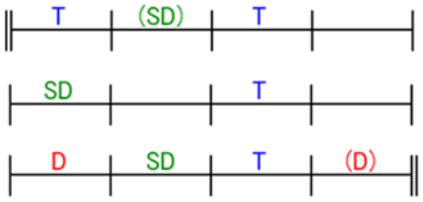


Fig. 1: Blues chord progression

Tab.1, a musical scale is plotted with I to VII. III is not presented because the property of III is not included.

2.2 Blues chord progression

Blues is a music format that consists of 12-bars, and is played with adding blue notes to the major scale to fit to the fixed chord progression, using only three of the four types of Seventh chord (dominant seventh, major seventh, minor seventh, minor major seventh). Figure.1 represents a blues chord progression. In harmonics of traditional western music, the chord progression of D-SD is not used because it is considered as unstable sound. But it is included between the 9^{th} and 10^{th} bars in the blues chord progression. When the 11^{th} and 12^{th} bars are T, it means that the song is finished.

2.3 Adding tension notes

A tension note is a note of non-harmonic tones to give a sense of tension in the chord sound without interfering the chord progression. A chord with tension notes is called a tension chord. In order to provide effective tension notes, voicing should be considered at the same time as the melody. The applicable tension notes to a chord are decided by music theory but it is difficult to make a decision which tension notes are actually used. Dissonant halftone collision against melody sound is an improper harmony to be avoided. In our model, the code generation is performed so that the relation between the top note of the tension embedded chord and the melody sound is considered.

2.4 Voicing

When a chord is given, the operation of arranging the composition tones of the chord in each voice part is called voicing that presents very colorful musical expressions. Especially in popular music such as jazz, the use of tension notes gives a tension to the chord sound to perform the effect of voicing. Adding a tension note to chord tones, while keeping the harmonic function of the chord, it is possible to express a colorful sound. There are two methods for voicing: 1) close voicing, which takes a dense placement of the chord tones within an octave and 2) open voicing, which take a wide placement beyond an octave. Any number of voicing patterns for a single chord is possible and the number of voicing patterns with tension notes increases explosively. In this paper, we just focus on a voicing technique called spread voicing, which is a kind of open voicing, for chord generation.

2.5 Spread Voicing

The spread voicing takes placement of the chord tones beyond an octave to keep the root of the chord as the bottom note. Generally used in jazz piano, since the harmonic tones are arranged in a wide range, it presents more effective and richer harmony tones to the melody. As the chord progression generation part, we adopt the chord generation using a 4-way spread voicing method.

3 A Music Composition Model with Genetic Programming

In this section, we explain the music composition model with genetic programming [1]. The composition model consists of two parts: chord progression and melody. In general there are two methods for composing music. One method is melody first to apply chords in accord with the melody while the other method is chord first to create melody using the chord. The composition model adopts the latter method provided that tempo and accent are not changeable.

To generate good music that is not confined by existing ones, both parts make use of genetic operators. To represent any length of notes such as a triplet and dotted one within each individual, we adopt GP to express complicated data such as tree structures rather than GA that is based on array structures. Figure 2 shows an example of genetic individuals. The depth level of the tree structure represents the length of the notes. The deeper the depth level is, the finer note it represents. Figure 3 gives examples of a binary tree and a triplet tree. In the case of the binary tree, the note length in the upper level is as twice as in the lower level. In the case of the triplet tree, the note length in the upper level is as 3 times as in the lower level. Until it meets certain criteria, genetic operators such as crossover or mutation are repeated to develop next populations. Each individual gene represents a chord progression and a melody line with the length of a fixed bar. Each node has the number of branches in the case of non-terminal nodes or in the case of terminal nodes the type of chord (chord progression) or the sound information (melody) such as sound, keyboard continuation symbol and rest. Using depth-first search, just the terminal nodes that contains the sound information are detected from left to right and it is possible to replace old gene individuals of chord progression and melody with new ones. Thus, it generates a melody line based on the chord progression by bar unit.

Although the use of GP eliminates the limit of the note length such a triplet or dotted note, it is difficult to maintain the similarity and the continuity of the new melody bar with the previous bar. When music is generated, we use some restrictions to retain the music similarity and the continuity from the past by utilizing the characteristics of the chord progression. In this model, we squeeze the music genre blues, and evaluate the generated music in terms of the blues characteristics. The number of bars of a song is set to 12 for one individual suitable for the blues. Based on the first 12 bars of a blues song, a partial chord progression is generated by selecting a good individual in the chord progression generation model. Then, a partial melody line is generated by selecting a good individual in the melody generation model.

The partial chord progression i is evaluated with the fitness using the following items.

I. Percentage that satisfies the chord progression pattern of blues

II. Integrity with the melody notes in each bar of the partial melody *i-1*

Item I is used for adapting to the blues chord progression pattern shown in Fig.1. Item II is used for imitating the i^{th} 12 bars with the $i\text{-}1^{th}$ 12 bars. The first note of the melody in each bar among partial melody *i-1* and the chord of each leaf node among partial chord progression *i* are examined to calculate the rate if the first note is included in the chord.

The partial melody *i* is evaluated with the fitness using the following items.

1. Integrity with the partial chord progression *i*

2. Comparison of the music entropy function with the partial melody *i-1*

3. Comparison of the rhythm patterns with the partial melody *i-1*

Item 1 is used for generating the partial melody *i* suitable for a given partial chord progression *i*. The notes of each bar in partial melody *i* are examined to calculate the rate if they are included in the chords of each bar in the partial chord progression *i*. Item 2 and 3 are used so that the impression between the partial melody *i* and the partial melody *i-1* does not change abruptly. Each comparison is performed by bar unit. In item 2, we propose the concept of impression as music entropy to compare the difference of impression as numeric values. Using the transition probability of the partial melody *i-1*, the music entropy is calculated by bar unit to quantify the occurrence degree for each note. We assumed that the impression is similar when the music entropy values are close. In item 3, the average of note values in a bar is calculated to compare the rhythm intervals.

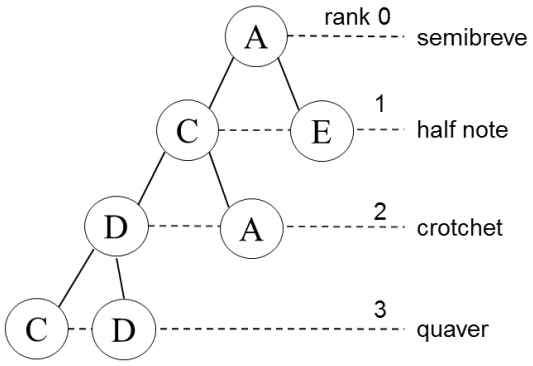


Fig. 2: An example of genetic individuals.

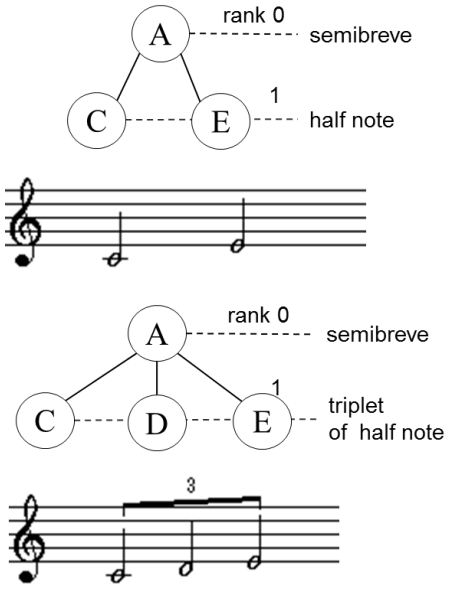


Fig. 3: Difference of expression by the number of branches.

Through the above fitness evaluations, the partial chord progression *i* and the partial melody *i* get involved in the partial melody *i-1* and the partial chord progression *i-1*, respectively. Since we focus on blues in this paper, each partial chord progression and melody *i* consist of 12 bars as explained in 2.2. Note that this kind of interaction is found in real Jazz blues improvisation.

4 Improvement of the music composition model

In this section, we explain the improvement of the chord progression generation part and the newly developed bassline generation part. Figure 4 shows the overview of the whole model. GP is applied in each part to generate melody, chord progression and bassline in this order.

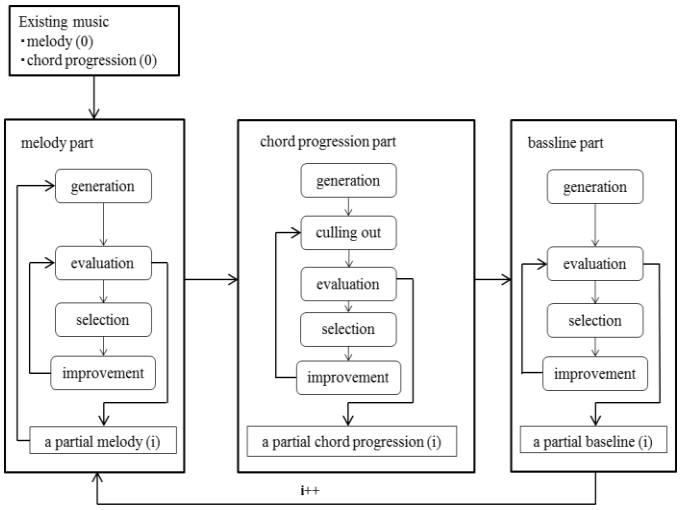


Fig. 4: Overview of the model.

Table 2: An example of 4-way spread voicing

	pattern1	pattern2
Top note	A tension note	A chord tone
Middle notes	7th	3rd
	3rd	7th
Bottom note	Root	

4.1 Chord progression part

In the chord progression generation model, a voicing is applied to the three chords used in the blues form to produce various types of chords. In order to generate the top note of the chord so that it is suitable for the melody, this model generates the melody first.

4.1.1 Information of individuals

In this model we perform a 4-way spread voicing: the root as the bottom note, the combination of 3^{th} and 7^{th} as the middle two notes and a tension note or a chord tone as the top note. Table 2 shows an example of a 4-way spread voicing. A candidate for the tension note includes minor 9^{th}, 9^{th}, major 9^{th}, 11^{th}, major 11^{th}, minor 13^{th} and 13^{th}. The notes that constitute the minor 9^{th} upwardly from each note of chord tones are called avoid note and often avoided from the use of tension notes. However it is sometimes intended to be used depending on the type of the target chord. For example, in the case of dominant seventh chords, although the note of 11^{th} should be avoided as an avoid note, other notes (minor 9^{th}, 9^{th}, major 9^{th}, major 11^{th}, minor 13^{th}, 13^{th}) can be used as a tension note because they do not interfere in the function of chords. In this chord progression generation model, voicing is applied to the three chords used in existing music to get candidates for terminal nodes, and 22 (11 (tension notes or chord tones) × 2 (Voicing patterns)) types available for a single chord are prepared to perform a 4-way spread voicing as shown in Table 2. In general, typical chord progression changes in a half note or longer when the theme is presented. However, in the case of improvisation, the chord progression becomes often-complicated and finer. In this paper, we limit the chord progression change by a half note.

4.1.2 Fitness evaluation

To improve the chord progression generation part explained in the previous section, we add the following two items to calculate the fitness for the partial chord progression *i*.

III. Comparison of the top notes of the chords and the melody in each bar of the partial melody *i*

IV. Comparison of the entropy function with the partial chord progression *i-1*

Item III is used for the comparison of the top notes, including the tension notes of chords for each bar, with the melody line. The aim of the evaluation is to avoid the dissonant semitone collision with tension notes and the melody as much as possible. When the first note of the partial melody *i* and the top note of the chord are in contact with major 2^{nd} and minor 2^{nd} at a chord change time, the fitness value is decreased. When an avoid note is selected for the top note, the fitness value is decreased, too.

Item IV is used for preventing abrupt changes in the impression between the partial chord progression *i* and the partial chord progression *i-1*. In chord voicing, the top note should be selected so that the continuous musical progression is observed. In the chord progression generation of this model, we compare the music entropy values for the top notes of the chords to avoid abrupt change of the music impression, and make the successive bars enough similar.

Let a set S be $\{X_1, X_2, \cdots, X_{14}\}$ that represent 12 kinds of top notes of a chord, continued and rests, namely {*C, Cs, D, Ds, E, F, Fs, G, Gs, A, As, B, *, ~*}. In addition, let the probability of the occurrence of an event X_k under the condition in event X_l occurrence be Y_{kl}. The music entropy is defined as follows.

$$\text{Music Entropy} = -\sum_{kl=2}^{nN} Y_{kl} \times \log Y_{kl} \qquad (1)$$

As the characteristic of the blues form, the tonic chord in the first four bars out of the 12 bar (1 chorus) is highlighted, then the next four bars are transited from the subdominant chord to the dominant chord, and finally the last four bars are transited from dominant chord to the tonic chord. Therefore, it has a structure with three parts, each of which contains 4 bars. From the blues structure, the value of the music entropy in the chord progression generation part is calculated every 4 bars to compare the value with the corresponding 4 bars section in the previous partial chord progression. Note that the partial chord progression with music entropy 0 means that the chord progression does not change from the previous chord progression at all. On the other hand, the partial chord progression with music entropy 1 means that the chord progression consists of completely random notes; it is more free than "free jazz".

From a given original blues song, the partial chord progression *0* is generated without any tension notes and each bar contains a single chord. In the partial chord progression *1*, the subdivision of each chord is performed to select the individual such that the total of the transition probabilities Y_{kl} between the chords included in the 4 bars becomes larger and the top notes of the chords walk smoothly. At the same time the music entropy of each 4 bars is calculated to compare the music entropy of the next partial chord progression.

4.2 Bassline part

In the bassline generate model, the 4-beat bassline often used in jazz is generated. To generate a bassline suitable for the chord progression, the bassline is generated right after the chord progression generation.

4.2.1 Information of individuals

Since we require the 4-beat bassline for jazz improvisation, generated bassline individuals are fixed to crotchet. Therefore the number of branches of a non-terminal node in the depth level below 0 is just 2 as in the same case of the chord progression generation part. The individual of terminal nodes includes the notes of the key range, hold (~) and rests (*).

4.2.2 Fitness evaluation

The partial bassline *i* is evaluated with the fitness using the following items.

A. Integrity with the partial chord progression *i*

B. Comparison of the music entropy function with the partial melody *i-1*

Item A is used for generating the partial bassline *i* suitable for a given partial chord progression *i*. The notes of each bar in partial bassline *i* are examined to calculate the rate if they are included in the chords of each bar in the partial chord progression *i*. In each bar, when the first beat and the third beat match the chord tones, the fitness value is highly increased. The fitness value is decreased when a note of the bassline is in contact with a corresponding chord tone by a semitone. Item B is used so that the impression between the partial bassline *i* and the partial bassline *i-1* does not change abruptly. This evaluation is performed every 4 bars unit.

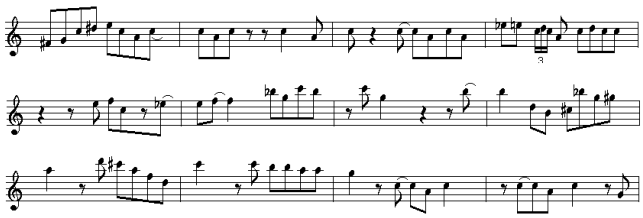


Fig. 5: Original melody

In the partial bassline *i*, individuals are selected so that the total of the transition probabilities between the bassline notes included in the 4 bars becomes larger and the notes walk smoothly. At the same time the music entropy of each 4 bars is calculated to compare the music entropy of the next partial bassline.

5 Experiments

In this section, we report the experimental results about chord progression generation and baseline generation.

5.1 Experimental environments

In our model, we perform a generation of music derived from the previous 12 bars every 12 bars, to be close to the method of blues improvisation. Therefore, we use an existing music as partial melody 0 and partial chord progression 0. In this experiment, we use a famous standard jazz number with the blues format, Billie's Bounce by Charlie Parker. Figure 5 shows the first 12 bars of Billie's Bounce. Each part of melody, chord progression and bassline is represented as a gene individual with 12 bars; the number of non-terminal nodes in the depth level (*level*) 0 is fixed to 3 to generate three blocks of 4 bars in *level* 1. Thus, it is easier to compare the partial melody, the partial chord progression and the partial bassline during fitness evaluation. Terminal nodes are set to be generated in *level* 4 or deeper to generate the 12 bars.

For terminal nodes of chord progression, we prepare voicing chords by adding tension notes or chord tones to existing three chords (C7, F7 and G7). Since typical chord progressions vary from two beats to one bar, the early genes individual depth level is set to 4; the early genes individual consist of half notes in this experiment. The parameters we used for the experiments are the number of individuals (*nI*), crossover probability (*c*), mutation rate (*m*), and the maximum generation number (*Generation*). These parameters are not theoretically decided but empirically found. In [1], the preliminary experiments for determining the numbers of average individuals and average generations are reported that good gene individuals among 100 individuals are mainly selected by the 50th generation. The crossover probability and

the mutation rate are set to the same values in [1]. Thus, we adopt the following parameter choices for the chord progression generation part.

- *nI*=100
- *c*=0.8
- *m*=0.1
- *Generation*=50

For terminal nodes of the bassline generation part, we prepare 17 kinds of nodes that consist of 15 notes for the target (2E - 3G) range, a hold and a rest. The initial gene individuals are adjusted to the depth level of 5. In addition good gene individuals are selected by the 200th generation from our preliminary experiments for bassline. Thus, we adopt the following parameter choices for the bassline generation part.

- *nI*=100
- *c*=0.8
- *m*=0.1
- *Generation*=200

5.2 Experiment results

Figure 6 shows an example of the partial score 1 (the partial melody 1, the partial chord progression 1 and the partial bassline 1). Figure.7 presents an example of the partial score 2. The first stage shows the melody score, the second and third stages show the chord progression score, and the fourth stage shows the bassline score.

In the chord progression generation part, all of generated chords consist of half notes that are the same to the initial individual because the chord progression generation part does not perform note length comparison like the melody part. As for top notes, the chord progression includes chords with tension notes. Comparing the top notes of the generated chords and the melody tones in each bar, the notes in contact with minor 2nd or major 2nd are relatively selected out. However several sound conflicts between tension notes of the chords and the melody tones are observed. So several improper chords are generated. A possible reason is that the target melody tone to be compared with the chord top notes is just focused on a melody tone when the chord progression changes. In the partial chord progression 1, the top notes are selected such that they do not comparatively create a large movement, and any unnatural leap is not observed. In the partial chord progression 2, since it performs a comparison with the music entropy calculated in the partial chord progression 1, the generated top notes are in a natural flow.

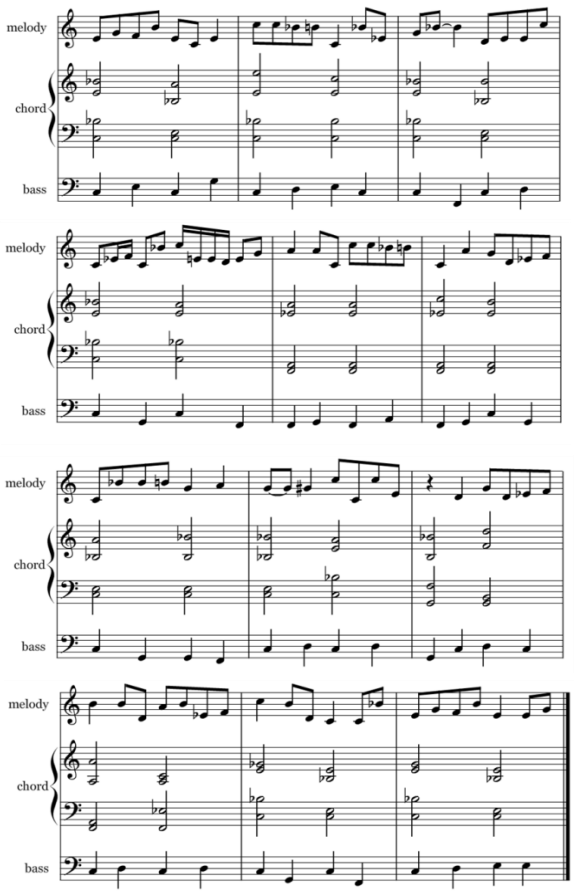


Fig. 6: An example of partial score 1.

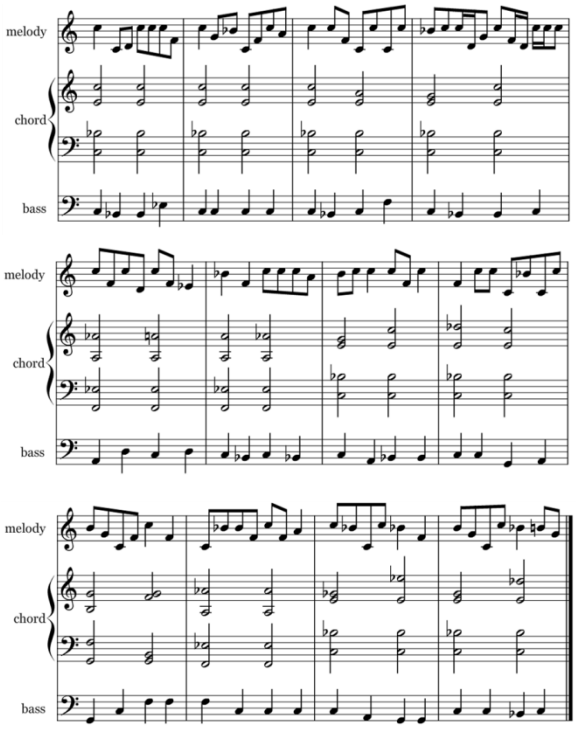


Fig. 7: An example of partial score 2.

The bassline generates only crotchets and four beat based individuals are generated. The first beat and the third beat are generated from the notes of the corresponding chord tones, and in particular the first beat is mainly generated from the individuals with the root of the chord. The second beat and the fourth beat are often generated with tension notes to collide in a semitone with chord tones and generate dissonant sounds.

From the experiments, in the case of the chord progression we observe that tension chords are generated by selecting out the sounds with semitone collisions and comparing the music entropy for the chord top notes at the fitness evaluation. In the case of the baseline generation, we observe that sounds suitable for chord tones are successfully generated. However, each evaluation formula interferes with each other. Therefore, any tone column to satisfy all the evaluation formula is not generated. In particular, since the chord progression and the bassline tend to generate the sounds causing semitone collision that are not completely selected out, it is necessary to improve the evaluation formula.

6 Conclusions

In this paper, we proposed a chord progression generation model considering tension notes and a baseline generation model using GPs. Since target music is blues, each part for individuals is fixed to 12 bars.

The chord progression generation model generates chords including tension notes suitable for the given melody with applying selection methods based on the specific chord progression blues, and select out the sounds where the chord top notes and melody tones become dissonant. The bassline generation model generates 4^{th} note based bassline suitable for the chord progression by fixing the rhythm to crotchet, and evaluates the integrity with the corresponding chord tones. In addition, both the chord progression model and the bassline model adopt an evaluation method for music impression by music entropy reported in [1]. Thus, we prevent abrupt changes during the top notes of each chord and the baseline. In the generation of the partial chord progression 1 and the partial baseline 1, we select the individuals with high transition probabilities between chords or baseline that is defined in the music entropy. In the partial score 1, thereby individuals such as top notes of chords and the bassline show natural flows.

Our future work includes the improvement of evaluation formula and multipurpose GPs because it is not possible to completely select out the sounds that cause semitone collision in our current models. Furthermore, we aim to create a system to perform a jazz session with human in real-time. Thus, we would like to apply our future system to real jazz improvisation.

7 References

[1] K.Komatsu, T.Yamanaka, M.Takata and K.Joe, "A Music Composition Model with Genetic Programming," in *Proc. PDPTA'10*, II, pp.686-692, 2010.

[2] J.A.Biles, "GenJam: A Genetic Algorithm for Generating Jazz Solos," In Proceedings of the 1994 International Computer Music Conference, ICMA, SanFrancisco, pp.3-4, 1994.

[3] T.Kitahara, M.Katsura, H.Katayose and N.Nagata, "Automatic Chord Voicing System Using Bayesian Network," IPSJ journal, Vol.50, No.3, pp.1067-1078, 2009.

[4] N.Emura, et al., "A modular system generating Jazz-style arrangement for a given set of a melody and its chord name sequence," Acoust.Sci.&Tech., Vol.29, No.1, pp.51-57, 2008

[5] Noteflight-Online Music Notation , http://www.noteflight.com/.

Sign Language Recognition using Leap Motion Controller

Makiko Funasaka, Yu Ishikawa, Masami Takata, and Kazuki Joe
Department of Information and Computer Sciences, Graduate School of Humanities and Sciences,
Nara Women's University, Nara, Japan

Abstract – *Making deaf people use an alternative method instead of current voice input to ICT equipment, we propose a sign language recognition method using Leap Motion Controller. As decisions using sign language, 16 kinds of decisions that focus on characteristic of hands and fingers are proposed. Sign language recognition algorithm is constructed using 16 kinds of decisions. The constructed flowchart is differing as order of decisions, recognition rate for all letters change from the difference accuracy rate of decisions. For sorting of decisions is enormous combination, genetic algorithm is applied to search for the optimal solution in the automatic construction of sign language recognition algorithm.*

Keywords: Leap Motion Controller, sign language recognition, Genetic Algorithm

1 Introduction

As smart phones and tablet devices have been improved, voice input and voice recognition interfaces have been widely used. The iPhone has Siri[1] that can answer questions by using natural language processing and web services. The Google Search[2] is a search engine that accepts voice input, too. The advantage of voice input is faster operations than by keybord or pad touch, so it is the minimal burden for ordinal users, in particular, for elderly users who are not familiar with text input. However, the voice input and the voice recognition interfaces is extremely difficult for deaf people. Therefore, any interfaces, which do not need voice input, should be developed for those poeple.

A sign language is a kind of communication means for deaf people. By using a sign language as an input interface to ICT devices, it is possible for deaf people who are considered very difficult to input by conventional keyboard and touch pad. A sign language use visual information with the use of finger, hand and arm operations. At the same time some finger operations are used with a part of the face such as line of sight and mouth. Fingerspelling can represent one of the alphabet 26 letters in the form of fingers.

In existing research for sign language recognition, the image recognition of color images, depth images and hand shapes are used[3]. Since it must be taken with colored gloves[3], the glove worn is not convenient. The image recogniton requires long computation time to detect the hand and fingers. Thus, it takes relatively a long interval to obtain the final recogntion result. In the case of the recognition with Kinect[4], large space is required for skeletal tracking. It is difficult to recognize the fingerspelling anywhere with Kinect. Therefore, sign language recogniton is required using a compact device that can directly recognize the shape of fingers or hands anywhere.

In this paper, we propose a fingerspelling language recogniton method using Leap Motion Contoroller[5][6]. Putting hands and fingers over a Leap Motion controller, the fingerspelling recognition is performed. Leap Motion has skeletal tracking that recognizes the framework of fingers to obtain a highly accurate various data such as the position of finger bones and the degree of the thumb and index finger. In addition, the use of Leap Motion allows fingerspelling recogniton without any physical contact.

The rest of the paper is constructed as follows. Section 2 provides related works of sign language recogniton in detail. Section 3 explains a fingerspelling recogniton method. Section 4 describes the search for the optimal solution in the automatic generation of fingerspelling recogniton algorithms. In Setion 5, we perform experiments using the proposed method.

2 Related works

As related works of sign language recognition, we explain three studies.

In the first study, colored gloves are adopted to obtain hand shape recognition applied to the conversion of a sign language [3]. Colored gloves dyed with 6 different colors are worn and the feature vectors are calculated using the Morphological Principal Component Analysis from the captured images. Hand detection and finger modeling are performed in natural background by using the colored gloves. The proposed feature vector extraction method makes the camera distance be highly independent. To analyze the performance of feature extraction, neural network is trained for the position recognition of a sign language with 26 alphabet letters. The neural network is a feed-forward three layers perceptron type with the backpropagation learning method. The numbers of neurons in the intermediate and output layers are 42 and 26, respectively. The 26 alphabet letters are characterized by a 20-dimensional vector from finger data without the palm of the hand. The result of experiments using a test set of 30 samples by each hand shows the recognition rate of 93.396%.

Second, in order to recognize a static sign language by hand, a robust approach using a novel combination of features

is proposed [7]. The features are the color of images, the depth of images and the hand shape. Obtaining depth, color and skeleton data by Kinect, the proposed accurate hand segmentation divides them into a hand color image and a hand depth image. Color and depth feature vectors are characterized by the Local Binary Pattern (LBP) histogram calculated from the hand color image and the hand depth image. Extracting a hand shape from the hand depth image, the shape feature vector is characterized. The combination of these three feature vectors are used for the recognition using template matching and Support Vector Machine (SVM). Two experiments are conducted. First, to examine the classification accuracy in combinations of different features, some experiments are conducted with 4 types of feature combinations; depth only, color only, color and depth or color, depth and hand shape. The three conditions of skin color, hand size and distance of camera are changed at data collection, and the proposed hand segmentation algorithm is used with different lighting conditions. The dataset composed of 8 different hand poses denoting letters of the fingerspelling alphabet from 'A' to 'H' is generated. Each sign is performed 10 times by 12 non-expert signers. It gives a total of 120 color and depth images for each of the 8 alphabet signs. As the result of the experiment, when the proposed feature combination of color, depth and hand shape is applied, the recognition rate of 95% is higher than other combinations. Next, the target signs from 'A' to 'Y' excluding sign 'J' and 'Z' which involve motion are used for the experiment. In a fingerspelling data set, 500 color and depth image pairs per sign are obtained and five data sets are used for the experiment. The result of the experiment is the recognition rate of 92.14%.

The third is a study to recognize the alphabet fingerspelling 26 letters using Leap Motion Controller [6]. In this study, finger data is obtained from the Leap Motion API. Palm data consists of the unit direction vector of the palm, the position of the palm center, the velocity of the palm and the accuracy of the data. At the same time, the grab strength, the pinch strength, the sphere center and the sphere radius are obtained. The finger data consists of the direction of each finger, the length of each finger, the tip velocity and the position of joints for distal phalanges, intermediate phalanges, proximal phalanges and metacarpals. To apply machine learning the data obtained from the Leap Motion API, feature vectors are calculated. As the features of palm, the pinch strength, the grab strength, the average distance, the average spread and the average tri-spread are used for the machine learning. The average distance is calculated as the sum of the distances between the fingertips in adjacent frames. The average spread for the palm is estimated based on the distance between adjacent fingertips. The tri-spread is the triangular area between two adjacent finger tips and the midpoint of the two finger's metacarpal positions. The average tri-spread is calculated by adding the triangle area of all pairs of fingers and dividing by the total number of the frames. For each finger, extended distance, dip-tip projection, orderX and angle are derived. The extended distance is the maximum distance of all points of the finger from the palm center. The Dip-tip projection is the projection of the dip-to-tip vector onto the palm normal vector. The OrderX is the order of the finger along the x-z plane with respect to other fingers. For the experiment, data are collected from two (deaf and normal) teachers of deaf education. In order to classify the 26 letters using, a k-nearest neighbor method is applied and the recognition rate is 72.78%. The use of SVM improves the recognition rate to 79.83%.

Among the above studies, in the case of colored glove based recognition, the user may feel a troublesome to wear them and a photograph picture must be taken with wearing the colored glove, which is not very convenient. In the case of Kinect based recognition, a large space is required to obtain depth as well as and color image for the skeletal tracking, which is not easy for ordinal use. In the case of Leap Motion based recognition, the machine learning requires large computation for a new person.

3 Sign language recognition using a sensor device

3.1 Detection of finger using Leap Motion Controller

Hardware and software of Leap Motion Controller are used for get the following five functions: 1) hands detected in a frame including rotation, position, velocity and movement from the last frame, 2) all fingers and pointing tools recognized by hand with rotation, position and velocity, 3) the exact pixel location on a display pointed at by a finger or a pointing tool, 4) recognition of basic finger gestures such as swipes and taps, and 5) detection of position and orientation changes between frames [6]. In this paper, hands and fingers are detected to obtain the normal vector of the palm, coordinates of fingertips and finger bone, the direction vector of arm and the direction vector of the fingertip.

3.2 Decision tree and the recognition rate for fingerspelling letters

The Recogniton target is fingerspelling 24 letters without movement out of 26 letters. Although there are several alphabetic representations for fingerspelling, we use the fingerspelling representation as shown in Fig.1 [9].There are differences in fingerspelling representations ; if the palm of hand is facing the opponent, if the back of hand is facing the opponent, or which finger is bent how. Considering these representations, the conditional branches constructing a decision tree are 16 kinds of 'a' to 'p' as shown in Table 1, and the decision tree is suitable for programming. The detail of each conditional branch is shown in Table 1. The conditional branch is just two ways of Yes or No. Figure 2 is an example of decision trees. In Figure 2, left arrow is Yes and right arrow is No. It should be noted that, "a finger is fully extended" means all of the first, the second and the third

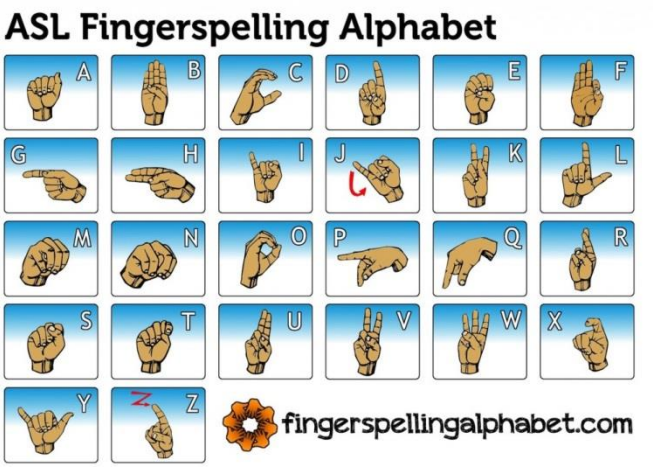


Figure 1 : Alphabetical fingerspelling

Table 1 : Detail of each conditional branch

	Conditional Branch
a	The palm of hand is facing the opponent.
b	Hand orientation is above.
c	There is a finger that makes a wheel with the first and the second joints
d	The third joints are extended except the thumb.
e	The back of hand is facing the opponent.
f	Hand orientation is below.
g	All of the first and the second joints are bent.
h	The thumb is fully extended.
i	The index finger is fully extended.
j	The middle finger is fully extended.
k	The ring finger is fully extended.
l	The pinky finger is fully extended.
m	The thumb is contact with the middle finger.
n	The ball of the middle finger is on the nail tip of the index finger.
o	The index finger and the middle finger are separated.
p	The thumb and the index finger are open.

joints are extended. Also, "hand orientation" means the direction of the fingertips from the wrist.

We perform preliminary experiments for the recognition rate at each conditional branch. A right-handed examinee puts her hand over a Leap Motion Controller to make the controller recognize her fingerspelling. The recognition rate is calculated

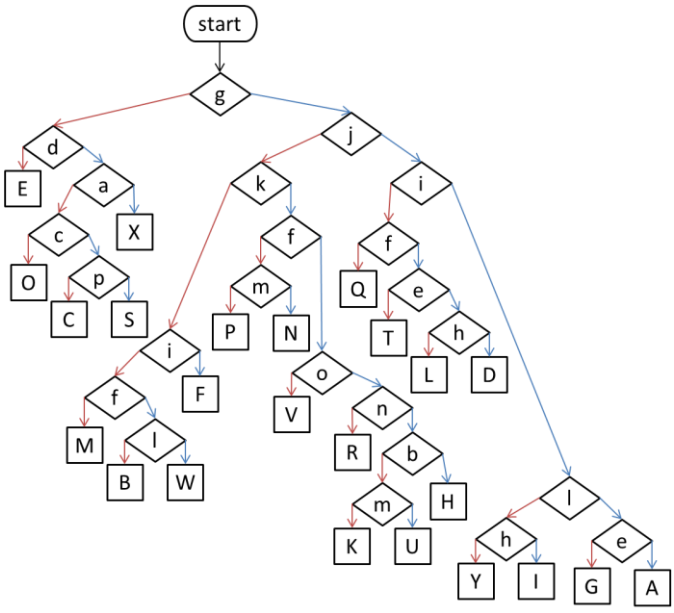


Figure 2 : An example of decision trees

Table 2 : The correct answer rate of each condition (%)

a	95.00	e	96.67	i	96.67	m	90.00
b	92.92	f	97.08	J	95.83	n	92.92
c	90.83	g	99.58	k	93.33	o	97.50
d	95.83	h	95.83	l	95.83	p	100.0

by 10 times experiments for all decision tree nodes.

The decision tree consists of nodes (non-terminal) and leaves (terminal). The node represents 16 conditional branches as shown in Table 1 while the leave represents fingerspelling 24 letters. Starting from the root (top node), there is a pass to get to each leaf. Each pass consist of one or more nodes. A leaf may have a lot of combinations of nodes to reach. However, depending on the combination of nodes that is required for each fingerspelling, the overall recognition rate is different. For example, when we want fingerspelling 'M', the node 'j', 'k', 'i' and 'f' are Yes and the node 'g' is No. By using the recognition rates of nodes 'j', 'k', 'i', 'f' and 'g' of Table 2, the recognition rate of fingerspelling 'M' in Figure 2 is calculated as $0.9583 * 0.9333 * 0.9667 * 0.9708 * 0.9958 * 100 = 83.58$. Calculating the recognition rates for other fingerspelling in the same manner, the average recognition rate of a decision tree shown in Figure 2 is 81.97%.

Note that there is a decision tree where a fingerspelling has multiple passes to reach the terminal node. Namely, there is a conditional branch whose decision does not affect the recognition result. In this case, the recognition rate of the conditional branch is 100%.

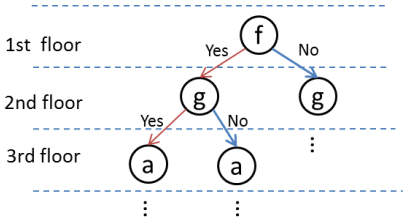


Figure 3: Structure of decision tree

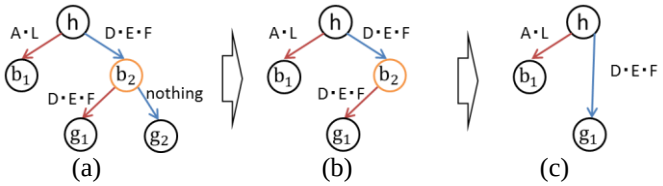


Figure 4: Unnecessary nodes: (a) shows a case including an unnecessary node. (b) shows a state deleting 'g_2'. (c) shows a state deleting 'b_2'.

3.3 Fingerspelling recognition algorithm

The structure of the decision tree is shown in Figure 3. The left arrow represents Yes while the right arrow represents No, where each node has a condition from "a" to "p". The depth of the decision tree to be generated is 17 with 16 layers for nodes (conditional branch) and 1 layer for leaves (final fingerspelling letter). At the root node of a decision tree, all the fingerspelling letters are candidates. When visiting each node, the candidates are sieved by the condition to select less candidates until the number of candidates becomes one, namely it is a leaf. The leaf represents a fingerspelling letter that is determined uniquely by the decision tree.

In the fingerspelling recognition algorithm, first 16 kinds of conditions are arranged in an appropriate order. This order corresponds to the hierarchical order of the decision tree. In other words, the nodes in the same hierarchy have a conditional branch from the same node. In addition, when generating a decision tree in some hierarchical order, it may include unnecessary nodes. The unnecessary node has a condition that all the fingerspelling letters from the parent node are sent to a child node without splitting the fingerspelling letters. Figure 4 is a diagram showing the process of deleting an unnecessary node. Figure 4 (a) shows a state including the unnecessary node (b_2). Node 'h' has two branches for the left side of node 'b_1' and the right side of node 'b_2'. In addition, the node 'b_2' has two branches for the left side of node 'g_1' and the right side of node 'g_2'. 'g_1' has 3 letters 'D', 'E' and 'F' while 'g_2' has nothing. Namely, 'g_2' is removable. Figure 4 (b) is the state after removing 'g_2'. In Figure 4 (b), since all the fingerspelling letters of 'b_2' from the parent node ('D', 'E' and 'F') are sent to the child node (g_1) without splitting the letters, 'b_2' is unnecessary. Figure 4 (c) is a state that 'b_2' is deleted. In this way, if unnecessary nodes are included, it is possible to shorten the processing of fingerspelling recognition by deleting them as shown in Figure 4.

Since there are 16 kinds of conditions, the possible number of decision trees to be generated is 16!, which is about 20 trillion ways. In addition, different decision trees have different recognition rates for fingerspelling. Therefore, it is necessary to obtain an optimum decision tree in consideration of the correct answer rate of Table 2.

4 Search the optimal fingerspelling recognition method

The conditions for fingerspelling recognition proposed in Section 3 are changeable by order. Since the correct answer rates in Table 2 are different, the average recognition rate for fingerspelling letters obtained in decision trees is changeable by the order of conditions. We propose an algorithm for searching the optimal solution for automatically generated decision trees with varying the order of 16 types of conditions.

As the optimal solution search for combinatorial optimization problem, there are a Branch and Bound method that gives a strict solution and a Genetic algorithm [10] that gives approximate solutions. In the case of Branch and Bound, although limited operations narrow the search space, it requires huge computation to search all over the limited search space. Therefore the amount of computation is unrealistic time consuming in order to obtain the result.

ID3 is known as a decision tree learning algorithm. It is for each independent variable with determining the average amaount of information and the expected value in the case of determining the value of variable. The largest variable is selected so that the operation of the variable to the node of the tree is performed recursively. In the case of ID3, creating a decision tree considering the average recognition rate is too difficult.

Using the Genetic algorithm, it is not possible to find the optimum solution because it gives approximate solutions while it is determined with less computation time to get solutions close to the optimal solution. We employ a Genetic alogorithm for the combinatorial optimization problem of 16 kinds of conditions to find the quasi optimal solution in realistic computation time. The constraints of the combinatorial optimization problem to be handled in this paper are the following.

1. 16 kinds of conditions are arranged so that the maximum average recognition rate of all fingerspelling is obtained.
2. Each condition is selected only once.

In this paper, we use a simple genetic algorithm with the most basic configuration. One point crossover and a roulette selection are adopted. The Order Representation [11] is used for the gene expression. Using the Order Representation, it is possible to satisfy the above constraint 2 as well as the one point crossover without causing lethal genes. The order Representation consists of the list L_1 where conditions are arranged alphabetically and the list L_2 where conditions are

arranged along a phenotype. The phenotype gives the order to perform the conditions. Each conditional branch destination of list L_2 is searched in list L_1 to be replaced, namely the initial conditions are removed from the list L_1. The above operation is repeated. Finally list L_2 becomes a list represented by a genotype.

Fitness is the recognition rate calculated with the automatic generation algorithm for fingerspelling recognition described in subsection 3.2 and 3.3. In the case of mutation operations, the i-th number from the beginning of the gene list by the Order Representation is rewritten by a number less than $N + 1 - i$ to avoid a lethal gene.

By the genetic and evolution operations, the individuals with high fitness to the target problem are increased. Finally, the fitness of the best individual becomes a suboptimal solution, and the genes of the best individual represent the optimal order for the conditions.

5 Experiments

5.1 Experimental method

We apply a Genetic algorithm to the automatically generated fingerspelling recognition algorithms described in subsection 3.3. To obtain suboptimal solutions, we perform three experiments: 1) Find appropriate crossover probability, 2) Find appropriate mutation probability, and 3) Obtain suboptimal solutions using 1) and 2).

In 1), the crossover probability is varied 0.7 to 0.9 by 0.05 while the mutation probability is fixed to 0.08. In 2), the mutation probability is varied from 0.02 to 0.1 by 0.02 while the crossover probability is fixed to 0.8. For both 1) and 2), the number of individuals and the maximum number of generations are 1,000 and 300, respectively.

Both 1) and 2) are performed 10 times for each parameter. In 3), the appropriate parameters obtained in 1) and 2) are used. 3) is performed 100 times with 1,000 individuals and 300 generations.

5.2 Experimental results and discussion

Figure 5 shows the fitness values of the best individual at the 300th generation with various crossover probabilities of 1). As the result of 1), the quasi-optimal solution is 82.71% achieved 3, 3, 5, 4 and 4 times out of 10 times trials with the crossover probability from 0.7 to 0.9 by 0.05, respectively.

Figure 6 shows the fitness values of the best individual at the 300th generation with various mutation probabilities of 2). As the result of 2), the quasi-optimal solution is 82.71%

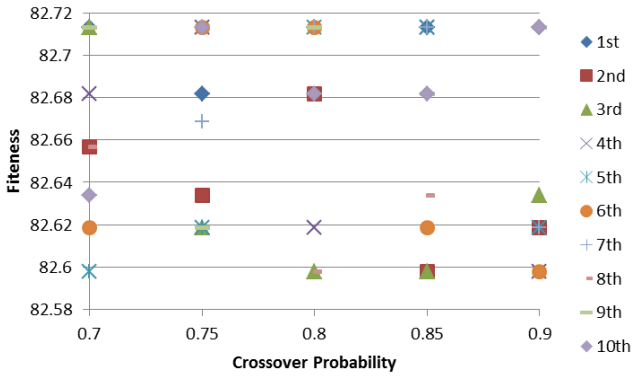


Figure 5: Fitness of the best individuals at the 300th generation with various crossover probability

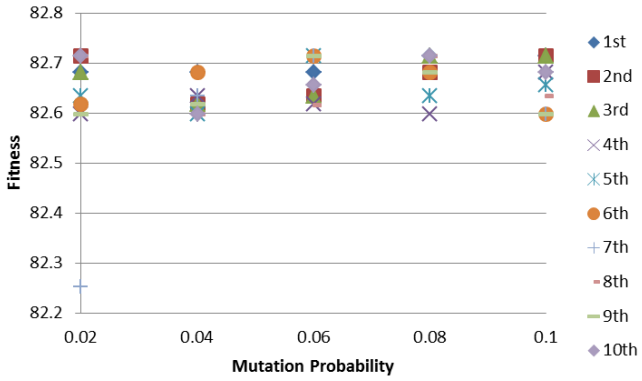


Figure 6: Fitness of the best individuals at the 300th generations with various mutation probability

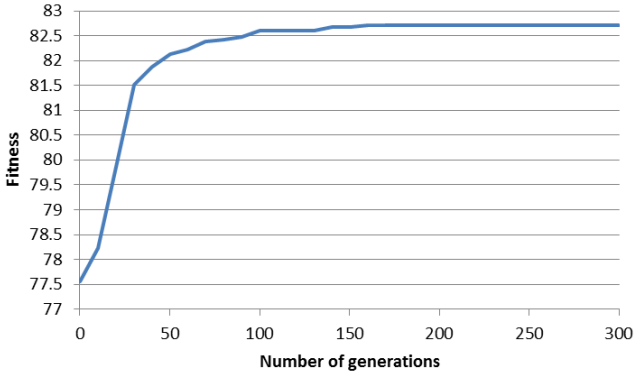


Figure 7: The fitness change by generation in the case of quasi-optimal solution.

achieved 3, 4, 5 and 3 times out of 10 times trials with the crossover probability 0.02, 0.06, 0.08 and 0.1, respectively.

The results of 1) and 2) show that the quasi-optimal solution for the fingerspelling recognition is 82.71%. So the

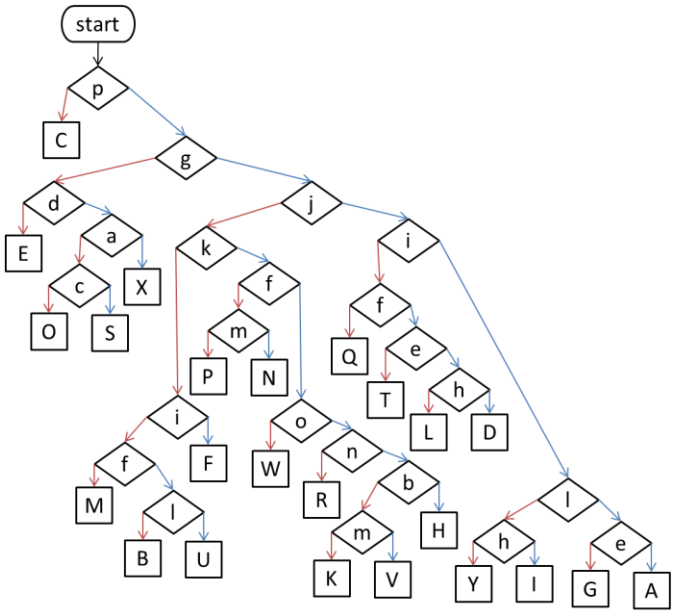


Figure 8 : An example of quasi-optimal solution of decision tree

optimum parameters for the genetic algorithm are the crossover probability 0.8 and the mutation rate 0.08.

3) is performed 100 times using the crossover probability 0.8 and the mutation probability 0.08. As the result, the maximum, average and minimum values of the fitness are 82.71%, 82.66% and 82.25%, respectively. The dispersion is 0.004. Also, 37 trials out of 100 reach the maximum value.

From the above results, by using the optimal parameter (the crossover probability is 0.8 and the mutation probability is 0.08) the quasi-optimal solution are stably obtained. Figure 7 shows the fitness changes by generation in the case of quasi-optimal solution 82.71%. The fitness converges at the 160th generation.

An example of the quasi-optimal solution is "p→g→j→k→i→f→d→o→n→b→l→m→e→h→a→c", and the decision tree is shown in Figure 8. By using the decision tree, it is possible to recognize the 24 letters with average recognition rate of 82.71%. In addition, the decision tree as the quasi-optimal solution can be generated in 3.4 minutes on average.

Although the quasi-optimal decision tree generation requires more than three minutes, the resultant decision tree can be used in any way. The real-time fingerspelling recognition for 24 letters is achieved using the resultant decision tree and Leap Motion Controller without any compute intensive processes such as image processing or neural networks. The real-time fingerspelling recognition is required for human interface by fingerspelling.

6 Conclusions

In this papeer, we propose the fingerspelling recogniton using Leap Motion Controller to give an alternative method of voice input for deaf people. The recogniton target is fingerspelling 24 letters exclude 2 fingerspelling letters that require finger movement.

As conditional branches to be used for the decision tree, we use 16 kinds of conditions that forcuse on the characteristics of hand and finger. By changing the order of the conditional branches, a different decision tree is generated with a different average recogniton rate for the fingerspelling 24 letters. The decision tree is automaticall generated by a Genetic algorithm to obtain quasi-optimal solutions.

We perform several experiments for the application of the Genetic algorithm and we obtain the quasi-optimal solution of the recogniton rate 82.71%. From the above, we validate the proposed method is very effective.

In the future work, it is necessary to include the fingerspelling two letters with movement. We think the decision tree should not be fixed. As a user makes use of the fingerspelling recognition, the recognition rates at conditional branches may change. In this case, some incremental learning mechamism should be performed.

7 References

[1] iOS8 Siri, Apple, https://www.apple.com/jp/ios/siri/ (last access: 2015-04-13)

[2] Google, https://www.google.co.jp/ (last access:2015-04-13)

[3] Marcus V. Lamar, Md. Shoaib Bhuiyan, Akira Iwata. "Hand alphabet recognition using morphological PCA and neural networksNeural Networks"; IJCNN '99. International Joint Conference on, vol.4, 2839-2844,(1999)

[4] Xbox 360 – Kinect, Microsoft, http://www.xbox.com/ja-JP/Kinect (last access: 2015-04-13)

[5] Leap Motion, https://www.leapmotion.com/ (last :access: 2015-02-03)

[6] Mischa Spiegelmock. "Leap Motion Development Essentials"; Packt Publishing (2013)

[7] C. S. Weerasekera, M. H. Jaward, N.Kamrani. "Robust ASL Fingerspelling Recognition Using Local Binary Patterns and Geometric Features"; Digital Image Computing: Techniques and Applications (DICTA), 2013 International Conference on, 1 – 8, (2013)

[8] Ching-Hua Chuan, Eric Regina, Caroline Guardino. "American Sign Language Recognition Using Leap Motion

Sensor"; Machine Learning and Applications (ICMLA), 2014 13th International Conference on, 541 – 544, (2014-12)

[9] Fingerspellingalphabet.com, http://www.fingerspellingalphabet.com/fingerspelling-chart-print-pdf-download/ (last access:2015-04-13)

[10] David E. Goldberg. "Genetic Algorithms in Search, Optimization, and Machine Learning"; Addison-Wesley Professional (1989)

[11] John J. Grefenstette, Rajeev Gopal, Brian J. Rosmaita, Dirk Van Gucht. "Genetic Algorithms for the Traveling Salesman Problem"; Proceedings of the 1st International Conference on Genetic Algorithms, 160 – 168 (1985)

Semi-supervised based learning for Idiopathic Interstitial Pneumonia on High Resolution CT images

Hayaru SHOUNO[1], Shoji KIDO[2]
[1] Graduate School of Informatics and Engineering, University of Electro-Communications,
Chofugaoka 1-5-1, Chofu, JAPAN
[2] Graduate School of Medicine, Yamaguchi University,
Tokiwadai 2-16-1, Ube, JAPAN

Abstract— *In the classification task, the number of labeled samples is one of the important factor for accuracy, however, gathering such data is hard work since it requires diagnosing task in the field of medical engineering, In order to overcome this problem, we introduce a semi-supervised learning (SSL) classifier for computer aided diagnosis (CAD) for idiopathic interstitial pneumonias (IIPs). The semi-supervised learning requires a lot of unlabeled training data, which does not require diagnosing cost, as well as labeled data. In this study, we show the low performance classifier, which has only chance level classification performance, would be improved to achieve around 90% accuracy performance by SSL. We also propose a pre-processing method of gray-scale transformation for appropriate application to the SSL. Without proper gray-scale transformation, the SSL might cause decreasing performance however, we find our pre-processing procedure make increasing the performance in almost all the cases.*

Keywords: Semi-supervised learning (SSL), Idiopathic Interstitial Pneumonia classification, gray-level transformation

1. Introduction

In the medical diagnosis, the classification task is important for the diagnosis quality. For classifying and detecting the idiopathic interstitial pneumonias (IIPs), high-resolution computed tomography (HRCT) image is regarded to be effective since IIPs affected part looks diffused in the lung [1][2][3][4][5]. Unfortunately, determining the border of the site is difficult work, because the IIPs on HRCT images show a lot of varieties in the meaning of texture patterns. The quality of diagnosis is influenced by the ability of physician, and improving the quality is desired for proper treatment of IIPs. In order to decrease the burden of physicians, development of the computer aimed diagnosis (CAD) system is desired for objective diagnosis in these decades [1][6][5]. The CAD systems are designed to provide a classification function for second opinion using machine learning techniques.

In the field of machine learning, the supervised learning is usually used for such classification task. and it requires pairs of input patterns and its corresponding labels for its learning. For improving the classification performance of such supervised learning system, a lot of labeled learning data is required, however, the obtaining cost of such data is expensive since it requires physicians diagnoses to get the proper labels. On the contrary, the cost for obtaining unlabeled data, which does not require physicians diagnosis, is lower than that of the labeled. The semi-supervised learning (SSL) uses massive unlabeled learning data as well as the labeled in order to improve the performance of classification accuracy [7].

The IIPs sites in the HRCT images are usually diffused in the lung, so that, we can obtain these unlabeled data easily by slightly shifting of the labeled region of interests (ROIs). Our purpose is to improve the accuracy performance of the CAD system for IIPs classification using the SSL by use of such unlabeled data. In this study, we try to develop and evaluate a classifying engine for IIPs in the CAD system.

2. Method

2.1 Semi-Supervised Learning system

In this study, we denote the input feature as a vector $\boldsymbol{x}$, and its desired label as t. The supervised learning data are denoted as pairs of the features and labels, $\{\boldsymbol{x}_n, t_n\}$, where n means the index. On the contrary, we use $\boldsymbol{x}^u$ as the unlabeled input feature, and denote the unlabeled data as $\{\boldsymbol{x}^u_m\}$ where m means the index.

The fig.1 shows the schematic diagram of a simple SSL, which is called "self-training" architecture proposed by Yarowski [8][9]. In the SSL, at first, a supervised classifier is trained by only use of labeled data $\{\boldsymbol{x}_n, t_n\}$. In the next step, the supervised classifier predict labels for the unlabeled

features $\{\boldsymbol{x}_m^u\}$ with beliefs, which mean the confidences for the labels of the classifier. Thus, the unlabeled data can be regarded as the labeled $\{\boldsymbol{x}_m^u, t_m^u\}$, where t_m^u is the label by the supervised classifier. After that, we drop off the low belief data by thresholding. The SSL classifier is trained by both the labeled data for supervised learning and classified unlabeled data with high beliefs.

This simple algorithm might be applied to the previous works for IIPs classification[5]. However, it is hard to evaluate of the genuine ability of the SSL by use of the complex learning system, so that, we apply a simple naive naive Bayes classifier in this study. Assuming the predicting label as y for the input feature $\boldsymbol{x}$, the Bayes classifier calculate posterior probability of $P(y \mid \boldsymbol{x})$ by use of the Bayes' theorem, that is, we can denote the posterior probability as

$$P(y \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid y)P(y)}{\sum_y P(\boldsymbol{x} \mid y)P(y)}, \tag{1}$$

where $P(\boldsymbol{x} \mid y)$ and $P(y)$ mean the likelihood and prior probability respectively. The prior $P(y)$ is defined as $P(y = k) = n_k/N$ where k means the class label and n_k means the number of labeled images belonging to the class k in the training set. N means the total number of the labeled images of training set $N = \sum_k n_k$.

The likelihood function $P(\boldsymbol{x} \mid y = k)$ is derived the multi-dimensional Gauss distribution:

$$P(\boldsymbol{x} \mid y = k) = \frac{1}{\sqrt{|2\pi\Sigma_k|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_k)^{\mathrm{T}}\Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{m}_k)}, \tag{2}$$

where $\boldsymbol{m}_k$ and Σ_k means average of feature vectors in the class k and corresponding covariance matrix respectively.

In the SSL, the Bayes posterior probability works as the beliefs for the unlabeled input feature $\boldsymbol{x}_m^u$. The class label for these unlabeled inputs are given by the supervised classifier with maximization of the posterior probability: $t_m^u = \mathrm{argmax}_k P(t_m^u = k \mid \boldsymbol{x}_m^u)$. In this maximization process, we obtain the probability value $P(t_m^u = k \mid \boldsymbol{x}_m^u)$ which means the confidence for the classification label, so that, we treat this value as the belief for the label $t_m = k$.

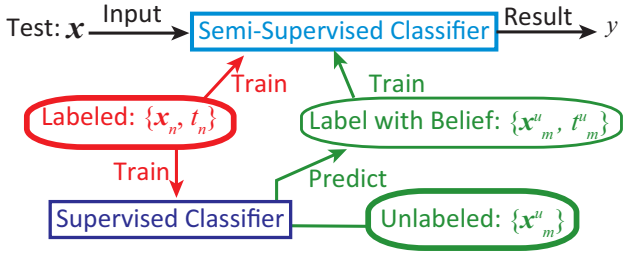


Fig. 1: Schematic diagram of the SSL architecture (Yarowski, 1995)[8], [9]

3. Experiments

3.1 Materials

In order to construct the SSL classifier, we prepare 360 labeled images and 3600 unlabeled images. In the labeled images, the number of each class are following: Consolidation(CON):38, Ground-Grass-Opacity (GGO):76, Honeycomb(HCM):49, Reticular(RET):37, Emphysema(EMP):54, Nodular(NOD):48, and Normal(NOR):58 cases. We assume the 32×32 [pixels] ROIs, and each ROI is segmented under the direction of a physician, and diagnosed by 3 physicians.

The acquisition parameters of those HRCT images are as follows: Toshiba "Aquilion 16" is used for imaging device, each slice image consists of 512×512 pixels, and pixel size corresponds to $0.546 \sim 0.826$ [mm], slice thickness are 1 [mm]. The number of patients is 69 males and 42 females with age 66.3 ± 13.4. The number of normal donor is 4 males and 2 females with age 44.3 ± 10.3. The origin of these image data is provided Tokushima University Hospital. Fig.2 shows a typical image example of each disease in HRCT image. The left shows an overview of the axial HRCT images of lungs including lesion, and the right shows segmented images of typical examples of lesion from the left image collections. The consolidation (CON) and ground-grass opacity (GGO) patterns are often appeared with the cryptogenic organizing pneumonia diseases (COPD). The GGO pattern is also often appeared in the non-specific interstitial pneumonia (NSIP). The reticular (RET) pattern which sometimes includes GGO patterns is also appeared in the NSIP. The honeycomb (HCM) pattern has more rough mesh structure rather than that of the crazy-paving, and it appeared in idiopathic pulmonary fibrosis (IPF) or usual interstitial pneumonia (UIP).

3.2 Labeled and Unlabeled dataset

From the 360 labeled ROI images, we define the labeled dataset as followings. The labeled ROI images have been carried out gray-level transformation in order to diagnose by physician. The raw HRCT image pixel value, which is counted with Hounsfield unit (H.U.), is adjusted to describe the physical matters, and its resolution takes 4096 grades in the range of $[-1024, 3071]$ [H.U.]. For example, air takes -1024 [H.U.], water takes 0 [H.U.], and over over 500 [H.U] shows bone typically. In order to diagnose the lung, which mainly occupied with air, the full resolution of pixel value is too much information for diagnose, so that gray-level transformation is applied as pre-process. The gray-level transformation for the raw HRCT image pixel value I is

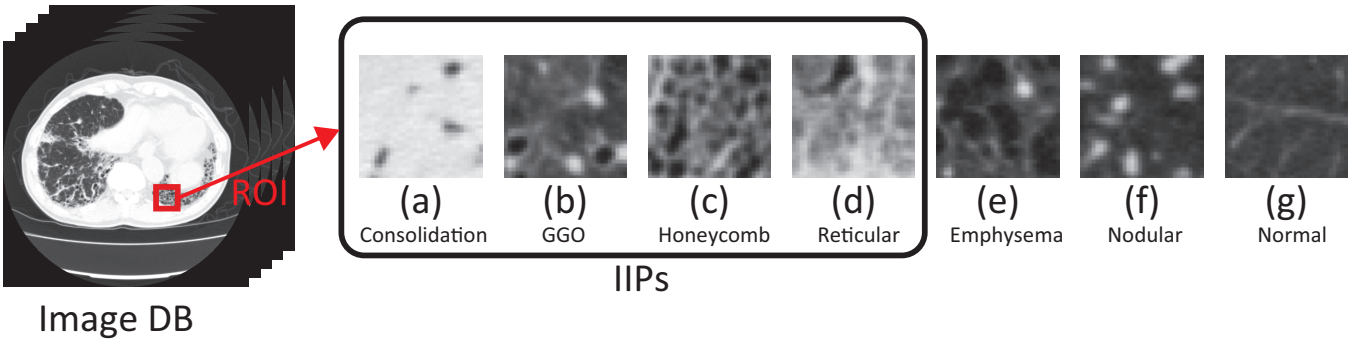


Fig. 2: Typical HRCT images of diffuse lung diseases: The top row shows each overview, and bottom shows magnified part (ROI) of each lesion. From (a) to (g) represents "Consolidation", "GGO", "Honeycomb", "Reticular", "Nodular" "Emphysema", and "Normal" image respectively.

described as piece-wise linear transformation:

$$q = \begin{cases} 0 & I < \mathrm{WL} - \frac{\mathrm{WW}}{2} \\ 255 & I > \mathrm{WL} + \frac{\mathrm{WW}}{2} \\ \frac{255}{\mathrm{WW}}\left(I - \left(\mathrm{WL} - \frac{WW}{2}\right)\right) & \text{else} \end{cases}, \quad (3)$$

where q means the 256 grades gray image value. Thus, the gray-level transformation is controlled by the window-level (WL) and the window-width (WW) parameters. Typically, these parameters are defined for diagnosing part such like lung, and sometimes adjusted by the physician manually. The labeled data has been processed by WL $= -600$[H.U.] and WW $= 1500$[H.U] in order to adjust the pixel values of the ROI images in $[0, 255]$ range.

The labeled dataset named as L is randomly selected from each class k ($k = 1 \cdots 7$, which means the class label) evenly, and we denote the total number of the labeled dataset L as N. This labeled dataset L is used for the supervised learning of the Bayes classifier.

We prepare 3 unlabeled datasets as follows. The first dataset U_1 is simply use of the rest of labeled data, which consists of $360 - N$ images. The dataset U_1 is used for control dataset against other unlabeled. The other unlabeled dataset U_2 and U_3 come from the raw HRCT images. These unlabeled candidates are gathered from the surrounds of labeled ROIs site, and the total number of collected unlabeled image data becomes $3,600$. In order to use these images for training data, we should carry out the gray level transformation 3 as a preprocessing. In our gray level transformation, we assume the transformation parameters WW and WL are not given since these parameters, which are adjusted manually, are not rigid even in the labeled data. Thus, we should infer these parameters, and details are followings. The pixel of the raw HRCT image has 16-bit depth in this case and the range of unlabeled data pixels values are in $[-1152, 7281]$.

Fig. 3 shows the average gray-level histograms of labeled and unlabeled data. The parameters WL and WW for dataset U_2 are defined by these averaged histograms. We optimize the parameters WW and WL as to maximize the similarity between the labeled and transformed unlabeled histograms. We denote the average histogram of labeled as q_L, which can be regarded as a probability distribution. We also describe the transformed unlabeled histogram as the function of the parameters WW and WL, that is, $q_U(\mathrm{WW}, WL)$. Then, we introduce Kullback-Leibler (KL) divergence as a similarity measure between the gray level histograms q_L and $q_U(\mathrm{WW}, \mathrm{WL})$:

$$\mathrm{KL}(q_L \mid q_U; \mathrm{WW}, \mathrm{WL}) = \sum q_L \ln \frac{q_L}{q_U(\mathrm{WW}, \mathrm{WL})}. \quad (4)$$

We adopt WW and WL as the minimization values of the $\mathrm{KL}(q_L \mid q_U)$. From the strategy, we optimize eq.4 and obtain the parameters WW $= 1234$[H.U.] and WL $= -434$ [H.U.] for the unlabeled data.

For comparison, we prepare the other unlabeled dataset name U_3 whose WW and WL are chosen as a typical values to observe lung-area, that is WW $= 1500$[H.U.] and WL $= -550$ [H.U.].

3.3 Feature Extraction and Selection

We introduce a texture analysis proposed by Sugata *et al.* for feature extraction [1][10]. From the input HRCT ROI image, we calculate gray-level histogram, gray-level difference statistics, the co-occurrence matrix, run length matrix, and Fourier power spectrum, at first. After that, from these 5 quantities, we derive 39 texture statistics as the candidates for features[10]. Using whole statistics candidates as the input features for classifier might cause the decreasing the performance because of "curse of dimensionality". Thus,

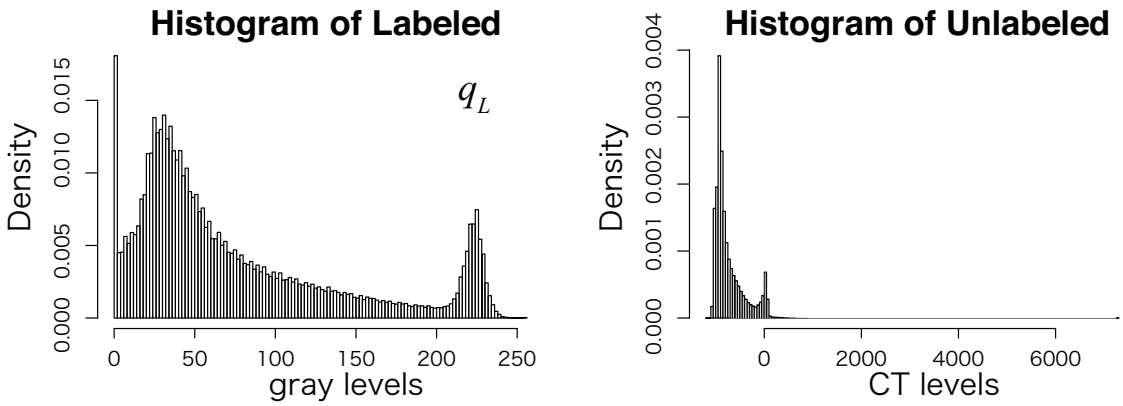


Fig. 3: Gray-level histograms of labeled and unlabeled images. The left shows the labeled which have the range $[0, 255]$. The right shows the unlabeled that have raw-HRCT values with the range $[-1152, 7281]$.

we would select the 4 input features as the input for the classifier. These feature are determined experimentally.

3.4 Evaluation Method

In order to evaluate performance, we adopt leave-one-out cross-validation (LOOCV) [11][12]. In the LOOCV method, we choose a ROI image from the labeled dataset L and use other $N-1$ images for supervised learning. After supervised learning, M unlabeled images from the unlabeled dataset U_1, U_2, or U_3 is used for the self-training method. The preserved ROI image is used for evaluating the classification performance. This evaluation process is applied alternate to the whole labeled dataset L, and finally the average accuracy is used for the performance measure.

For the performance evaluation, we carry out the SSL method as follows:

1) We trained Bayes classifier (1) by supervised learning that is we apply only the labeled data. For LOOCV method, we pulled out a pair of training datum from labeled dataset. Then, we calculate the mean vector $\boldsymbol{m}_k$ and covariance matrix Σ_k for the pulled dataset.
2) We predict the class label for the unlabeled dataset with confident that comes from the posterior probability. The largest posterior class is to become the predicted class.
3) By thresholding, we drop out the low confident unlabeled data. We adopt the threshold value as 0.80 in this experiment.
4) From the rest of the unlabeled data, which are high confident unlabeled data, we select M images randomly.
5) We train the Bayes classifier (1) with both labeled and the M predicted data again
6) We evaluate the classifier accuracy by the selected labeled data pair in the procedure 1 (LOOCV method).

4. Results

We compare the accuracy performances among the added number of unlabeled data from U_1, U_2, and U_3 whose differences are gray-level transformation parameters of WW and WL. The dataset U_1 has identical statistical property since it comes from labeled data. The dataset U_2 might have similar property to the labeled data in the meaning of the KL-divergence. The dataset U_3 might have the most different property to the labeled data, however, it is typical parameters for observing lung areas. Fig.4(a) shows the accuracy performance against the added number of the unlabeled images M. The horizontal axis shows the number of the unlabeled images M in log-scale. The vertical one shows the accuracy performance. The size of labeled dataset L is $N = 35$. Adding small number of unlabeled data ($M \sim 100$) increase the accuracy performance in the every unlabeled dataset U_1, U_2, and U_3. The number of dataset U_1 is $360 - N$, so that, the curve ends in the value with high accuracy performance. The curve using the U_2 saturate around $M \sim 1000$ with also high accuracy performance. However, the curve using the dataset U_3 saturates in the low accuracy performance, while U_1 and U_2 increase the accuracy performance. In this case, the final accuracy performances for U_1, U_2, and U_3 datasets are 96.6%, 95.8%, and 61.1% respectively.

Fig.4(b) shows also the accuracy performance under the larger labeled dataset L that have $N = 70$. In this case the initial accuracy performance, which comes from the supervised classifier, is higher than the previous result, that is, the accuracy performance is 84.5% correct. We can see the similar tendency to the previous result in the meaning

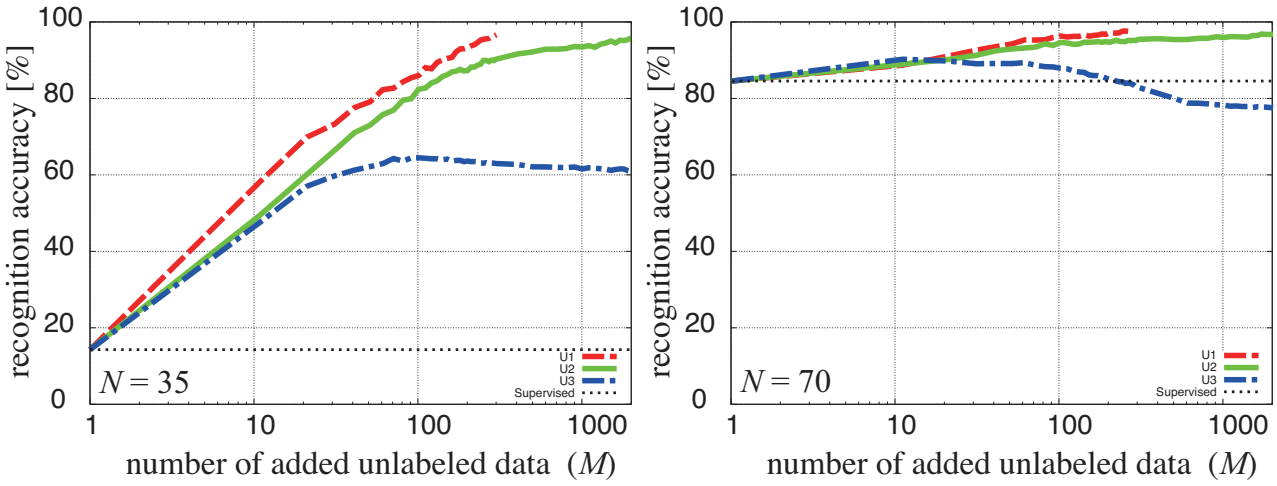


Fig. 4: Accuracy performance for several unlabeled dataset U_1, U_2, and U_3. The horizontal axis shows the number of added unlabeled images denoted as M. The vertical shows the accuracy performance. (a): The left shows the result with the supervised classifier that is trained by $N = 35$ samples. (b): The right shows the result for the classifier trained by $N = 70$ labeled samples.

of the performance improvement, while the curve using U_3 decrease the its performance. In this case, the final accuracy performances for U_1, U_2, U_3 dataset are 97.5%, 96.8%, and 77.5% respectively.

From these results, statistical similarity between the labeled image and the unlabeled images is important factor to the accuracy performance.

5. Conclusion & Discussion

We investigate classification performance of the SSL for the classification task of the IIPs. We can confirm increasing of the accuracy performance in several cases while the self training method is a simple method in the SSL.

We found several important factors for the IIPs classification by the SSL. One is the statistical quality of the features, that is, the gray-scale histograms of unlabeled images should have similar property to that of the labeled images used in the supervised learning. Fig.5 shows the scatter plot for 2-dimensional features space. Each horizontal axis shows the average of gray-level of the ROIs, and vertical shows the entropy of cooccurance matrix. The left figure shows the plot for whole labeled data that have $N = 360$ points. In the SSL, we assume we can obtain only a part of the labeled data, so that, randomly selected points, in which $N = 35$, are shown in the middle and right figures. The unlabeled data, which should be processed by the gray-scale transform, are overlapped to the middle and right figures. Hence, the middle one shows the unlabeled dataset U_2 with labeled samples, and the right shows the U_3 with labeled. We can see the U_2 dataset looks better fit to the labeled dataset. These figures suggest the optimization parameters WW and WL is an important factor. The unlabeled dataset U_2 optimized for the minimization of the KL-divergence between gray-level histograms of labeled and unlabeled. This result suggests that unlabeled data that come from another HRCT device might be available when we carry out appropriate pre-processing.

Moreover, we evaluate several trials for the another labeled dataset L, and stable improvement is confirmed by the SSL. When the initial supervised learning make good accuracy performance, the SSL improvement does not work well, however, it does not cause cause adverse affect in this investigation. Thus, we can consider the SSL is a good framework to improve classification ability for our task.

Acknowledgment

We thank Professor Junji Ueno, Tokushima University. He provided several advice for this study as well as a set of high resolution HRCT image of IIPs. This work is supported by Grant-in-Aids for Scientific Research (C) 25330285, and Innovative Areas 26120515, MEXT, Japan.

References

[1] R. Uppaluri, E. Heitmman, M. Sonka, P. Hartley, G. Hunninghake, and G. Mclennan, "Computer recognition of regional lung disease patterns." *American Journal of Respiratory and Critical Care Medicine*, vol. 160, no. 2, pp. 648–654, 1999.

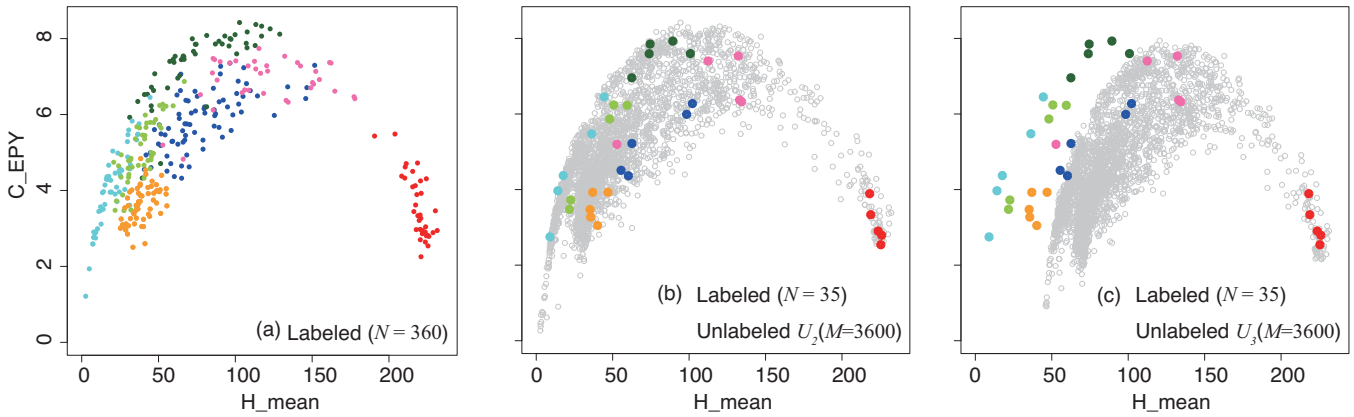


Fig. 5: Scatter plot for the 2 dimensional feature space. The horizontal axis is the mean of the histogram, and the vertical is the entropy of the co-occurence matrix. (a) shows the scatter plot of whole labeled samples ($N = 360$). (b) shows the dataset L which have $N = 35$ samples, and whole unlabeled samples of U_2 (M=3600). (c) shows the dataset L with U_3 (M=3600).

[2] H. U. Kauczor, K. Heitmann, C. P. Heussel, D. Marwede, T. Uthmann, and M. Thelen, "Automatic detection and quantification of ground-glass opacities on high-resolution CT using multiple neural networks: comparison with a density mask," *AJR Am J Roentgenol*, vol. 175, no. 5, pp. 1329–1334, Nov 2000.

[3] W. Webb, N. L. Müller, and D. Naidich, *High Resolution CT of the Lung, 4th edn.* Baltimore: Lippincott Williams & Wilkins, 2008.

[4] "American Thoracic Society/European Respiratory Society International Multidisciplinary Consensus Classification of the Idiopathic Interstitial Pneumonias. This joint statement of the American Thoracic Society (ATS), and the European Respiratory Society (ERS) was adopted by the ATS board of directors, June 2001 and by the ERS Executive Committee, June 2001," *Am. J. Respir. Crit. Care Med.*, vol. 165, no. 2, pp. 277–304, Jan 2002.

[5] R. Xu, Y. Hirano, R. Tachibana, and S. Kido, "Classification of diffuse lung disease patterns on high-resolution computed tomography by a bag of words approach," in *MICCAI*, vol. 14. Springer-Verlag Berlin Heidelberg, 2011, pp. 183–190.

[6] I. Sluimer, A. Schilham, M. Prokop, and B. Ginneken, "Computer analysis of computed tomography scans of the lung: A survey." *IEEE Transactions on Medical Imaging*, vol. 25, no. 4, pp. 385–405, 2006.

[7] Z. Xiaojin, "Semi-Supervised learning literature survey," Computer Sciences, University of Wisconsin-Madison, Tech. Rep., 2005. [Online]. Available: http://www.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf

[8] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," in *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, ser. ACL '95. Stroudsburg, PA, USA: Association for Computational Linguistics, 1995, pp. 189–196. [Online]. Available: http://dx.doi.org/10.3115/981658.981684

[9] G. Haffari and A. Sarkar, "Analysis of semi-supervised learning with the yarowsky algorithm," in *UAI*, R. Parr and L. C. van der Gaag, Eds. AUAI Press, 2007, pp. 159–166.

[10] Y. Sugata, S. Kido, and H. Shouno, "Comparison of two-dimensional with three-dimensional analyses for diffuse lung diseases from thoracic ct images," *Medical Imaging and Information Sciences*, vol. 25, no. 3, pp. 43–47, 2008. [Online]. Available: http://ci.nii.ac.jp/naid/130000097652/en/

[11] M. Stone, "Cross-validation: A review." *Math.Operations.Stat.Ser.Stat*, vol. 9, no. 1, pp. 127–139, 1978.

[12] C. M. Bishop, *Pattern Recogition and Machine Learning*. Springer, 2006.

An Effective and Interactive Training Data Collection Method for Early-Modern Japanese Printed Character Recognition

Kazumi Kosaka, Taeka Awazu , Yu Ishikawa , Masami Takata, and Kazuki Joe
Dept. of Advanced Information & Computer Sciences, Nara Women's University, Nara, Japan

Abstract *– In this paper, we present a web application that supports to collect training data efficiently for early-modern Japanese printed character recognition. The national diet library in Japan provides a lot of early-modern (AD1868-1945) Japanese printed books to the public, but full-text search is essentially impossible. In order to perform advanced search in historical literatures, it is required extracting texts from images. To solve this problem, we have already proposed a multi-font Kanji character recognition method using the PDC feature and an SVM. For growing in performance of this method, we need big amounts of training data. However, collecting training data by hand is extremely inefficient. Therefore, we propose a Web application that supports collecting training data.*

Keywords: early-modern Japanese printed books, a multi-font Kanji character recognition method, Web-application

1 Introduction

The National Diet Library (NDL) [1] in Japan keeps about 35,000,000 books dating from the Meiji era to the first half of Showa era (AD1868-1945). The books cover a broad range including philosophies, literatures, histories, technologies, natural sciences, etc. Most of them are out of print and valuable materials in scholarly. The NDL started a project called "The Digital Library from the Meiji Era" from 2002. In the project, early-modern Japanese printed books are recorded on microfiches page by page. The microfiches are converted into digital images and viewable at the project Web site [2]. Converting the books into digital images enabled the valuable books to be opened to the public while they are preserved in good condition. Users can view the digital images of the books whenever or wherever with the Internet connection. At the NDL Web site, user can search the materials by setting up detailed items, such as title, author, publisher and publication year. However, since the text of early-modern Japanese printed books is exhibited as picture image, full-text search is essentially impossible. In order to perform the full-text search, it is required extracting texts from images. The information including titles and author names of the books in the Digital Library is given as text data while main body is picture image data. There are no functions for generating text data from the image data. Thus full-text search is not supported yet. To make early-modern Japanese printed books data more accessible, their main body should be presented as text data, too. As described above, although there are considerably many early-modern Japanese printed books that are academically precious, the text extraction of hundreds of thousands of books is impossible in budget if it is performed by hand. There is no existing research on early-modern Japanese printed character recognition. Based on the above backgrounds, we collaborate with the NDL to have started the research project of automatic text extraction for 'Digital Library from the Meiji Era'. In extracting text data from image data of early-modern Japanese printed books, when commercially available OCRs are applied to the image data, the recognition rates are too low to be practical. We have reported that the recognition of early-modern Japanese printed characters is possible using a method of handwritten Kanji character recognition [3][4]. In early-modern Japanese printed books, it is reasonably inferred that each publisher adopts a different typography. Even if within the same publisher, we have reported that typography differs by age [5]. For these reasons, we use the method of handwritten Kanji character recognition for text extraction of early-modern Japanese printed books. We have proposed a multi-fonts Japanese character recognition method for early-modern printed books. In this method, the Support Vector Machine (SVM), which is one of promising recognition methods, is used for the recognition of feature vectors. The fonts used in the early-modern Japanese printed books vary by publisher and year of publication. To recognize image data of early-modern Japanese printed books exhaustively, training data for SVM must increase. However, it is too heavy burden for persons to collect various characters as training data by hand. Thus, in order to collect data samples efficiently, we propose a Web application based method for early-modern Japanese printed character recognition that collects training data efficiently and interactively. The process of character recognition and adding new characters to the database as training data is performed automatically. Thus, what users operate is just to correct wrong recognition results which the web application presents, therefore the burden of users becomes very light. The web application enables plurality of users to access to collect training data simultaneously.

The rest of the paper is organized as follows. In section 2, we describe tendency of Kanji included in books. In section 3, we

introduce the multi-fonts Japanese character recognition method for early-modern printed books. In section 4, we present a web application based method for early-modern Japanese printed character recognition. In section 5, we evaluate the experiment results to compare the results by the web application with by hand, and discuss the web application availability.

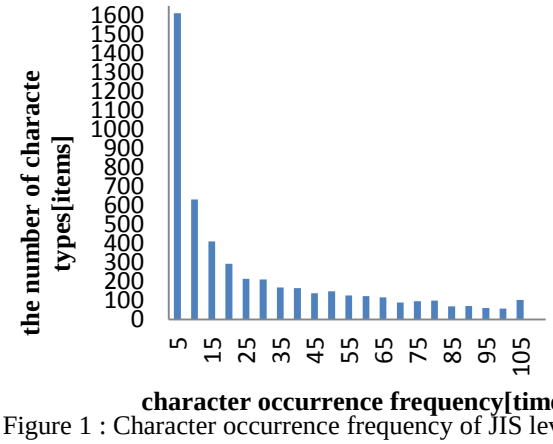


Figure 1 : Character occurrence frequency of JIS level-1 and level-2 kanji set

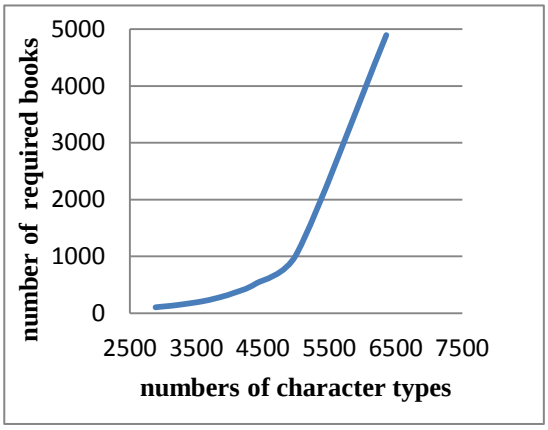


Figure 2 : Number of character types and required books

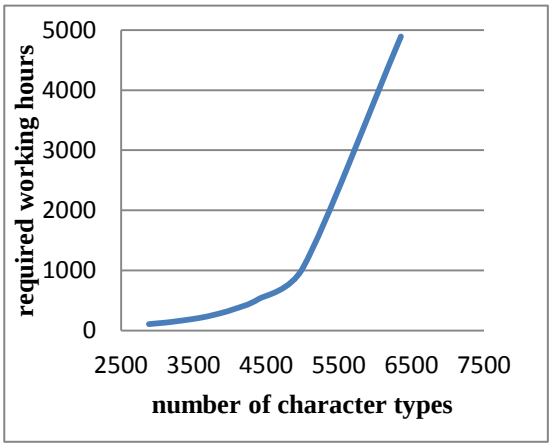


Figure 3 : Number of character types and required working hours

2 Tendency of Kanji occurrence in books

In order to collect training data, the number and the type of Kanji included in the target books should be investigated in advance. However, early-modern Japanese books in the Digital Library are not presented with text data. On the other hand, many titles in text format are published on the web in the case of Aozora Bunko [6]. Most of them are included in the books found in NDL. Therefore, the Aozora Bunko can be substituted for the Digital Library in NDL in the case that we investigate the tendency of Kanji occurrence in early-modern books. The Aozora Bunko is a Web based online collection of the public domain or copyright licensed literary works in Japanese since 1997. Users can download the works as text data. As the result of the investigation, the Aozora Bunko shares 105 books in common with the Digital Library from the Meiji era. The number of the total Japanese characters and their occurrence frequency are 5,203,045 and 144,076, respectively. The number of JIS level-1 to level-2 Kanji set [7] and their occurrence frequency are 1,569,439 and 131,080, respectively. The total number of JIS level-1 to level-2 Kanji types is 5,009, where JIS level-1 to level-2 Kanji set contains 6355 types.

Figure 1 shows the number of character occurrence frequency of JIS[1] level-1 to JIS level-2 kanji set. In Fig.1, 115 Kanji types appear in 100 to 105 books while 2,125 Kanji types just appear in 1 to 10 books. We confirm that the appearance is biased. Therefore, a lot of books are required in order to collect various Kanji types.

From our past studies, we temporally assume that the number of training data for each Kanji type should be at least 10. Figure 2 shows the number of Kanji types and the estimated number of required books to collect at least 10 types. In Fig.2, more than 4,000 books are needed to collect all 6,355 types which are JIS level-1 to level-2 Kanji set. In order to collect about 110,000 Kanji characters from the 105 books, 7 persons work 45 days. The total working time is about 945 hours. It may be presumed that around 1,000 per page. Figure 3 shows the number of character types collected manually and required working hours. In Fig.3, it takes too much time to collect required characters by hand because types and the number of characters spread over a wide range in books.

3 Multi-fonts Japanese character recognition

We use our multi-fonts Japanese character recognition method for early-modern printed books [3][4][8]. The operation process of the multi-fonts Japanese character recognition for early-modern books consists of preprocessing, character clipping, PDC (Peripheral Direction Contributively, PDC) [9] feature extraction, and the learning phase of SVM [10]. In this section, we explain this method in detail.

[1] Japanese Industrial Standards

3.1 Pre-processing

First, we apply binarization, noise reduction, angle correction and ruby removal to the target image as pre-processing.

Information about the density of the character is unnecessary for recognizing the character shape. Thus, binarization should be adopted to get the minimum image information.

Noise reduction should be also applied since the printing and archiving quality of early-modern printed books is mostly poor. Therefore, we apply noise reduction. Noise reduction prevents that noises on a character image are erroneously recognized as wrong character lines.

Most of early-modern Japanese printed books are out of print and valuable materials in scholarly. It is impossible to press such books against the scanner for converting the books into digital images. Thus, once a book is opened, then the opened page is recorded as a photo image. Therefore, deflection of the page may occur. In order to eliminate the displacement of the deflection, angle correction is performed to modify the distorted page angle.

Next, fonts and the ruby system found in early-modern Japanese printed books are explained.

Japanese letter consists of three kinds of characters: Hiragana, Katakana, and Kanji. The first two character sets are syllabaries and include about 70 phonetic characters while the last character set is an ideogram and include more than six thousands characters. Sentences in Japanese books are usually printed vertically using the above three character sets. Ruby is a system for low educated Japanese to support pronouncing difficult Kanji characters written in the books by locating small Hiragana or Katakana characters at the right side of difficult Kanji characters. The small phonetic characters and the (big) Kanji characters are called as ruby characters and parent characters, respectively. Most Japanese books adopt the ruby system, and currently most books are generated in a desktop publishing system, where the standard of fonts and ruby is defined by JIS. Since early-modern Japanese printed books are published before the standard with typographical printing, various fonts and ruby systems are used. When a ruby system is used in a book, ruby characters are located at the right side of the corresponding parent Kanji character so that the outer frames of the ruby characters contact with the outer frame of the parent Kanji character. In fact, ruby characters are often connected with their parent Kanji characters with the bleeding ink and likely to cause noise that disturbs recognition of early-modern Japanese printed books. Therefore we developed a new ruby removal method. We apply the method that takes notice the ratio between width and height of Kanji characters to ruby character removal [8]. It is calculated with the averages of widths and heights of Kanji characters in each row as (1).

$$f_1 = \frac{\text{width include ruby}}{\text{heights}} \quad (1)$$

The formulas generated by genetic programming using the averages are used for the ruby removal.

3.2 Character Clipping

We clip a character from the digital images performed in the pre-processing in subsection 3.1. At first, a circumscription rectangle is generated by labeling strongly connected black pixels by eight direction neighborhoods. Then, the character divided with several small rectangles is segmented by the circumscription rectangle generated with duplicated small rectangles. A Japanese character consists of several strongly connected components in many cases. Because of separated components in multiple, it is difficult to clip a character just using the circumscription rectangle. However, we can clip such a character by integrating the components. The conditions for the integration are as follows.

- The distance between two vertically located components is 0.3 times less than the vertical distance between the corresponding two adjacent characters.

- The distance between two horizontally located components is 0.2 times less than the average width of the characters in the row.

The average width is explained in subsection 3.1.

3.3 Peripheral Direction Contributivity feature

Peripheral Direction Contributivity (PDC) feature [9] is one of very efficient features for the recognition of Kanji characters written carefully in a standard style by hand. PDC feature reflects four statuses of character-lines: complexity, direction, connectivity and relative position.

The complexity of character-lines is represented by line density. The direction and the connectivity are represented by direction contributivity. The direction contributivity is given as a four dimensional vector, which is calculated by dot of character bit-map image. d_p is defined as $d_p = (d_{1p}, d_{2p}, d_{3p}, d_{4p})$. Each element d_{mp} (m=1,2,3,4) is defined as

$$d_{mP} = \frac{l_m + l_{m+4}}{\sqrt{\sum_{j=1}^{4}(l_j + l_{j+4})^2}} \quad (2)$$

where *li (i=1,2,...,8)* is the length of connected black dots scanned for eight directions.

3.4 Recognition using SVM

In the Multi-fonts Japanese character recognition, the PDC feature is adopted for an SVM. Though the amount of

calculation has relatively light with simple principles, it has shown good results in various fields.

In [11], the training data for the SVM has been collected with 2,237 types among 2,965 types of JIS level-1 Kanji set and 720 types among 3,330 types of JIS level-2 Kanji set. Recognition experiment with a set of 1,000 types has been performed. The 1,000 types set includes JIS level-1 Kanji with five or more training data. The recognition rate of Kanji with 9 or more training data is about 99%.

4 The web application for data collection

To convert book images in the Digital Library, recognition accuracy of the multi-fonts Japanese character recognition method for early-modern Japanese printed books must improve. Thus, it is necessary to efficiently collect the training data for SVM. Therefore we develop a Web application for collecting training data. The operation of the

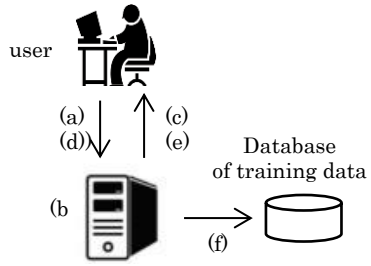


Figure4 : Flow of a Web application for data collection

Web application should be easy for the general users.

4.1 Configuration of the web application

Figure 4 shows a flow diagram of the Web application. The enumerated alphabets in Fig.4 show the operation procedure of the application. Figure 5 shows the flowchart of the user interface to operate the Web application in Fig.4 (a) and (d). Figure 6 shows the initial screen of the Web application. Figure 6(a) is the title of the Web application called 'the Web application of data collection support for early-modern Japanese printed character recognition. Figure 6(b) shows user instructions. In Fig.4 (a), users specify the NBN (national bibliography number)[12] as in Fig.6(c) and select the scope of the book they want to recognize in Fig.6 (d). The NBN is the number that National Diet Library assigns based on the Legal Deposit System, and posted on the national bibliography of countries. The national bibliography of countries is issued by country. Users access to recognize in Fig.6 (e). In Figure 4(b), the image data of the specified books to be recognized is loaded and applied the recognition method described in Section 3 in the web application. In Fig.4(c), the web application displays a set of a recognition result and the original image side by side. Users confirm whether the recognition result has been recognized correctly by referencing the original image. In Fig.4 (d), in the case of false recognition, users correct the result. The users can also correct using the IME pad. In Fig.4 (e), the corrected result character is added to the database of training data as new training data. In this way, it is possible to easily increase the number of training data when the web application is used, which leads to the higher recognition rate.

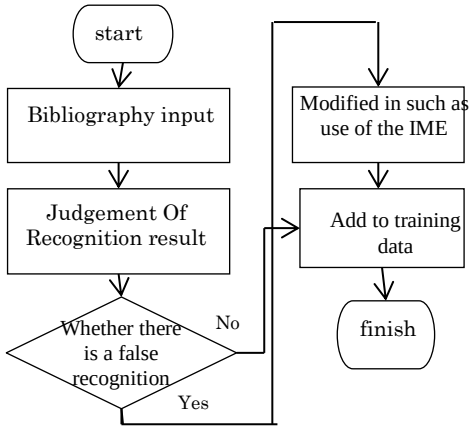


Figure 5 : Flow chart of user interface

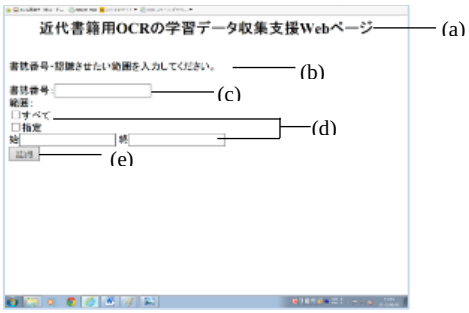


Figure 6: Initial screen of web

4.2 External and internal design

Figure 7 shows the internal design of the web application. Each step is developed with various languages. The processes of character clipping, extracting PDC feature, SVM, and the access to the database are developed with C/C++, Java, Python, and Java, respectively. The web application is developed in the servlet format that is written in Java. By the servlet format, the all processes can be performed on the server side. The processing speed is faster so the burden of users is very limited. The web application uses the Apache Tomcat servlet container 7. The web server software in conjunction with Apache Tomcat is Apache HTTP Server 2.0 so that it corresponds to the Apache Tomcat 7. The web application performs the processing with shared as dynamic content processing is performed with Apache Tomcat and static content processing is performed with Apache. The format of image data is JPEG2000 so that it is used with any suitable format according to the process without conversion problem.

First the web application loads each of spread pages. Next the image data is converted to the bmp format from the JPEG2000 format and isolates one by half a page. Then, the half page image is divided by line. The line image is saved in the pbm format. The process of ruby removal requires the conversion of the pbm format into the pgm format. Since the image format required for extracting PDC features is the bmp format, characters clipped from the images are saved in the bmp format to pass to the process of PDC feature extraction.

We use LIBSVM2.81[13] which is an SVM library for the process of SVM identification. The LIBSVM is invoked from a Python program. The web application needs to call the Python program from Java because some processes are written in Java. Therefore, we use a reference implementation

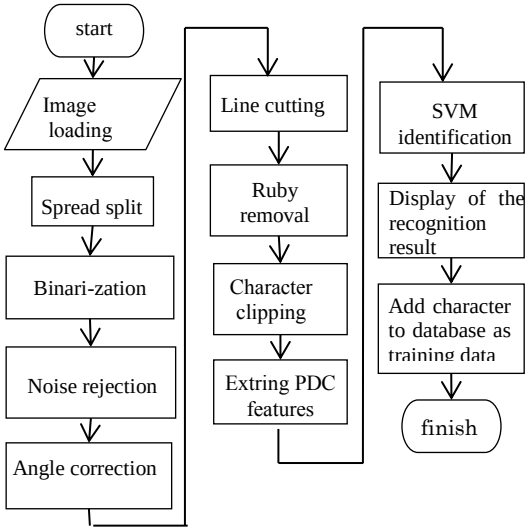


Figure 7: Internal design

of Python called Jython. Java with Jython can operate Python in Java codes. It is released as a free license under the Python Software Foundation and we use Jython 2.5.2 because LIBSVM2.83 is suitable for Python supported by Jython 2.5.2.

The database of training data is configured with three items: number, character and PDC features. The number is assigned to Hiragana, Katakana and Kanji characters from JIS level-1 to level-4 kanji set beginning from 1. Otherwise, the number is set to 0. We use MySQL5.5.38 for the database. After extracting the PDC feature, a process in the SVM accesses the database for training data. The result of the SVM process is represented by numbers assigned in the database. The recognition result is converted to a character corresponding to each number. Then the web application displays the recognition result as a Japanese character and the original image of the half page on the screen side by side. Thus, the

Figure 8: The screen of the recognition result for modification

user can easily find the wrong recognition by just looking at them. Figure 8 shows a display screen of a recognition result.

In order to correct false recognition results, a file displayed as the result is downloaded directly from the server as shown

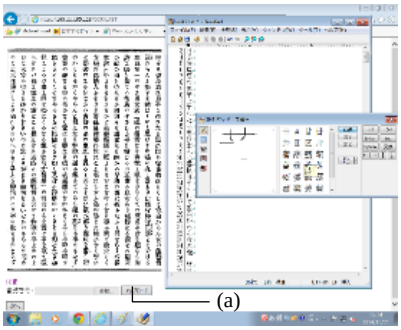


Figure 9 : The edit screen

in Fig.8 (a). When users do not know how to read a displayed (correct) Japanese character, an IME pad can be used; the IME pad corresponds to JIS level-1 to level-2 kanji set. It is not necessary for users themselves to search unknown characters by using the IME pad. The IME pad reduces the burden of users. Figure 9 shows a screen that users are correcting a false recognition by using the IME pad. Users upload the file that they finish modification to the server in Fig.9 (a). This process is the last operation. All the operations are performed with the buttons on the web. When users finish the corrections in all of the specified range, the screen displays just a button to add them to the training data in the database.

The web application loads the files that are uploaded to the server and the database of training data is finally updated by click the confirmation button. The web application displays a message that operations are completed.

Table 1 : Comparison of processing times

Process (times) (hour:h,minute:m,second:s)	Web application	hand
Process	5.22s	15h27m
Extracting PDC features	1.76m	7m
SVM identification	8.42m	61.2m

Table 2 : Classified matching rate of the web application

Comparison	Number	Mismatch	Matching rate
(a)	1140	191	83.2%
(b)	934	69	92.8%
(c)	304	8	97.5%
(d)	630	61	90.4%
(e)	189	105	35.6%
(f)	17	17	0%

5 Experiments

In order to confirm the utility of the web application, we compare the cases of collecting training data by hand and by the web application. For comparison, we use books published

in 1883. The average number of characters per page is 1,140. We use a PC with Core i7-4770K @ 3.50GHz CPU and 16GB memory.

5.1 Comparison results

Table 2 shows the processing times to collect training data by hand and by the web application. Using the web application, the operation times of the image pre-processing, PDC feature extraction, and SVM identification are about 9.4×10^{-5}, 1.4×10^{-1}, and 2.5×10^{-1} times compared with the hand collection times, respectively. It is considered to be very efficient because the application significantly reduced not only calculation times but also physically burden of users. The operation of collecting training data by hand takes approximately 16 hours and 35 minutes. When users use the web application spending the same time, they would collect approximately 96.9 times, namely 110,466 pieces of the character images for training data. Therefore, considering we need to collect a wide variety of training data from more books, the web application gives the user much less burden to collect training data.

Next, we examine the matching rates of the recognition results between by hand and by the web application. The matching rates are represented as a percentage that indicates recognition results obtained by both methods. This is different from simple recognition rates. Table 2 shows the results.

Comparison is performed using the items classified as follows.

(a) all of the characters
(b) characters without character clipping missing
(c) only Kanji characters out of (b)
(d) only Hiragana and Katakana characters out of (b)
(e) characters with character clipping missing
(f) separated characters with character clipping missing

Note that (e) and (f) are excluded from the recognition target in our previous studies [11]. It is because we used the target data collected by hand in our previous studies. In the case of (a) matching rate is 83.2%. The recognition with characters collected by hand does not include any characters with clipping missing. On the other hand, in the case of the web application, characters including missing to clip are the target of recognition, so the recognition rate is slightly lower. In the case of (b), only the characters without clipping missing are used similarly to previous studies [11]. However, the characters this time includes Hiragana and Katakana and it is different from the previous studies (just Kanji). The matching rate of (b) is 92.8% where the rate rises 9.6% compared with (a). The reason of the improvement is that the characters with clipping missing are removed from the target. In the case of (c), the target includes just Kanji characters without clipping missing. The matching rate of (c) is approximately 97.5% and increases 4.7% compared to (b). This is because Hiragana and Katakana characters are not included. Hiragana and Katakana are not investigated in our previous studies [11]. So we investigate the case of (d) where approximately 90.4% of Hiragana and Katakana characters without clipping missing are correctly recognized. The matching rate of (d) decreases by 7.1% compared to (c). We believe that the deterioration of Hiragana and Katakana recognition compared to Kanji is their simple structures. In general, the simpler a character is, the less features the PDC method extracts. So we need to increase the number of training data for Hiragana and Katakana rather than Kanji. In order to confirm the recognition rate of characters with clipping missing, we perform the experiment (e). The target characters in (e) are not correctly clipped as shown in Fig.10. The matching rate of (e) is approximately 35.6%. In this paper, we present the method how to increase the number of training data for early-modern printed Japanese character recognition. So we just abandon the characters with clipping missing from the training data target. The improvement of character clipping is outside of the scope of this paper. Figure 11 shows examples of character images incorrectly decomposed in the case of (f). Characters such as incorrectly connected, having too wide gaps and improperly penetrating between character components are likely clipped with incorrect decomposition. It indicates that 1.5% of the total characters are incorrectly decomposed from Tab.2. The problem of poorly decomposed characters are is currently under study.

We summarize the above descriptions as follows. From the results of (a) to (d), (c) has the highest matching rate. Also, from the results of (b) to (d), it is observed that the matching rate decreases slightly by including Hiragana and Katakana. Thus Kanji shows high recognition rates and likely to match the recognition results. By repeatedly using the web application, Hiragana and Katakana as well as Kanji are

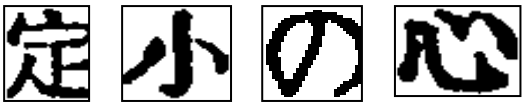


Figure 10:missing character images

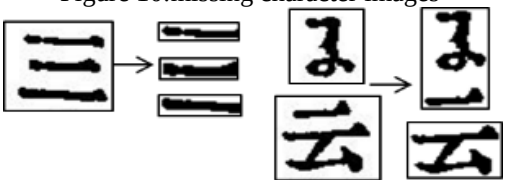


Figure 11: character images decompose

expected to be added at any time to the database of training data to get better recognition rates. Therefore, considering working time and the matching rate between manual and the web application, we conclude that the web application is sufficiently useful.

5.2 Comparison of the results by hand with the web Application

We show that we can collect how efficiently training data by the web application. Also we consider that how recognition rate changes by repeating the use of the web application. The initial database of the training data includes 1,000 types of characters and 4,345 pieces of the total characters. The number of average training data is 4.3. Using the web application, the number of training data increases.

Figure 12 shows the transition of the recognition rate and the number of character types. The number of character types included in the database becomes approximately 1.3 times

compared to the beginning. The recognition rate rises about three times. Then we compare the working time using the web application with by hand. The total working time is

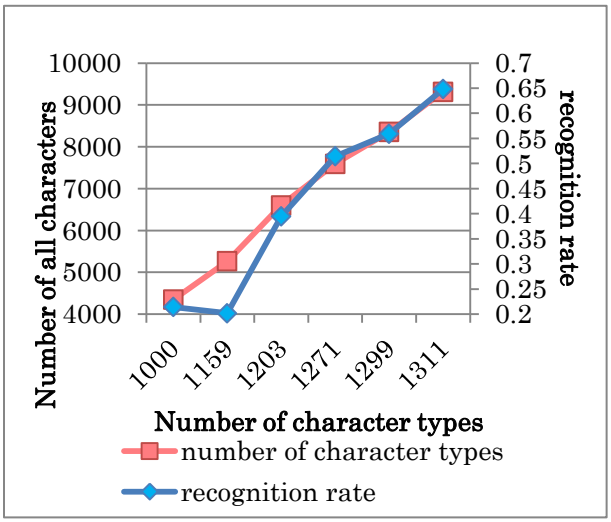


Figure 12 : transition of the recognition rate[%] and the number of character types[items] included in the database of training data.

approximately 10.5 hours to collect the number of training data using the web application while it is approximately 42.9 hours when collected similarly by hand. Thus, the working time for training data becomes approximately $\frac{1}{4}$ by using the web application.

6 Conclusions

In this paper, we described the development of a web application to efficiently collect necessary training data for early-modern Japanese printed character recognition. We compared recognition results and working time in the case of by hand and by the web application. The total processing time is approximately 10.3 minutes in the case of using the web application. In the other hand, it is approximately 16 hours 35 minutes by hand. In addition, the matching rate between only Kanji recognition results by the web application and by hand is about 97.5%. Then we conformed the transition of recognition rate and the types and the numbers of characters. The recognition rate becomes approximately 3 times compared to the beginning. The number of character types becomes approximately 1.3 times, and the number of the total raising data becomes approximately 2.1 times. The total working time by using the web application is approximately 10.5 hours while it takes approximately 42.9 hours by hand as well. Therefore the total working time of the web application is approximately $\frac{1}{4}$ times reduced. Therefore, our web application is sufficiently useful for the processing time, the matching rate, the recognition rate and the transition of types and the numbers of characters included in the database.

Acknowledgment

This work is partially supported by Grant-in-Aid for scientific research from the Ministry of Education, Culture, Sports, Science and Technology of Japan (MEXT) No. 26280119. We would like to give heartful thanks to Saki Okamura and Natsuko Takagita and Rena Kakihara and Akemi Kanematsu and Satsuki Tomizawa of Nara Women's University for their writing programs.

References

[1] National Diet Library :http://www.ndl.go.jp/
[2] Degital Library From the Meiji Era, kindai.ndl.go.jp/
[3] Ishikawa,C., Ashida,N., Enomoto,Y., Takata,M., Kimesawa,T., and Joe,K. : Recognition of Multi-Fonts Character in Early-Modern Printed Books , Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA' 09), Vol. II , pp. 728-734(2009).
[4] Fukuo,M., Enomoto,Y., Yoshii,N., Takata,M., Kimesawa,T. and Joe,K. : Evalua-Tion of the SVM based Multi-Fonts Kanji Character Recognition Method for Early- Modern Japanese Printed Books, Proceedings of The 2011 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA2011), Vol. II , pp. 727-732(2011).
[5] M. Fukuo, M. T. and Joe, K. (2012). The Kanji character recognition evalution for the modern book of the same publisher (in Japanese). The Information Processing Society of Japan. Mathematical Modeling and Problem Solving(MPS), 26:1–6.
[6] Aozora Bunko : http://www.aozora.gr.jp/
[7] Japanese Industrial Standards : https ://www.jisc.go.jp/
[8] Taeka Awazu, Manami Fukuo, Masami Takata and Kazuki Joe : A Multi-Fonts Kanji Character Recognition Method for Early-Modern Japanese Printed Books with Ruby Characters, International Conference on Pattern Recognition Applications and Methods (ICPRAM 2014), 637-645 (2014.3)
[9] N. Hagita, S. Naito and I. Masuda. "Handprinted Chinese Characters Recognition by Peripheral Direction Contributivity Feature", IEICE, Vol.J66-D, 10, pp.1185-1192, 1983. (in Japanese)
[10] Cristianini, N. andShawe-Taylor, J. : Support vector machine Introduction, Kyoritsu Publisher(2005).
[11] Taeka Awazu, Manami Fukuo, Masami Takata and Kazuki Joe : Analysis of the Relation between the Number of Necessary Training Data and Each Category for Multi-Fonts Kanji Character Recognition Method (in Japanese). *The Information Processing Society of Japan, Mathematical Modeling and Problem Solving(MPS)*, 97 :1-6.
[12] National Institute of Informatics : http://www.nii.ac.jp/
[13] V. Vapnik. : "The Nature of Statistical Learning Theory". Springer-Verlag, (1995)

Residual Inter-Contact Time for Opportunistic Networks with Pareto Inter-Contact Time: Two Nodes Case

Juntao Gao[1] **and Minoru Ito**[1]
[1]Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma City, Nara, Japan

Abstract— *Opportunistic networks (OppNets) are appealing for many applications, such as wild life monitoring, disaster relief and mobile data offloading. In such a network, a message arriving at a mobile node could be transmitted to another mobile node when they opportunistically move into each other's transmission range (called in contact), and after multi-hop similar transmissions the message will finally reach its destination. Therefore, for one message the time interval from its arrival at a mobile node to the time the mobile node contacts another node constitutes an essential part of the message's whole delay. Thus, studying stochastic properties of this time interval between two nodes lays a solid foundation for evaluating the whole message delay in OppNets. Note that this time interval is within the time interval between two consecutive node contacts (called inter-contact time) and it is also referred to as residual inter-contact time. In this paper, we derive the closed-form distribution for residual inter-contact time. First, we formulate the contact process of a pair of mobile nodes as a renewal process, where the inter-contact time features the popular Pareto distribution. Then, we derive, based on the renewal theory, closed-form results for the transient distribution of residual inter-contact time and also its limiting distribution. Our theoretical results on distribution of residual inter-contact time are validated by simulations.*

Keywords: Opportunistic networks, DTNs, inter-contact times

1. Introduction

Nowadays, portable mobile nodes (e.g., smart phones, tablets, digital cameras, censors) have been used ubiquitously in our daily life. Equipped with advanced wireless communication technologies (e.g., Bluetooth, WiFi Direct and ZigBee), these mobile nodes are now able to communicate directly with each other when they opportunistically move into transmission range (also called in contact). This promises a novel communication paradigm, opportunistic networks (OppNets)[1], which exploit opportunistic direct contacts of mobile nodes to deliver messages among them [1], shown in Fig.1a. Since OppNets are cost-effective, resilient to node failures and can be deployed rapidly, they can be used to enable communications in extreme environments (e.g., disaster, rural areas and wildlife monitoring) and enhance communications in existing networks (e.g., offloading data traffic in cellular networks, where mobile nodes share data directly when in contact).

[1]OppNets are also referred to as delay tolerant networks (DTNs).

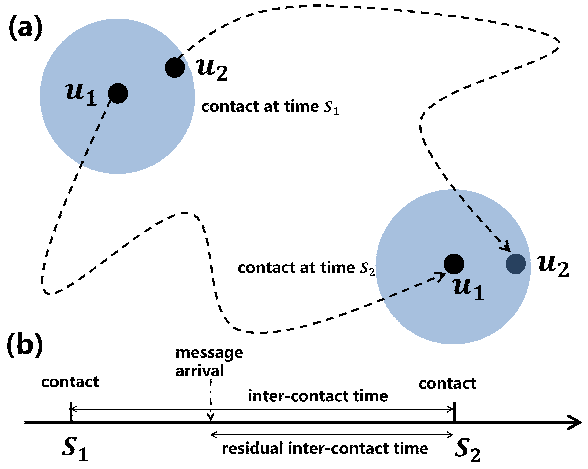


Fig. 1: (a) Direct communication when node u_1 contacts u_2. (b) Relationship between residual inter-contact time and inter-contact time.

In OppNets, a message arriving at a mobile node is transmitted directly to another mobile node when the two nodes opportunistically contact each other, and after multi-hop similar transmissions the message will finally reach its destination. Therefore, for one message the time interval from its arrival at a mobile node to the time the mobile node contacts another node constitutes an essential part of the whole delay of that message and thus significantly impacts the message's delay performance. Note that this time interval is within the time interval between two consecutive node contacts (called inter-contact time) as shown in Fig.1b, and it is also referred to as residual inter-contact time. Since residual inter-contact time represents the time a message has to wait at a mobile node before getting transmitted to next mobile node, its study serves as a cornerstone for evaluating the whole message delay.

As residual inter-contact time is embedded in inter-contact time, extensive works have been done first on investigating inter-contact time distribution. By analyzing real mobility traces of OppNets [2]–[4], the authors [5], [6] found that Pareto distribution with or without exponential cutoff is a good fitting distribution for inter-contact time there. By ana-

lyzing synthetic mobility models (such as random waypoint, random walk), the authors [6], [7] also concluded that Pareto distribution is a reasonable distribution for characterizing inter-contact time (refer to Section 2 for more related works on inter-contact time). Based on such findings, researchers studied the distribution of residual inter-contact time for OppNets with Pareto inter-contact time. The authors [5] derived upper and lower bounds for the distribution of residual inter-contact time. Later, the authors [6] presented a general expression for calculating the accurate distribution of residual inter-contact time, but arrived at a wrong distribution result for OppNets with Pareto inter-contact time [8]. The authors [8] attempted to derive the correct distribution of residual inter-contact time, however, they used the limiting time-average fraction of residual time less than given value to represent the real distribution of residual inter-contact time, which is not justified.

Our contribution in this paper is to rigorously derive the accurate distribution of residual inter-contact time for homogeneous OppNets, where inter-contact times of all node pairs follow the same Pareto distribution. Our work also justifies the result in [8].

- First, we formulate the contact process of a pair of mobile nodes as a renewal process, where the inter-contact time features the popular Pareto distribution [5], [9].
- Then, we derive, based on the renewal theory, closed-form results for transient distribution of residual inter-contact time and also its limiting distribution as time goes to infinity. For a tagged node, we also derive the distribution of the shortest residual inter-contact time between that node and all other nodes.
- Finally, we conduct simulations to validate the theoretical results on distribution of residual inter-contact time.

Our results on distribution of residual inter-contact time can be used to analyze delay performance of popular routing protocols in OppNets, such as epidemic routing and two-hop relay routing protocols.

The rest of this paper is organized as follows. In Section 2 we give more related works on inter-contact times. In Section 3, we present problem formulation by rigorously defining inter-contact time, residual inter-contact time and relative quantities. We then derive both transient and limiting distributions for residual inter-contact time in Section 4. We present simulation results to validate our derived distribution in Section 5. Finally, we conclude this paper in Section 6.

2. Related Works

Cai and Eun [10] investigated the age of inter-contact time between two mobile nodes, which could be used to study their residual inter-contact time properties. Passarella and Conti [11] characterized the relationship between the distributions of inter-contact times of different node pairs and the resulting aggregate distribution (the distribution obtained from aggregating samples of inter-contact times of all node pairs) in heterogeneous opportunistic networks where inter-contact times of different node pairs follow different distributions. Through empirical statistical analysis, Zhu and others [12] reported that inter-contact times of vehicles follow exponential distribution. By modeling general synthetic mobility model, La [13] showed that the distribution of inter-contact times can be well approximated by an exponential distribution under some conditions. Most works in the literature on inter-contact times used aggregated samples of inter-contact times of all nodes to estimate inter-contact time distribution of a pair of nodes, however, Orallo and others [14] showed this method cannot accurately characterize pair-wise inter-contact time distribution. Instead, they proposed another two methods (namely, aggregate nodes and any contact) to better characterize inter-contact time distribution. Biondi and others [15] studied the effect of power saving policy (duty cycling) on the performances of inter-contact times between mobile devices and showed that the inter-contact times under duty cycling are approximately exponential when original inter-contact times of devices are exponential.

3. Problem Formulation

Suppose two mobile nodes u_1 and u_2 move around in an OppNet, they employ the same wireless transmission range as shown in Fig.1a. We define the following terms for the two nodes.

Contact: We say node u_1 contacts node u_2 if u_2 moves into the wireless transmission rang of u_1, as illustrated in Fig.1a.

Contact Epoch: A contact epoch is the time instant at which u_1 contacts u_2. We denote successive contact epochs of u_1 and u_2 by $S_0, S_1, S_2, \cdots$ where $0 = S_0 < S_1 < S_2 < \cdots$.

Inter-Contact Time: An inter-contact time X for u_1 and u_2 is the time interval between their two consecutive contacts, as shown in Fig.1b. Thus, the i-th inter-contact time $X_i = S_i - S_{i-1}$, where $i \geq 1$. We assume X_i's are independent and identically distributed (IID) random variables as previous works [5]–[7].

Contact Process: The contact process of u_1 and u_2 is denoted by $\{N(t); t > 0\}$ where $N(t)$ records the number of total contacts between u_1 and u_2 occurring in time interval $(0, t]$, i.e., up to and including time t. Then, $N(t) = n$ if and only if $S_n \leq t < S_{n+1}$, as shown in Fig.2. Note also that if u_1 and u_2 contact at time t then $S_{N(t)} = t$. Since all inter-contact times are IID, the contact process is actually a renewal process (a renewal process is an arrival process where all inter-arrival intervals are positive IID random variables) [16].

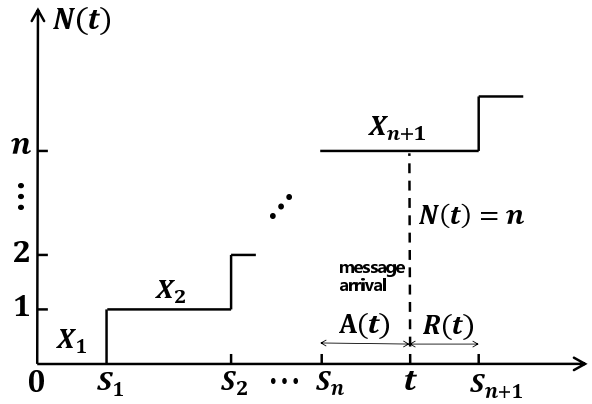


Fig. 2: Sample path of contact process $N(t)$ for u_1 and u_2.

Residual Inter-Contact Time: Assume a message arrives at u_1 at time t, then the time interval from t to the next time node u_1 contacts node u_2 is defined to be the residual inter-contact time for that message at time t, which is denoted by $R(t)$ and $R(t) = S_{N(t)+1} - t > 0$, i.e., the time interval within the inter-contact time as shown in Fig.2.

Age of Inter-Contact Time: Assume a message arrives at u_1 at time t, then the time interval from the most recent contact between u_1 and u_2 before/at t to time t is defined to be the the age of inter-contact time for that message at time t, which is denoted by $A(t)$ and $A(t) = t - S_{N(t)} \geq 0$, as shown in Fig.2.

Note that if $N(t) = 0$ at time t, it means that no contact has happened between u_1 and u_2 in time interval $(0, t]$, then the first inter-contact time X_1 must satisfy $X_1 > t$ and consequently $A(t) = t - S_0 = t$; if $N(t) \geq 1$ at time t, then $S_{N(t)} > 0$ and $A(t) = t - S_{N(t)} < t$. To sum up, $0 \leq A(t) \leq t$.

Concerned Inter-Contact Time: Assume a message arrives at u_1 at time t, then the time interval from the most recent contact epoch of u_1 and u_2 before/at t, $S_{N(t)}$, to the next time they contact each other, $S_{N(t)+1}$, is defined to be the concerned inter-contact time, which is denoted by $\widetilde{X}(t)$ and $\widetilde{X}(t) = S_{N(t)+1} - S_{N(t)} = X_{N(t)+1} > 0$.

From the definitions of $R(t)$, $A(t)$ and $\widetilde{X}(t)$, we have the followings: $\widetilde{X}(t) = R(t) + A(t)$ and $\widetilde{X}(t) > A(t)$, for any $t \geq 0$.

Pareto Inter-Contact Time: Analysis of real mobility traces and synthetic mobility models suggests that Pareto distribution can well approximate the distribution of inter-contact times in OppNets [5]–[7]. Thus, we assume all inter-contact times X_i's for node u_1 and u_2 feature a Pareto distribution with scalar parameters $x_m > 0$ and $\alpha > 1$ as follow [6], [8],

$$F_X(x) = Pr\{X \leq x\} = \begin{cases} 1 - \left(\frac{x_m}{x}\right)^{\alpha} & \text{if } x \geq x_m, \\ 0 & \text{if } 0 < x < x_m, \end{cases} \tag{1}$$

from which we also have the mean value

$$\overline{X} = \mathbb{E}\{X\} = \frac{\alpha x_m}{\alpha - 1}. \tag{2}$$

4. Inter-Contact Time Analysis

In this section we derive closed-form results for the distribution of residual inter-contact time. We first present a lemma below, which will be used in our following derivation.

Lemma 1: Consider the contact process $\{N(t); t > 0\}$ between u_1 and u_2 defined before. Suppose a message arrives at u_1 at time t. For given constant a, δ and x satisfying $0 \leq a < a + \delta \leq t$, $a + 2\delta \leq x$, let E denote the following event

$$E = \{a \leq A(t) < a + \delta, x - \delta < \widetilde{X}(t) \leq x\}, \tag{3}$$

where $A(t)$ is the age of inter-contact time at time t for that message, $\widetilde{X}(t)$ is the concerned inter-contact time.

Then, we have

$$Pr\{E\} = \big(m(t-a) - m(t-a-\delta)\big)\big(F_X(x) - F_X(x-\delta)\big) \tag{4}$$

where $m(t) = \mathbb{E}\{N(t)\}$.

Proof: Since the contact process forms a renewal process, this lemma follows directly from Theorem 5.7.2 [16]. ■

Now, we derive the transient distribution of residual inter-contact time for nodes u_1 and u_2.

Theorem 1: For an OppNet with Pareto inter-contact times given in (1), the transient distribution of residual inter-contact time $R(t)$ for a message at time t is

$$\begin{aligned} Pr\{R(t) \leq r\} = &\left(\frac{x_m}{t}\right)^{\alpha} - \left(\frac{x_m}{t+r}\right)^{\alpha} \\ &+ \int_0^t \big(F_X(t-\tau+r) - F_X(t-\tau)\big)\mathrm{d}m(\tau) \end{aligned} \tag{5}$$

where $r > 0$.

Proof: Assume a message arrives at node u_1 at time t. Then,

$$Pr\{R(t) \leq r\} \tag{6}$$

$$= Pr\big\{\widetilde{X}(t) - A(t) \leq r\big\} \tag{7}$$

$$= \underbrace{Pr\big\{\widetilde{X}(t) - A(t) \leq r, A(t) = t\big\}}_{P_1} + \underbrace{Pr\big\{\widetilde{X}(t) - A(t) \leq r, 0 \leq A(t) < t\big\}}_{P_2} \tag{8}$$

where (8) follows from the fact that $0 \leq A(t) \leq t$ and the law of total probability.

We calculate probability P_1 first. Recall that $\widetilde{X}(t)$ is the concerned inter-contact time containing time t and $\widetilde{X}(t) >$

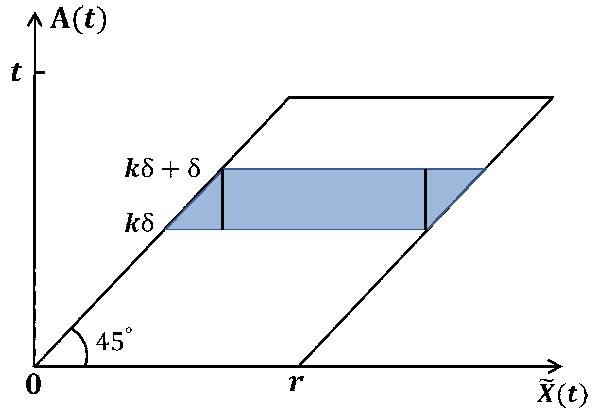


Fig. 3: Sample points of joint $A(t)$ and $\widetilde{X}(t)$.

$A(t)$ for any t.

$$P_1 = Pr\{A(t) < \widetilde{X}(t) \le A(t) + r, A(t) = t\} \quad (9)$$
$$= Pr\{t < \widetilde{X}(t) \le t + r, A(t) = t\} \quad (10)$$
$$= Pr\{t < X_{N(t)+1} \le t + r, A(t) = t\} \quad (11)$$
$$= Pr\{t < X \le t + r\} \quad (12)$$
$$= F_X(t + r) - F_X(t) \quad (13)$$
$$= \Big(\frac{x_m}{t}\Big)^{\alpha} - \Big(\frac{x_m}{t+r}\Big)^{\alpha} \quad (14)$$

where (12) follows from the fact that $A(t) = t$ indicates no contact happens in $(0, t]$, i.e., $N(t) = 0$, thus, $\widetilde{X}(t) = X_{N(t)+1} = X_1$. Note also that all X_i's follow the same Pareto distribution given in (1).

We next calculate probability P_2.

$$P_2 = Pr\{A(t) < \widetilde{X}(t) \le A(t) + r, 0 \le A(t) < t\} \quad (15)$$
$$= \sum_{k=0}^{l-1} \underbrace{Pr\{A(t) < \widetilde{X}(t) \le A(t) + r, k\delta \le A(t) < k\delta + \delta\}}_{\widetilde{P}_k} \quad (16)$$

where we divide interval $0 \le A(t) < t$ into l sub-intervals $[k\delta, k\delta + \delta)$, $0 \le k \le l - 1$, and $\delta = \frac{t}{l}$.

Next, we calculate the general term $\widetilde{P}_k$ in (16),

$$\widetilde{P}_k = Pr\{A(t) < \widetilde{X}(t) \le A(t) + r, k\delta \le A(t) < k\delta + \delta\} \quad (17)$$

Note that $\widetilde{P}_k$ is the probability of the event illustrated by the gray area of sample points of age $A(t)$ and $\widetilde{X}(t)$, shown in Fig.3. From this figure, we see that

$$\widetilde{P}_k \ge Pr\{k\delta + \delta < \widetilde{X}(t) \le k\delta + r, k\delta \le A(t) < k\delta + \delta\}, \quad (18)$$
$$= \big(m(t - k\delta) - m(t - k\delta - \delta)\big) \cdot \big(F_X(k\delta + r) - F_X(k\delta + \delta)\big), \quad (19)$$

where (19) follows from Lemma 1. Similarly,

$$\widetilde{P}_k \le Pr\{k\delta < \widetilde{X}(t) \le k\delta + r + \delta, k\delta \le A(t) < k\delta + \delta\}, \quad (20)$$
$$= \big(m(t - k\delta) - m(t - k\delta - \delta)\big) \cdot \big(F_X(k\delta + r + \delta) - F_X(k\delta)\big). \quad (21)$$

Thus, from (15), (19) and (21), we have

$$P_2 \ge \underbrace{\sum_{k=0}^{l-1} \big(m(t - k\delta) - m(t - k\delta - \delta)\big) \cdot F_X(k\delta + r)}_{L_1} - \underbrace{\sum_{k=0}^{l-1} \big(m(t - k\delta) - m(t - k\delta - \delta)\big) \cdot F_X(k\delta + \delta)}_{U_2}, \quad (22)$$

and

$$P_2 \le \underbrace{\sum_{k=0}^{l-1} \big(m(t - k\delta) - m(t - k\delta - \delta)\big) \cdot F_X(k\delta + r + \delta)}_{U_1} \quad (23)$$
$$- \underbrace{\sum_{k=0}^{l-1} \big(m(t - k\delta) - m(t - k\delta - \delta)\big) \cdot F_X(k\delta)}_{L_2}, \quad (24)$$

where U_1 and L_1 are just the upper and lower Stieltjes sums of $F_X(t - \tau + r)$ with respect to $m(\tau)$ on $[0, t]$, U_2 and L_2 are the upper and lower Stieltjes sums of $F_X(t - \tau)$ with respect to $m(\tau)$ on $[0, t]$.

Since $m(\tau)$ is the expectation of $N(t)$, $m(\tau)$ is an increasing function on $[0, t]$ and thus is of bounded variation on $[0, t]$ (Theorem 6.5 [17]). Note also that $F_X(t - \tau + r)$ is a continuous function on $[0, t]$ since $F_X(x)$ is Pareto distribution. These two conditions indicate the existence of the following Riemann-Stieltjes integral (Theorem 7.27 [17])

$$\int_0^t F_X(t - \tau + r)\mathrm{d}m(\tau). \quad (25)$$

The existence of Riemann-Stieltjes integral in (25) further indicates that (Theorem 7.19 [17]) for any $\epsilon_1 > 0$,

$$0 \le U_1 - L_1 < \epsilon_1, \quad \text{as} \quad l \to \infty. \quad (26)$$

Similarly, the following Riemann-Stieltjes integral also exists

$$\int_0^t F_X(t - \tau)\mathrm{d}m(\tau) \quad (27)$$

and for any $\epsilon_2 > 0$,

$$0 \le U_2 - L_2 < \epsilon_2, \quad \text{as} \quad l \to \infty. \quad (28)$$

From (26) and (28), we know that for any $\epsilon > 0$,

$$0 \le (U_1 - L_2) - (L_1 - U_2) < \epsilon, \quad \text{as} \quad l \to \infty. \tag{29}$$

Thus, according to Theorem 7.19 [17], we have

$$P_2 = \int_0^t \big(F_X(t-\tau+r) - F_X(t-\tau)\big)\mathrm{d}m(\tau) \tag{30}$$

■

Next, we derive the limiting distribution of residual inter-contact time $R(t)$ as $t \to \infty$.

Theorem 2: For an opportunistic network with Pareto inter-contact time given in (1), the limiting distribution of residual inter-contact time $R(t)$ of a message as $t \to \infty$ is

$$\lim_{t\to\infty} Pr\{R(t) \le r\} = \begin{cases} 1 - \frac{1}{\alpha}\left(\frac{x_m}{r}\right)^{\alpha-1} & \text{if } r \ge x_m, \\ \frac{r\alpha - r}{\alpha x_m} & \text{if } 0 < r < x_m. \end{cases} \tag{31}$$

Proof: From (5), we know

$$\lim_{t\to\infty} Pr\{R(t) \le r\}$$
$$= \lim_{t\to\infty} \int_0^t \big(F_X(t-\tau+r) - F_X(t-\tau)\big)\mathrm{d}m(\tau). \tag{32}$$

From the key renewal theorem [16], we know

$$\lim_{t\to\infty} \int_0^t \big(F_X(t-\tau+r) - F_X(t-\tau)\big)\mathrm{d}m(\tau)$$
$$= \frac{1}{\overline{X}} \int_0^\infty \big(F_X(x+r) - F_X(x)\big)\mathrm{d}x \tag{33}$$

where $F_X(x)$ and $\overline{X}$ are given in (1) and (2), respectively.

We next calculate the integral in (33). For $0 < r \le x_m$,

$$F_X(x+r) = \begin{cases} 1 - \left(\frac{x_m}{x+r}\right)^{\alpha} & \text{if } x + r \ge x_m, \\ 0 & \text{if } 0 < x + r < x_m. \end{cases} \tag{34}$$

Thus,

$$F_X(x+r) - F_X(x)$$
$$= \begin{cases} \left(\frac{x_m}{x}\right)^{\alpha} - \left(\frac{x_m}{x+r}\right)^{\alpha} & \text{if } x > x_m, \\ 1 - \left(\frac{x_m}{x+r}\right)^{\alpha} & \text{if } x_m - r < x \le x_m, \\ 0 & \text{if } 0 < x \le x_m - r. \end{cases} \tag{35}$$

Then, we have

$$\int_0^\infty \big(F_X(x+r) - F_X(x)\big)\mathrm{d}x$$
$$= \int_{x_m-r}^{x_m} \Big(1 - \Big(\frac{x_m}{x+r}\Big)^{\alpha}\Big)\mathrm{d}x + \int_{x_m}^{\infty} \Big(\Big(\frac{x_m}{x}\Big)^{\alpha} - \Big(\frac{x_m}{x+r}\Big)^{\alpha}\Big)\mathrm{d}x \tag{36}$$
$$= \quad r \tag{37}$$

For $r > x_m$, since $x + r > x_m$ for any $x > 0$, we have

$$F_X(x+r) = 1 - \big(\frac{x_m}{x+r}\big)^{\alpha},\ x > 0. \tag{38}$$

Thus,

$$F_X(x+r) - F_X(x)$$
$$= \begin{cases} \left(\frac{x_m}{x}\right)^{\alpha} - \left(\frac{x_m}{x+r}\right)^{\alpha} & \text{if } x > x_m, \\ 1 - \left(\frac{x_m}{x+r}\right)^{\alpha} & \text{if } 0 < x \le x_m. \end{cases} \tag{39}$$

Then, we have

$$\int_0^\infty \big(F_X(x+r) - F_X(x)\big)\mathrm{d}x$$
$$= \int_0^{x_m} \Big(1 - \Big(\frac{x_m}{x+r}\Big)^{\alpha}\Big)\mathrm{d}x + \int_{x_m}^{\infty} \Big(\Big(\frac{x_m}{x}\Big)^{\alpha} - \Big(\frac{x_m}{x+r}\Big)^{\alpha}\Big)\mathrm{d}x \tag{40}$$
$$= x_m + \frac{x_m^{\alpha} r^{-\alpha+1}}{-\alpha+1} - \frac{x_m}{-\alpha+1} \tag{41}$$

To sum up,

$$\int_0^\infty \big(F_X(x+r) - F_X(x)\big)\mathrm{d}x \tag{42}$$
$$= \begin{cases} x_m + \frac{x_m^{\alpha} r^{-\alpha+1}}{-\alpha+1} - \frac{x_m}{-\alpha+1} & \text{if } r > x_m, \\ r & \text{if } 0 < r \le x_m. \end{cases} \tag{43}$$

After substituting (2) and (43) into (33), we have

$$\lim_{t\to\infty} Pr\{R(t) \le r\} = \begin{cases} 1 - \frac{1}{\alpha}\left(\frac{x_m}{r}\right)^{\alpha-1} & \text{if } r > x_m, \\ \frac{r\alpha - r}{\alpha x_m} & \text{if } 0 < r \le x_m. \end{cases} \tag{44}$$

This completes the proof. ■

Finally, we want to find out how long it will take a node with a new arrival message to contact another node in the OppNet, thus having opportunities to forward the message to the next node.

Suppose a homogeneous opportunistic network of W nodes, where all nodes move independently and the inter-contact times of every pair of nodes feature the same Pareto distribution given in (1). Let $R_{ij}(t)$ be the residual inter-contact time for node i and node j, then the time interval $R^*(t)$ from time t the message arrives at node u_1 to the time u_1 contacts any one of the other $W-1$ nodes in the network is

$$R^*(t) = \min\big\{R_{12}(t), R_{13}(t), \cdots, R_{1W}(t)\big\} \tag{45}$$

Lemma 2: The limiting distribution of $R^*(t)$ is given as follows.

$$\lim_{t\to\infty} Pr\{R^*(t) \le r\}$$
$$= \begin{cases} 1 - \left(\frac{1}{\alpha}\right)^{W-1}\left(\frac{x_m}{r}\right)^{(\alpha-1)(W-1)} & \text{if } r \ge x_m, \\ 1 - \left(\frac{\alpha x_m - r\alpha + r}{\alpha x_m}\right)^{W-1} & \text{if } 0 < r < x_m. \end{cases} \tag{46}$$

Proof:

$$Pr\{R^*(t) \le r\} = 1 - Pr\{R^*(t) > r\} \tag{47}$$
$$= 1 - Pr\{R_{12}(t) > r, R_{13}(t) > r, \cdots, R_{1W}(t) > r\} \tag{48}$$
$$= 1 - Pr\{R_{12}(t) > r\}Pr\{R_{13}(t) > r\}\cdots Pr\{R_{1W}(t) > r\} \tag{49}$$

where (49) follows from the independence of node mobility. After substituting (31) into (49), we got (46). ■

5. Simulation Results

To validate the derived distribution of residual inter-contact time, we developed a customized simulator in C++ to simulate the contact process between two nodes u_1, u_2, the random message arrival process to node u_1, and observe the residual inter-contact times regarding message arrivals. Specifically, we simulated three different network scenarios where inter-contact times between u_1 and u_2 all follow Pareto distribution but with different scalar parameter settings: $(x_m = 1.0, \alpha = 1.5)$, $(x_m = 1.0, \alpha = 2.0)$ and $(x_m = 2.0, \alpha = 3.0)$. We assume messages arrive at node u_1 according to a Poisson process with arrival rate of 0.001. During simulations, we measured the residual inter-contact time for a message as the time interval from the time it arrives at u_1 to the next time u_1 contacts u_2. From the measured residual inter-contact times, we calculated their distribution. The simulated distributions under three network scenarios are presented in Fig.4, where corresponding theoretical distributions are also given for comparison. From Fig.4, we can see that our derived distributions for residual inter-contact time perfectly match the simulated ones, verifying our theoretical results.

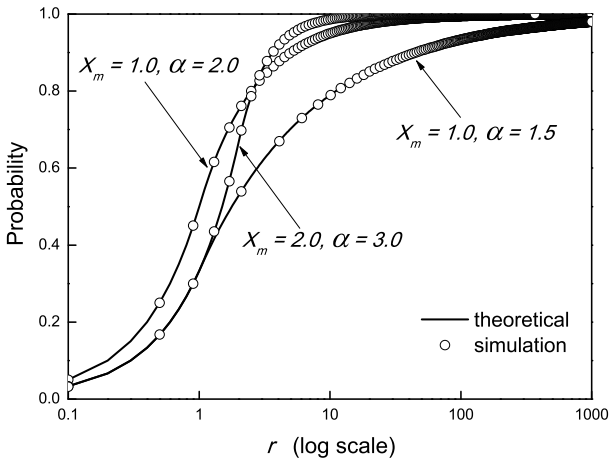


Fig. 4: Disbribution of residual inter-contact time under different Pareto distribution parameters.

6. Conclusion

In this paper, we rigorously derived the distribution of residual inter-contact time for opportunistic networks with Pareto inter-contact times. Our results have important implications for applications (by the law of large numbers): for a homogeneous OppNet, where contacts of all node pairs follow common inter-contact time distribution (e.g., students in a campus and corporate users [18]), the distribution of residual inter-contact times can be found out by collecting samples of residual inter-contact times from all node pairs in stead of collecting samples from the same node pair for a long time. This creates great experiment convenience, since long-time tracking of the same pair of nodes is usually prohibited due to privacy while short-time tracking all node pairs is much easier.

References

[1] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 134–141, November 2006.

[2] M. McNett and G. M. Voelker, "Access and mobility of wireless pda users," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 2, pp. 40–55, April 2005.

[3] J. Su, A. Chin, A. Popivanova, A. Goel, and E. de Lara, "User mobility for opportunistic ad-hoc networking," in *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 2004.

[4] N. Eagle and A. Pentland, "Reality mining: sensing complex social systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, pp. 255–268, May 2006.

[5] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Impact of human mobility on opportunistic forwarding algorithms," *IEEE Transactions on Mobile Computing*, vol. 6, no. 6, pp. 606–620, April 2007.

[6] T. Karagiannis, J.-Y. L. Boudec, and M. Vojnovic, "Power law and exponential decay of intercontact times between mobile devices," *IEEE Transactions on Mobile Computing*, vol. 9, no. 10, pp. 1377–1390, August 2010.

[7] H. Cai and D. Y. Eun, "Crossing over the bounded domain: From exponential to power-law intermeeting time in mobile ad hoc networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1578–1591, October 2009.

[8] C. Boldrini, M. Conti, and A. Passarella, "From pareto inter-contact times to residuals," *IEEE Communications Letters*, vol. 15, no. 11, pp. 1256–1258, November 2011.

[9] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[10] H. Cai and D. Y. Eun, "Aging rules: What does the past tell about the future in mobile ad-hoc networks?" in *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing (MobiHoc)*, 2009.

[11] A. Passarella and M. Conti, "Analysis of individual pair and aggregate inter-contact times in heterogeneous opportunistic networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 12, pp. 2483–2495, October 2013.

[12] H. Zhu, L. Fu, G. Xue, Y. Zhu, M. Li, and L. M. Ni, "Recognizing exponential inter-contact time in vanets," in *Proceedings IEEE INFOCOM*, 2010.

[13] R. J. La, "Distributional convergence of intermeeting times under the generalized hybrid random walk mobility model," *IEEE Transactions on Mobile Computing*, vol. 9, no. 9, pp. 1201–1211, Semptember 2010.

[14] E. Hernandez-Orallo, J.-C. Cano, C. T. Calafate, and P. Manzoni, "A representative and accurate characterization of inter-contact times in mobile opportunistic networks," in *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems (MSWiM)*, 2013.

[15] E. Biondi, C. Boldrini, M. Conti, and A. Passarella, "Duty cycling in opportunistic networks: the effect on intercontact times," in *Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems (MSWiM)*, 2014.

[16] R. G. Gallager, *Stochastic Processes: Theory for Applications*. Cambridge University Press, February 2014.
[17] T. M. Apostol, *Mathematical Analysis*. Addison-Wesley, 1974.
[18] J. Leguay, T. Friedman, and V. Conan, "Evaluating mobility pattern space routing for dtns," in *Proceedings of 25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

Web of Wine Words: Hierarchy Visualization of Wine Speak by Restricted Bootstrap

Brendan Flanagan[1], **Sachio Hirokawa**[2]
[1]Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan
[2]Research Institute for Information Technology, Kyushu University, Fukuoka, Japan

Abstract - *Visualization of the relation of characteristic words can be useful for interpreting search results and enable comparisons to be made between multiple searches. In this paper we introduce a method of analysis by applying restricted bootstrapping to a set of characteristic words for extracting specificity or generality relations. These relations are used to construct a tree structure of the characteristic words that represents their hierarchical specificity or generality. This method was applied to a corpus of wine tasting notes to identify the characteristics of two wine regions by hierarchy tree. The results are compared with the frequencies of the characteristic words.*

Keywords: Restricted Bootstrap, Characteristic Hierarchy Visualization, Topic Drift, Wine Speak

1 Introduction

When searching it can be difficult to interpret from the results whether a characteristic has specificity or generality to the query. This is particularly apparent when trying to compare characteristics of search results for queries that might share many common attributes, such as wines from different regions. The characteristics of wine are often described using specialist expressions, called wine speak. People not familiar with these types of descriptions can find them confusing, as they don't have an understanding of different levels of particular and general expressions. Some words might have high generality, such as "smell", where as others are more specific, such as "crushed tomato leaf" as used by Joe Czerwinski in his review of a Villa Maria 2009 Sauvignon Blanc[1].

A situation is also similar to that found when searching on the Internet due to the vast quantity of information that is available today. A user sometimes finds that the search query they have entered does not return the intended results in the case of keywords with high generality, or no results in the case of keywords that are too specific. This may occur because the user did not have an understanding of the different levels of particular and general keywords.

The relations of characteristics are not immediately obvious when comparing simple search results. Some characteristics could be more of general quality, whereas others might be specific to the search query. The characteristics of search results can be thought of as a hierarchy tree, with words that are similar attaching to the same parent node. The parent nodes are then connect to create a word tree in order of greatest generality to the query at the root and greatest similarity at the lowest leaf nodes.

In this paper, we will demonstrate how the restricted bootstrap algorithm can be used to extract the degree of common generality between a pair of words. We investigate a method of generating a hierarchy of words from the specificity and generality relation of characteristic words and a corpus of target documents.

2 Related work

2.1 Information Extraction by Bootstrap

In previous research, Mihalcea et al. [7] and Palshikar [8] analyzed networks of words as undirected graphs made from the results of query word searches. Palshikar [8] proposed that central vertices in the graph are representative keywords of the document. Mihalcea et al. [7] proposed the TextRank method that scores words in a similar way to how the HITS algorithm scores hub/authority sites in a graph of web pages. Pantel et al. [9] proposed a minimally supervised bootstrapping algorithm named Espresso to extract semantic relations. Komachi et al. [6] demonstrated that semantic drift in bootstrapping is similar to that of the HITS algorithm and proposed graph based methods based on von Neumann kernels and regularized Laplacian to reduce semantic drift. While the above researches were carried out independently of each other, Radev et al. [12] notes that they all are based on the analysis of word co-occurrence as a bi-partite graph of documents and words by traversing back and forth between different node types to find feature words and sentences. More recently, Tian et al. [14] proposed using bootstrapping to extract Chinese hyponyms from a Web corpus by extracting double anchored hyponyms using bootstrapping.

We have previously revealed in [2] that semantic drift can be reduced by restricting the query length at each iteration of the bootstrap process. This algorithm was used to visualize the generality relation of search results in [4]. Also in other research, [3] we have examined the relations of wine speak

[1] http://buyingguide.winemag.com/catalog/villa-maria-2009-taylors-

expressions found in wine magazine blogs and visualized these as mind maps to support the learning of wine speak. In the present paper, we propose a method of extracting the specificity or generality relation of characteristic words to a search query by applying restricted bootstrapping.

2.2 Wine Tasting Note Analysis

There are many papers on research into the language that is used to describe wines, called Wine Speak. Some of this research is dedicated to analyzing wine tasting notes from different points of view. Caballero [1] focused on how manner-of-motion verbs are used from the point of view of describing a wine's intensity and persistence. Manner-of-motion verbs occur often used in wine tasting notes to depict motions, such as "hints of milk chocolate and vanilla *sneak in* on the palate". A corpus of wine tasting collected from the Wine Enthusiast, Wine Spectator, and Wine Advocate was analyzed and examples of 56 typical sentences that contain such verbs were given. Paradis and Eeg-Olofsson [11] examined tasting notes to identify expressions and words that are related to the viewpoints of vision, smell, taste, and touch. 39 typical phrases of these sensory expressions were identified. Paradis [10] investigated the analysis of semantic middles in wine tasting notes and their use as a recommender to estimate prime drinking time. A sub corpus of 200 notes was randomly selected from a corpus of 80,000 notes from the Wine Advocate and a meticulous evaluation of 38 sentences was given.

There is also related research into the visualization of wine tasting notes for linguistic analysis. Kerren et al. [5] visualized wine tasting notes using word trees generated from parts of speech and words. Their system enables the analysis of linguistic patterns within single wine reviews or based on regions and varieties. The system is highly specialized with the intention to be used for linguistic exploration of wine tasting notes. A database of 84,864 documents was analyzed and various visualizations are given. The word trees that are generated are limited to single documents and do not allow a visual overview of a subset of the corpus. In previous research, we examined the relations of wine speak expressions found in wine magazine blogs and visualized these as mind maps to support the learning of wine speak [4].

In the present paper, we propose a method of using the restricted bootstrapping algorithm to search for common generality between pairs of query words. This is then analyzed to generate a word hierarchy of the query words with relation to the target documents in the corpus.

3 Hierarchy generation by restricted bootstrapping

The generation of characteristic word hierarchies by restricted bootstrapping involves two main steps: firstly, applying restricted bootstrapping for each characteristic word paired with the target query, and secondly, generating the characteristic word hierarchy based on the analysis of the restricted bootstrapping results.

3.1 Restricted Bootstrap Algorithm

The second author of this paper initially proposed a method of restricted bootstrapping [2] as a solution to the problem of semantic drift that sometimes occurs when the result of a bootstrap that has been applied to documents and words is too far from the initial query. When evaluated on the extraction of infrequent characteristic words, we confirmed that a tight bootstrap restriction of $k = 1$ produces a 10% increase in the Mean Average Precision when compared to a looser restriction of $k = 50$. However a comprehensive quantitative evaluation is still required as future work.

```
BS(U,q,k) {
 W = {}
 i=0
 while(true){
    Wi = word(doc(U && q))
    W = top(k,Wi)
    last if W == U
    i = i+1
    U = W
 }
 return W
}
```

Figure 1. Restricted Bootstrap Algorithm

In this paper, the restricted bootstrap algorithm *BS(U,q,k)*, shown in Fig. 1, extracts words for each word in the characteristic word set *U* with relation to the initial query *q*. The algorithm starts by searching for a query comprising of the initial query *q* and a word from the characteristic word set U. The words of the documents in the search results are then ranked. As the algorithm was realized using a search engine constructed using GETA[2], the default SMART weight [13] was used as the word score for ranking each word. The bootstrap length restriction *k* is used to limit the number of top ranking search result words that are then used as the query for the next iteration of the bootstrap process. The iteration continues until the bootstrap has converged, which is when the current *k* top ranked words are the same as the *k* top ranked words from a previous iteration. The top *k* ranked words of the last iteration are returned as the results of the restricted bootstrap process.

3.2 Hierarchy Tree Generation

The bootstrap results of words that match at small *k* are closely related to the initial query. Conversely, a larger *k* increases the possibility of topic drift. We propose that these

[2] http://geta.ex.nii.ac.jp/

properties of restricted bootstrap can be used to analyze the relations of words within a corpus.

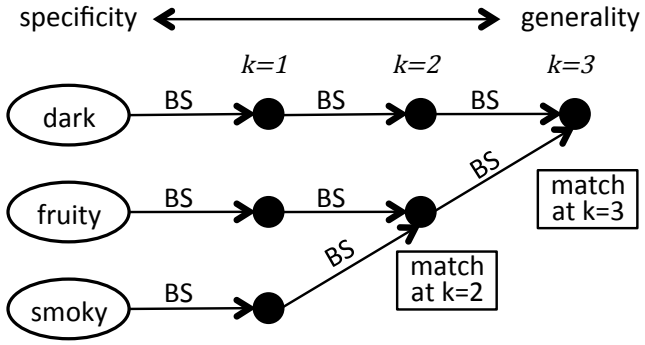


Figure 2. Hierarchy Tree Generation

An overview of the hierarchy tree generation process is shown in Fig. 2. The bootstrap results for all characteristic words are checked for exact matches at each step of increasingly larger k restriction lengths. Exact matches with the smallest restriction length k represent the relation between two or more characteristic words and are linked to the same parent node that represents the k search step. If the k is small, then the words have a strong relation to the initial search query. If the k is large, then the words have a weak relation to the initial search query. Small k nodes are linked with the next largest k node, until only one last k node is reached. We constructed a directed graph G = (N,E) of 17 characteristic words U of wine speak given a query q as follows where N is the set of nodes as defined in Equation 1, and E is the set of edges as defined in Equation 2.

$$N = \{BS(\{w_i\}, q, k) | i = 1, .., 17; k = 0,1, \dots; \} \quad (1)$$

$$\begin{aligned} \mathrm{E} = \{ & BS(\{u\}, q, m), BS(\{v\}, q, n) | \\ & \exists l \geq m \; s.t. \; BS(\{u\}, q, l) = BS(\{v\}, q, n) \\ & BS(\{v\}, q, n) = BS(\{u\}, q, l_0) \\ & l_0 = \min\{l | BS(\{v\}, q, n)\} \; \} \qquad \} \end{aligned} \quad (2)$$

This method can essentially be thought of as drawing paths of restricted bootstrapping results of increasingly larger k for each characteristic word in the set U. The paths are then merged at the point of an exact match for the lowest existing bootstrap restriction k.

4 Examples of hierarchy trees generated by applying restricted bootstrapping

4.1 Data Collection

A prototype system of the method was applied to a corpus consisting of 91,010 wine tasting notes from the Wine Enthusiast Magazine's Buying Guide[3] website. The attributes of each wine and the tasting notes were collected. We constructed a search engine to analyze the wine tasting notes, with each note containing an average of 2.8 sentences made up of 40 words. As an example of the method proposed in this paper, two types of target wine attributes that had been collected were selected for analysis: wine region and grape variety. Two regions were selected as the focus for the wine region category: New Zealand, and Marlborough. In the grape variety category we selected: Pinot Noir, Red Blend, and Zinfandel.

4.2 Characteristic Words: Sensory Expression

A list of 17 sensory modalities grouped in three categories from Paradis and Eeg-Olofsson [11] were chosen as the set of characteristic words U as shown in Table 1.

Table 1. 17 Example words that describe sensory modalities [11]

Modality	Example
VISION	purple, ruby, straw, gold, light, dark
SMELL	fruity, floral, spicy, smoky, weak
TASTE & TOUCH	flabby, soft, heavy, thin, long, crisp

The authors have previously used these words in the analysis of wine blogs [3].

4.3 Comparison of Wine Regions

A naïve analysis method would be to compare the frequency distributions of the characteristic word set, which is shown in Fig. 3 and Fig. 4. Words of a high frequency would have a stronger relation to the corpus than words of lower frequency. This is a simple ranking method when compared to the restricted bootstrap method proposed in this paper. This is because it doesn't take into account whether the characteristic keyword U are specific or general with relation to the query q.

A matching restricted bootstrapping search was applied with k from length 1 to 100 for each characteristic word to the tasting notes for wines from New Zealand and Marlborough (which is a famous region within New Zealand) containing 1427 and 715 documents respectively. The bootstrap results of all the possible pairs of characteristic words matched at least once before k = 100.

In Fig. 5 and 6 the hierarchy word trees produced with the proposed method for wine tasting notes from q = Marlborough and q = New Zealand are shown respectively, where the words in the nodes are colored red, green and blue

[3] http://buyingguide.winemag.com/

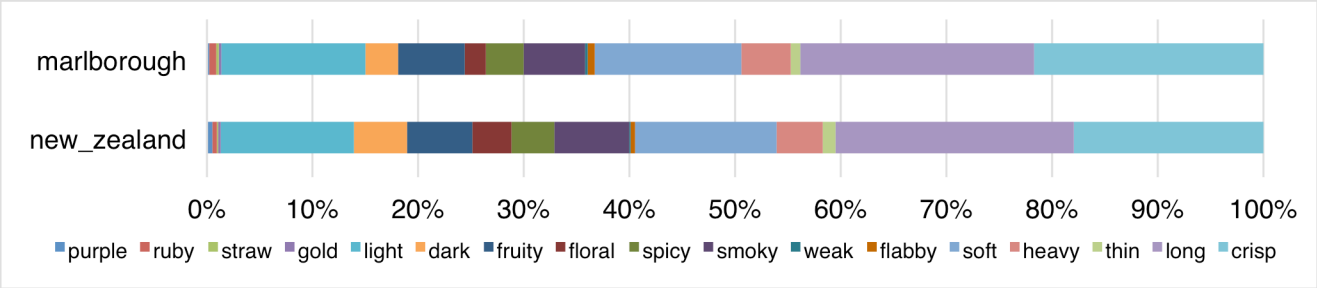


Figure 3. Comparison of region by word frequency

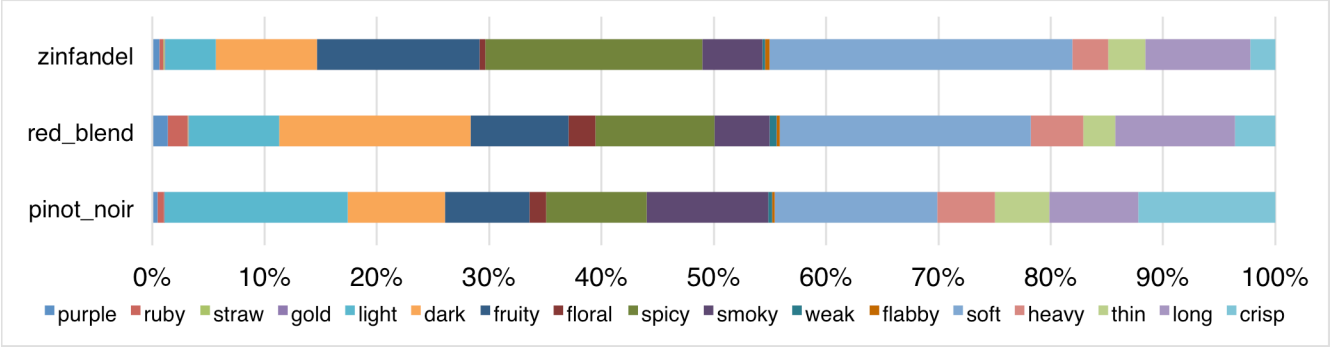


Figure 4. Comparison of grape variety by word frequency

for each category. The hierarchy tree is drawn from left to right as the bootstrap restriction length k increases, with words on the left side of the graph having a stronger relation to the corpus than words on the right side of the graph.

The five least frequent characteristic words U in the New Zealand corpus in ascending frequency order are: *straw*, *weak*, *gold*, *ruby*, and *flabby*. However these words are in the middle of the hierarchy tree suggesting that they have a stronger characteristic relation to New Zealand wines than would be expected by looking at the word distribution. The same can also be seen in the six least frequent characteristic words U for Marlborough in ascending order are: *purple*, *straw*, *gold*, *weak*, *dark*, and *flabby*. These words also occur in the middle of the hierarchy tree suggesting a stronger relation than could be deduced from the frequency distributions.

The distribution for the characteristic word *floral* has no difference between New Zealand and Marlborough by sample size rank (both rank 9), where $rank(w_i) = \#\{w_j \in W | w_i > w_j\}$ for the set of characteristic words W. However, in the hierarchy word tree, the difference between the depths of the *floral* node from the root of the trees is of 5 nodes. The *floral* node is closer to the root of the tree for the Marlborough corpus, which indicates that the minimum k matching restricted bootstrap results occurred at a larger k than found in the New Zealand corpus. This indicates that *floral* is a stronger characteristic of New Zealand wines than those from Marlborough, which is not apparent when comparing the distributions of words.

4.4 Comparison of Grape Varieties

Three grape varieties were selected as the focus of the example for this category: q = Pinot Noir, q = Red Blend, and q = Zinfandel, with each containing 8088, 5190, and 3002 documents respectively. The frequency distribution for each of these grape varieties is shown in Fig. 4. A matching restricted bootstrap search was applied in a similar method to the previous section, with a restriction length k of 1 to 100 for each characteristic word, with all possible pairs matching before k = 100.

The hierarchy trees generated for the three grape varieties, Pinot Noir, Red Blend, and Zinfandel, are shown in Fig. 7, 8, and 9 respectively. The depths of the grape variety trees are greater than those of the wine regions as there are a larger number of documents. The less frequency characteristic words U for each grape variety are matched near the middle of the hierarchy trees (in ascending frequency order, Pinot Noir: *straw, gold, flabby, weak, purple*; Red Blend: *gold, straw, flabby, weak, purple*; Zinfandel: *gold, straw, weak, ruby, flabby*). This is similar to those found in the wine region hierarchy trees.

When comparing by frequency rank, the characteristic word floral does not have much variation between the three different grape varieties with rank 15, 16, and 16 for Pinot Noir, Red Blend, and Zinfandel respectively. However the

results in the hierarchy trees suggest that the relation of this attribute is more complex as the node position varies between the different grape varieties.

5 Conclusion and future work

In this paper, we proposed that by applying restricted bootstrapping for a set of characteristic words to a corpus of documents, a hierarchy tree representing the specificity and generality relation of the characteristics could be generated. This method was then applied to a corpus of wine tasting notes as an example of the analysis of differences in sensory expression characteristics of wine regions and grape varieties. Hierarchy trees were generated using the proposed matching restricted bootstrap search method for two wine regions, and three grape varieties. The results were then compared to the frequencies of the characteristic words as a naïve analysis baseline.

In future work, we plan to investigate the influence that the corpus size has on the relations of characteristics and the generated hierarchy trees. A formal method is also required to evaluate the effectiveness of extracting and generating specificity and generality relations of characteristic word sets.

6 Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 24500176.

7 References

[1] Caballero, R. 2007. Manner-of-motion verbs in wine description. *Journal of Pragmatics*, 39, 12, 2095-2114.

[2] Hirokawa, S. 2012. Feature Extraction Using Restricted Bootstrapping. *ICIS2013*, 283-288.

[3] Hirokawa, S., Flanagan, B., Suzuki, T., Yin, C. 2014. Learning Winespeak from Mind Map of Wine Blogs. In S. Yamamoto (Ed.): *Human Interface and the Management of Information Part II* (Springer LNCS 8522), 383-393.

[4] Hirokawa, S., Flanagan, B., Yin, C., Nakae, H. 2014. Visualization of relation and generality of words in research results. *ACIS2014*, 90-95.

[5] Kerren, A., Prangova, M., Paradis, C. 2011. Visualization of sensory perception descriptions. *Proc. of the 2011 15th International Conference on Information Visualisation IEEE*, 135-144.

[6] Komachi, M., Kudo, T., Shimbo, M., Matsumoto, Y. 2008. Graph-based analysis of semantic drift in Espresso-like bootstrapping algorithms. In *Proc. of the Conference on Empirical Methods in Natural Language Processing* (EMNLP 2008), 1011-1020.

[7] Mihalcea, R., & Tarau, P. 2004. TextRank: Bringing order into texts. *Proc. of Conference on Empirical Methods in Natural Language Processing* (EMNLP'2004), 404–411.

[8] Palshikar, G. K. 2007. Keyword extraction from a single document using centrality measures. *In Pattern Recognition and Machine Intelligence*, Springer LNCS 4815, 503-510.

[9] Pantel, P., & Pennacchiotti, M. 2006. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proc. of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, 113-120.

[10] Paradis, C. 2009. "This beauty should drink well for 10–12 years": a note on recommendations as semantic middles. Text and Talk - An Interdisciplinary Journal of Language, *Discourse Communication Studies*, 29, 1, 53-73.

[11] Paradis, C., Eeg-Olofsson, M. 2013. Describing Sensory Experience: The Genre of Wine Reviews. *Metaphor and Symbol*, 28, 1, 22-40.

[12] Radev, D. R., Mihalcea, R. 2008. Networks and natural language processing. *AI magazine*, 29, 3, 16-28.

[13] Salton, G., McGill, M. J. 1983. *Introduction to modern information retrieval*. McGraw-Hill.

[14] Tian, F., Yuan, C., Ren, F. 2012. Hyponym extraction from the web by bootstrapping. *IEEJ Transactions on Electrical and Electronic Engineering*, 7, 1, 62-68.

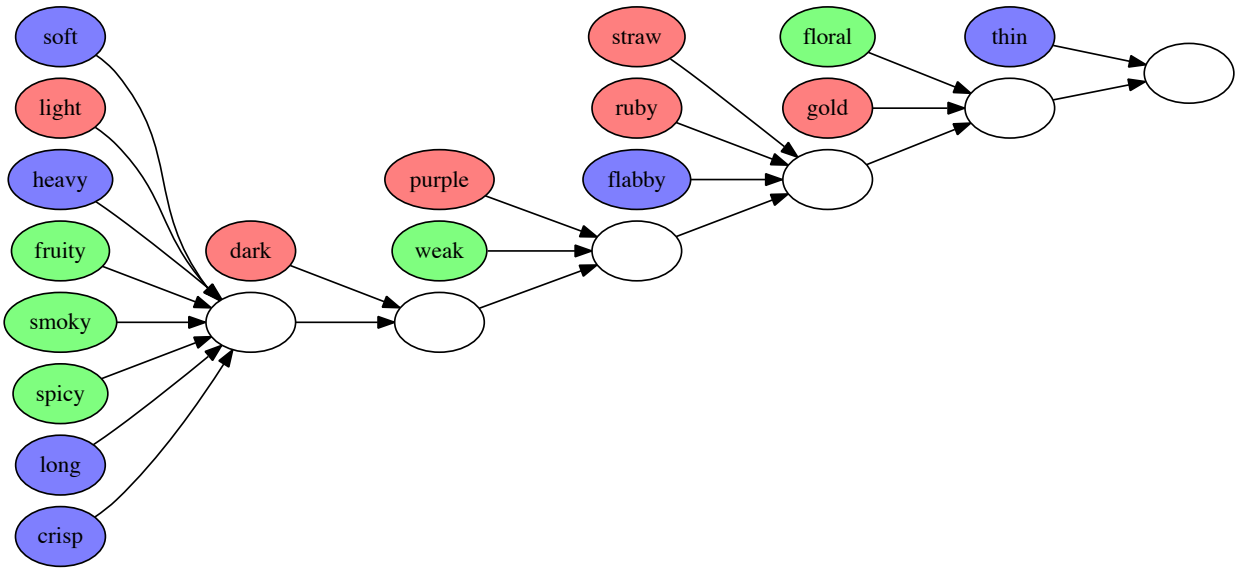


Figure 5. Hierarchy word tree of query word q = Marlborough

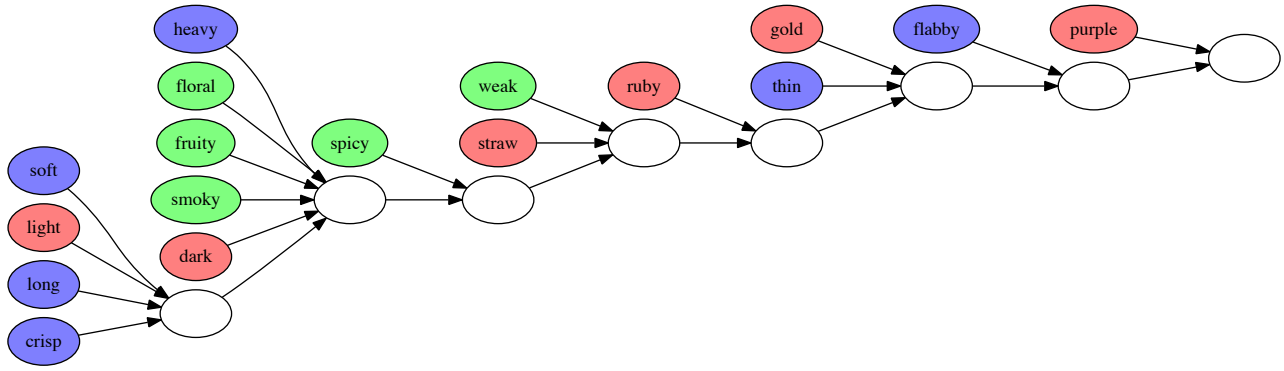


Figure 6. Hierarchy word tree of query word q = New Zealand

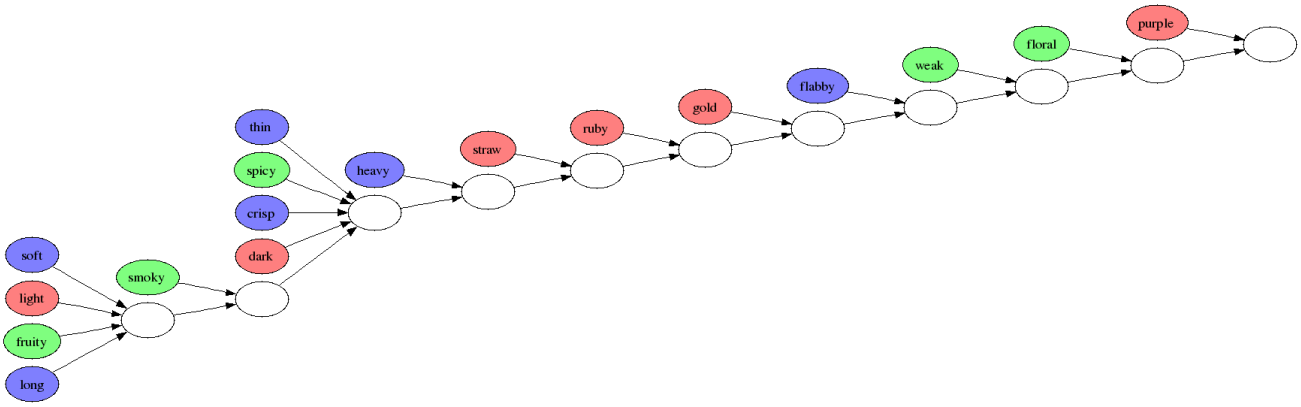


Figure 7. Hierarchy word tree of query word q = Pinot Noir

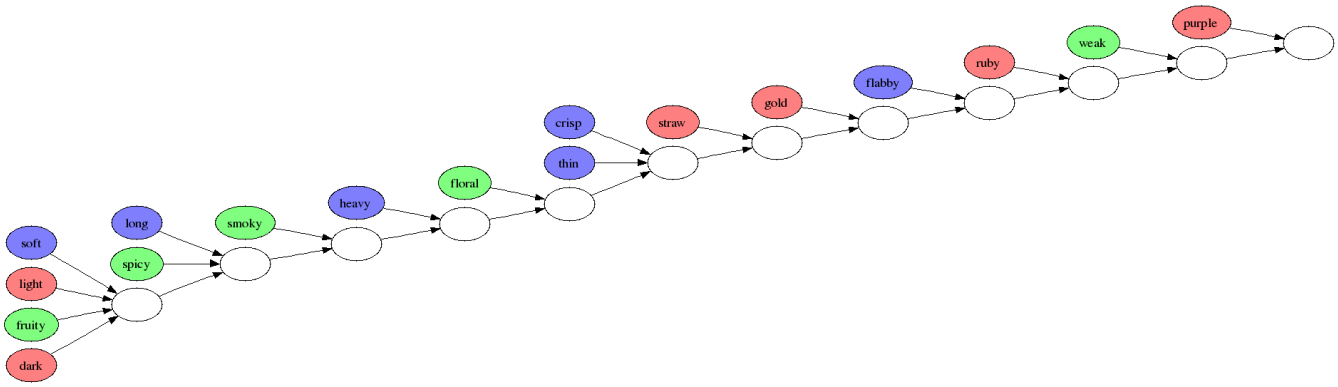


Figure 8. Hierarchy word tree of query word q = Red Blend

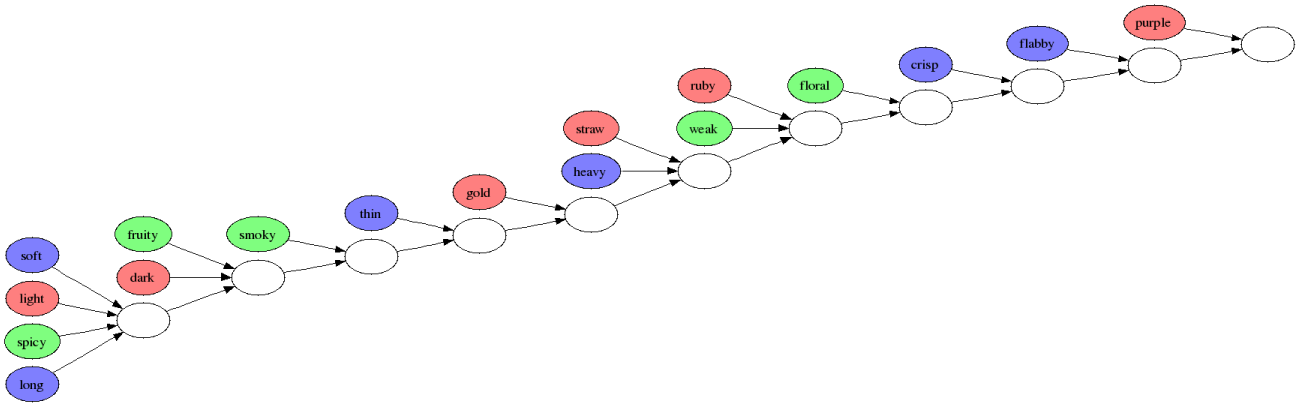


Figure 9. Hierarchy word tree of query word q = Zinfandel

Visualization of Sensory Weight for Shouldering Randoseru

Hitomi Oigawa, Yu Ishikawa, Masami Takata, Kazuki Joe
Nara Women's University, Nara, 630-8506, JAPAN

Abstract – *In this paper, we propose an evaluation method for sensory weight of randoserus. We measure the pressure distributions of the back of kindergarten children and elementary school pupils right after their shouldering a randoseru and after the behaviors such as bowing and walking with shouldering the randoseru. We propose 4 evaluation indexes: the burden ratio of the back, the number of pressure sensing elements, the variance of each pressure sensing element by row, and the variance by column. We visualize the evaluation indexes of three types of randoserus by a radar graph. As the result, it is possible to intuitively understand which randoseru gives the lightest sensory weight.*

Keywords: pressure sensing device, sensory weight, visualization

1 Introduction

Various cultures, traditions and products of Japan have been known as "COOL JAPAN". Recently in Japan, a considerable number of foreign tourists buy randoserus, which are a school bag for elementary school students in Japan and known as a COOL JAPAN product. Current randoserus are decorated with a variety of colors and delicate embroideries that make overseas youth have a lot of attention. A famous Hollywood actress with shouldering a randoseru as a fashion was reported by CNN in Sep. 2014. Japanese anime also give a great influence to foreigners to buy randoserus as a cosplay tool because some anime characters shoulder randoserus.

Apart from the fashion, some foreigners attach importance to children's safety that is the basic function of randoserus. A randoseru is a bag for a child who enters the first grade of an elementary school. However, since randoserus should be tough and kept for 6 years (the period of the elementary school in Japan), it is difficult to reduce the weight of randoserus. For example, randoserus are designed to be durable and thick in order to protect child's occipital area when he/she is overturned. There are many handmade parts by craftspeople such as cutting, sewing and finalizing randoserus fabric. Craftspeople polish with a day-to-day technology to keep the reliable quality.

In Japan a lot of randoseru manufacturers produce different features of randoserus. When someone is going to buy a randoserus, the price, color, design, inner dimension, and ruggedness of the randoseru are considerable issues. In particular, a burden less design for an elementary school pupil is an important key [1]. For pupils with a small body to take a long commute time, a heavy school bag is physically unsuitable [2] [3] and believed to affect the growth of the child [4]. Therefore, randoseru manufacturers continue to use their thinking mind to solve the problem by reducing the sensory weight, which is the weight that is perceived to feel on the shoulders rather than the weight of the bag itself, of their products.

When someone considers the purchase of the lightest sensor weight randoseru for his/her child, all the targetable randoserus need to be actually shouldered by the child for the comparison. However, the child (preschool year old) is too yang to judge the sensory weight because the number of randoseru manufactures and randoseru types is too large to compare their shouldering tests. After all, parents or grandparents tend to buy a randoseru for his/her child or grandchild recommended by the randoseru shop clerk. Namely, when buying a randoseru, there is a selection problem that has not been solved because of the age of users. If the sensory weight of a randoseru is objectively quantified to be visualized for ordinal people, the problem is to be solved.

In this paper, we propose an objective quantification method for the sensory weight of randoserus so that it can be visualized for ordinary people to intuitively understand. We use a pressure sensor sheet for the quantification to measure the pressure distribution when shouldering a randoseru. The measurement results are analyzed with the following 4 items: the burden ratio of the back against the total randoseru weight, the number of pressure sensing elements detecting any pressure, the variance of each pressure sensing element by row, and the variance by column. The analyzed results are visualized with a radar graph for intuitive understanding.

The rest of the paper is organized as follows. In section 2, we describe the study for the sensory weight of randoserus in Japan. We describe the method for measuring the sensory weight in section 3. The result and the discussion of experiments are presented in section 4.

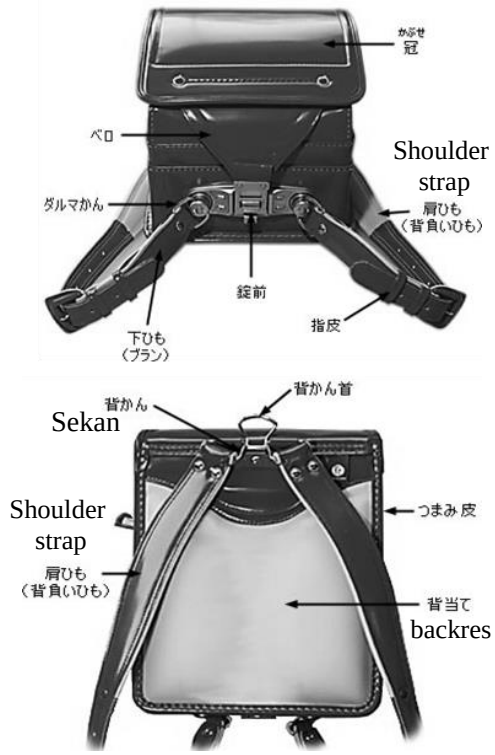


Figure 1 Part names of randoseru

2 Related Work

As for the weight of randoserus, one of direct factors is what kind of materials the randoseru is made of. A thinner material makes the randoseru lighter but decreases the intensity. Therefore, it is difficult to simply decrease the weight of randoserus.

The sensory weight is different from the physical weight. When someone holds a bag, he/she feels different weights by changing the way of holding the bag. In the case of shouldering a randoseru, the sensory weight depends on how much it has close contact with the back; the less contact, the stronger sensory weight.

Figure 1 shows the part names of randoseru. The parts mainly related to the sensory weight include backrest, shoulder straps, and Sekan. A backrest is the cushion part of a randoseru which has close contacts with the back. Sekan is a joint bracket that connects the shoulder straps and the randoseru body. If we just focus on the sensory weight, the backrest should be preferably flat. However, considering the asperity of the back and the stuffiness in the summer, the just flat backrest may affect the fitness and the breathability. Shoulder straps are a part that directly contacts the entire shoulders and significantly affect the sensory weight as the backrest. Considering the fitness and the balance in the right and the left shoulders, the mobility of Sekan is most important [5].

We explain three randoseru products: "Tenshino Hane", "Fit Chan", and "Hanessel". Each product has some attempt to decrease the sensory weight.

"Tenshino Hane" by Seiban Co., Ltd. [6] provides a mechanism to fit the randoseru at the center of the back using a special Sekan that controls to extend the shoulder straps equally to the left and right. Furthermore, the inner and the outer calibers of the shoulder straps are designed to be short and long, respectively, so that the randoseru has larger contact area to the child with more balanced weight to decrease the sensory weight. Even if the child flounces naughtily, the center of the randoseru is stationary and the sensory weight is light.

"Fit Chan" by Hashimoto Co., Ltd. [7,8] provides a mechanism to distribute the randoseru's pressure from the shoulders to the back so that the sensory weight is light. Concretely speaking, the shoulder straps are hanged up 25 degrees so that the contacting area to the back increases. Furthermore, their research collaboration with Shinshu University proposes a method of decreasing the weight on the shoulders with making Sekan and shoulder straps round.

"Hanesseru" by KMW Co., Ltd. [9] provides load balancing using the randoseru design based on the analysis of the body structure of the child. The shoulder straps have 38 stage adjustments function by 5mm pitch so that the randoseru is fit to subtly tilted shoulders. The length of the backrest is set to 28cm because they believe it is the right length to make the randoseru fitting to the child's back.

In the case of overseas, randoserus are sometimes studied instead of usual school bags. [10] studies the burden on the back when a child wears a school bag.

With respect to the sensory weight measurement method, the changes in oxygen uptake, changes in the EMG, changes in the heart rate, the change in body pressure distribution, and the subjective declaration before and after randoseru shouldering are reported [11]. In general, since vital data are related each other, we do not think all the above "changes" are required. Furthermore, the oxygen uptake, the EMG, and the heart rate are affected by the psychological factors such as degree of tension. Therefore, those data are not suitable for the sensory weight measurement for shouldering a randoseru. In addition, the subjective declaration is not also suitable because of the age of the users. Thus, the sensory weight of a randoseru should be measured by the body pressure distribution because it is less sensitive to the state of the body and gives the accurate measurement as numeric.

3 Measurements and analysis

3.1 I-SCAN

For the measurement of the randoseru pressure distribution to the back of children, we use a surface pressure

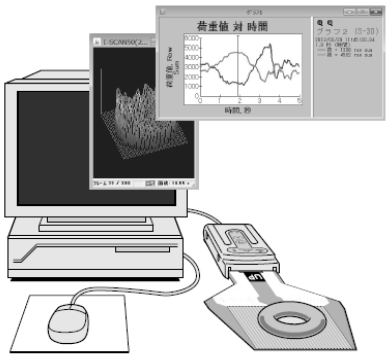


Figure 2 I-SCAN

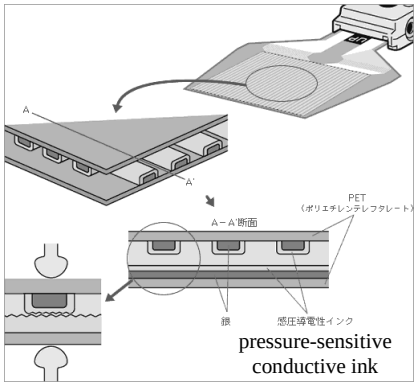


Figure 3 Sensor sheet structure

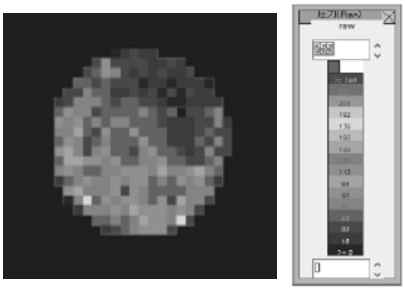


Figure 4 the pressure scale by 16 color mapping

distribution measurement system I-SCAN as shown in Fig.2 [12]. It consists of software and hardware that includes a sensor sheet and a sensor connector. The sensor sheet measuring the pressure is formed with two thin films that are coated with pressure-sensitive conductive ink as shown in Fig.3. The ink changes its electrical resistance according to the pressure. On the films, column electrodes and row electrodes are evenly arranged as a matrix form on the top and the bottom. Each intersection between the column and the row electrodes is a measurement point. In other words, I-SCAN calculates the pressure by measuring the electrical resistance of the conductive ink, and sends electric resistance values of the measurement points to the PC connected via USB. When transmitting, each electric resistance value is converted into a digital signal with 256 steps (resolution: 8bits).

Figure 5 Measurement by an I-SCAN

The attached software is used for visualizing the digital signals in real time up to 100Hz sampling frequency. The visualized data is the pressure scale and the electrical resistance. Figure 4 present the representation of the pressure scale by 16 colors mapping when a circular load is detected. The calibration function is used for the unit conversion from the electric resistance (raw) to the pressure value (KPa). Furthermore, a noise cut method is available for narrowing down the measurement range. The measurement results are saved with a format selected from ASCII, AVI, JPEG, and MATLAB.

In this study, the size of the sensor unit is 238 × 238 (mm) and the sensor consists of 44 × 44 elements. The calibration result shows the measurement range is 0 ~ 271kPa and the each pressure value is saved in the ASCII format for evaluation.

3.2 Measurements

As a prerequisite, we put several books of which weigh is 2 kg in a randoseru. The weigh is the average of the total weighs for the first year elementary school pupils [6]: textbooks and writing utensils. The measurement method is shown in Fig.5. When shouldering a randoseru, the shoulder straps are set perpendicular to the ground and the backrest is adjusted in close contact with the back of the examinee so that the examinee does not feel painful. An I-SCAN sensor sheet is taped on the backrest in advance, and we make the examinee shoulder a randoseru with the sensor sheet. At this time, before mounting it, we confirm that there are not pressures detected using the attached software.

Measurements of the pressure distribution are performed twice per randoseru. The first is measured immediately after mounting when the examinee is upright while the second is measured when the examinee is in an upright state right after the operations of bowing or walking with shouldering the randoseru.

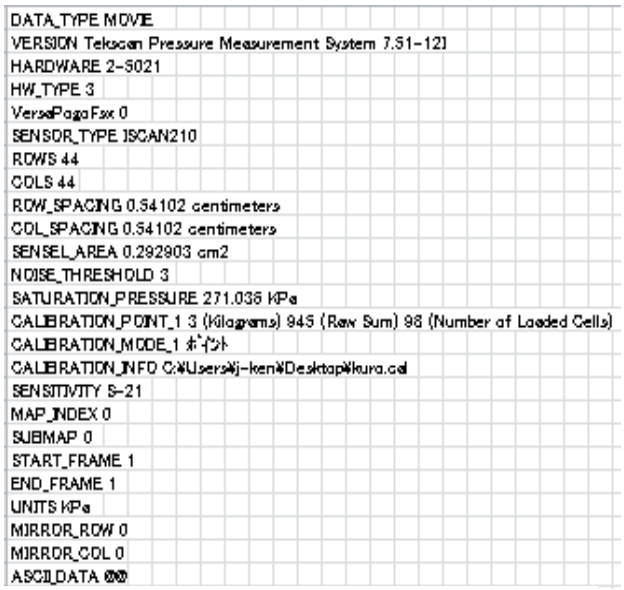


Figure 6 Results of measurement, header part

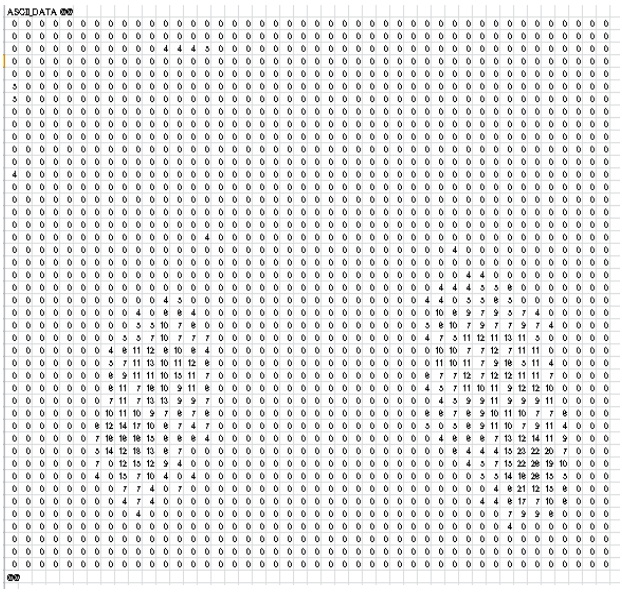


Figure 7 Results of measurement, pressure values

3.3 Analysis

Figure 6 and 7 show the contents of the file that includes the results of a measurement with an I-SCAN in the ASCII format. Figure 6 shows the header part while Fig.7 shows the result of the pressure values measured at each measurement point of 44 × 44. In the header, the types of sensor sheet and the calibration data, etc., are described. After the header, pressure values for a measurement point of 44 × 44 are recorded as an array of 44 × 44. In the matrixes used in the following evaluation indexes, a row i (= 0, 1, ..., 43) is an observation point at the shoulder direction from the waist in the ascending order while a column j (= 0,1, ..., 43) is an observation point at the right direction from the left in the ascending order.

In the analysis, we focus on the following four items: the burden ratio of the back against the total randoseru weight R, the number of pressure sensing elements S_E, the variances of each pressure sensing element by row V_i, and by column V_j.

The burden ratio of the back against R is expressed by the following equation.

$$S_P = \sum_{i=0}^{n}\sum_{j=0}^{n} p_{ij} \quad (1)$$

$$S_w = 0.01 S_P a \ \ (1kPa = 0.01kg/cm^2) \quad (2)$$

$$\mathrm{R} = S_w / T_w \quad (3)$$

where S_P is the sum of measured pressure values and p_{ij} is the pressure value of each measurement point. By adding all the pressure values, the total pressure value of the back is calculated. Since the matrix is 44 × 44, $n = 43$ is used. Also S_w represents the weight to the back with multiplying the area with the total pressure value. a represents an area for a corresponding sensor. Namely, $a = 23.8^2/44^2$ is used. T_w shows the total weight of the randoseru.

The number of pressure sensing element S_E is the number of observation points detected greater than zero pressure in the observation point 44 × 44.

The variance V_i of each pressure sensing element that focuses on the row is expressed by the following equation.

$$V_i = \frac{1}{S_E}\sum_{m=1}^{S_E}(i_m - \bar{\imath})^2 \quad (4)$$

where i_m is the column number of the i-th row and j-th row element detecting pressure value p_{ij} and $\bar{\imath}$ is an average value obtained by dividing the number of pressure sensing elements S_E with the sum of the row number of the pressure sensing elements. The variance is calculated by the mean square of the differences between i_m and $\bar{\imath}$.

The variance V_j of each pressure sensing element that focuses on the column is expressed by the following equation.

$$V_j = \frac{1}{S_E}\sum_{m=1}^{S_E}(j_m - \bar{\jmath})^2 \quad (5)$$

where j_m is the column number of the i-th row and j-th element detecting pressure value p_{ij} and $\bar{\jmath}$ is an average value obtained by dividing the number of pressure sensing elements S_E with the sum of the column number of the pressure sensing

elements. The variance is calculated by the mean square of the differences between j_m and $\bar{j}$.

3.4 Evaluation indexes

A randoseru is supported by the back and the shoulders. When the pressure is to be more dispersed in the back and the shoulders, it is possible to support the randoseru on the entire upper body and the sensory weight decreases, namely, it leads to feel easy to shoulder the randoseru. In particular, the supporting back is divided by the top around the shoulder blade and the bottom around the waist. In other words, when the randoseru comes into contact with the upper back, the gravity center verge the bottom and it make the examinee stand straight and feel that the sensory weight is light. Conversely, when the randoseru is not in contact with the back so much, the randoseru is pulled back and it makes the examinee be a position of anteversion and feel a burden on the lower part of the shoulders and the back [11]. From the above investigations, we find two important things for the reduction of the sensory weight with a randoseru: 1) the variance of pressures on the back and the shoulders and 2) the contact area between the back and the randoseru.

As for the variance, we evaluate it by using the ratio *R* of the burden ratio of the back S_w against the total randoseru weight T_w. Incidentally, the ratio *1-R* is the burden except the back and is the ratio of the pressure of the shoulder straps on the shoulders.

As for the contact area, we calculate it using the number of pressure sensing elements S_E. With the use of V_i we can see how many elements are dispersed toward the shoulder blades from the waist within the back while with the use of V_j we can see how many elements are dispersed from the left to the right. These two variance values are used for the calculating how the pressure sensing elements in the entire back are dispersed and it is possible to infer the effective area that supports the randoseru.

To evaluate multiple randoserus, we compare the analytical results by examinee and analysis item. The analytical results are normalized (0-100) for easy comparison.

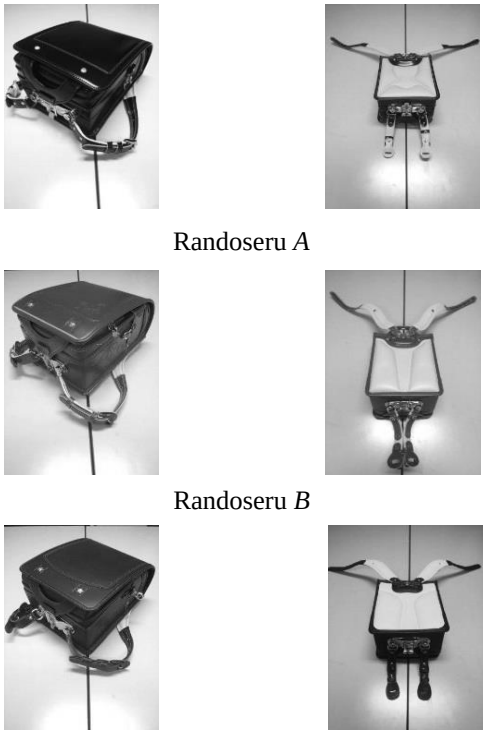
4 Experiments

By using the evaluation indexes described in subsection 3.4, we compare multiple randoserus. The indexes are expressed as follows.

(1) The burden ratio of the back against the total randoseru weight

(2) The number of pressure sensing elements

(3) The variance of each pressure sensing element by row



Randoseru *A*

Randoseru *B*

Randoseru *C*

Figure 8 Three Randoserus

(4) The variance of each pressure sensing element by column

4.1 Examinees

The examinees are kindergarten children or elementary school pupils, proviso from 5 years to 7 years old. The number of the examinees is 20 (10 boys and 10 girls). The reason for limiting the examinee's age from 5-year-old to 7-year-old is that 6-year-old children start to use randoserus. Actually, the average heights of the examinee groups are boys 117.1 ± 4cm (112-122cm) and girls 116.1 ± 4cm (111-121cm). The school health statistical survey reports that the 6-year-old average height of boys is 116.6 ± 5cm and girls is 115.8 ± 5cm [13]. Provided with the 1% significance level, the test statistic for boys is 0.3188 (≤2.576) and girls is 0.3274 (≤2.576). So we confirm that there is no significant difference between the examinee groups.

4.2 Randoseru

It takes approximately 20 minutes for the measurement, and the examinees bear about 3kg; the weight of the randoseru itself and text books. We use three kinds of randoserus for the comparison with considering the examinees' ability to concentrate against the burden.

We use randoserus *A*, *B*, and *C* as shown in Fig.8. All the randoseru material is Clario. The weights of *A*, *B*, and *C* are 900g, 1,100g, and 1,100g, respectively. Each randoseru is

Table 1 Analysis result right after shouldering

	(1)	(2)	(3)	(4)
A	28.55	41.05	195.49	117.78
B	38.86	57.90	218.19	128.85
C	36.98	54.45	137.53	141.54

Table 2 After behaviors of bowing and walking

	(1)	(2)	(3)	(4)
A	22.08	32.20	188.74	129.76
B	44.52	63.15	204.36	121.87
C	30.62	45.10	175.63	123.46

Table 3 comparison of each Randoseru of two states

	(1)	(2)	(3)	(4)
A	-6.48	-8.85	-6.75	11.98
B	5.66	5.25	-13.84	-6.98
C	-6.36	-9.35	38.11	-18.09

manufactured by different makers. The differences in each randoserus include the form of backrest, the positions of shoulder strap's cushion and the belt hole, and the mobility of the shoulder straps using Sekan.

4.3 Measurement result

Table 1 shows the analysis results right after shouldering the randoserus. The row and the column represent the analysis items and the types of randoseru, respectively. *A* is the worst about (1) and (2). From the viewpoints of (1), (2), and (3), *B* is the best. *C* is the best for (4) while the worst for (3).

Table 2 shows the analysis results after the behaviors of bowing and walking while shouldering randoserus. *A* is the best for (4) while poor in (1) and (2). From the viewpoints of (1), (2), and (3), *B* is the best. *C* is the worst for (3).

Table 3 shows the comparison of each randoseru of two states. Each value is obtained by subtracting the values of Tab.2 with Tab.1. *A* is better in (4) while (1) and (2) decrease significantly. As for *C*, (1) and (2) decrease significantly as same as *A* while (3) is the best. *B* is the best for (1) and (2); they increase unlike *A* and *C*. However, (3) is the worst and (4) is not so good.

The results of comparison of three randoserus are presented as a radar graph in Fig.9. The averages for all the items are 63.85, 86.12, and 71.81 for *A*, *B*, and *C*, respectively.

4.4 Discussions

In Tab.1, it is observed that *A* forces the most intense shoulder burden because both (1) and (2) are the lowest. On the other hand, we find that *B* is well balanced although the vertical range is not in effective use.

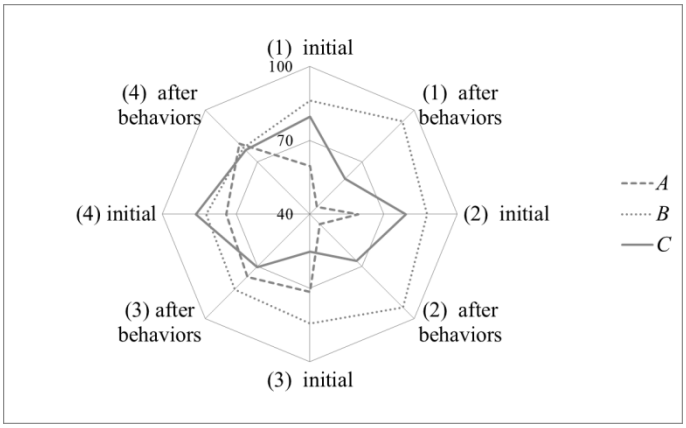


Figure 9 Visualize each evaluation index for three randoserus by a radar graph

In Tab.2 and 3, which describe the state changes between initial and after behavior, since randoserus *A* and *C* are less in (1) and (2), and the contact area between the back and the randoserus is reduced, it is regarded that the load on the shoulders increases. In the meanwhile, (1) and (2) of *B* increase after the behavior, so the contact area between the back and the randoserus increases. Since the load on the shoulders is regarded as dispersed, the sensory weight becomes lighter by the behavior. Namely, by the behavior, *B* fits to the body and examinees feel easy to shoulder the randoseru. Furthermore, although (3) is reduced compared with *A* and *C*, the values themselves of (3) are the highest. Thus we conclude *B* is the best randoseru.

In Fig.9, the higher the evaluation index values are, the lighter the sensory weight is. *A* shows the lowest value for most items. Thus, although the weight of randoseru itself is lightest among the three types, the sensory weight is the largest and we conclude *A* is a large burdened randoseru. For *B*, from the fact that it achieves the highest value for the most items, the sensory weight is lighter. For *C*, the initial shouldering randoseru state is relatively close to the state of *B*, but after the behavior *C* is isolated heavily rather than *B*. Therefore, from the viewpoint of shouldering time, we conclude *C* is a randoseru that gives a considerable burden to elementary school pupils.

The averages of values for all the evaluation items are the same order, too. From the foregoing results, we conclude the lightest sensory weight is given by *B*. Further, the visualization of analysis results with a radar graph intuitively explains the sensory weight.

5 Conclusions

In this paper, we measure and analyze the pressure distribution on the back with a randoseru to visualize the sensory weight of randoseru. As the measurement conditions, we measure the pressure distributions at the time examinees shoulder a randoseru and the time after the behaviors of

bowing and walking. 20 examinees from 5 to 7 years old shoulder three types of randoserus to measure the pressure distributions using a pressure sensor sheet. To measure the pressure distribution, four evaluation indexes are used: (1) the burden ratio of the back, (2) the number of pressure sensing elements, (3) the variance by row, and (4) the variance by column. From the results of the experiments, we found that type *B* randoseru gives the lightest sensory weight. It is intuitively presented by the visualization, too.

Our future work includes more evaluation indexes for improvement posture using a three-axis accelerometer. In concrete, mounting a three-axis acceleration on children's back, we calculate the angle of the back to the ground at the time of shouldering randoserus and after several behaviors. By comparing various randoserus, we show an evidence to "Which randoseru is the best?".

6 References

[1] "kuraray Randoseru purchaser questionnaire in 2011" [online]Available:http://www.kuraray.co.jp/enquete/ransel/2011/data1.html

[2] Pascoe,D.D,Pascoe,D.E.,Wang,Y.T.,shim,D.-M.,and Kim,C.,K., "Influence of carrying book bags on gait cycle and posture of youths", Ergonomics, vol.40, issue 6, 631—641,1997

[3] Minako Yoshida, Yoshie Shibata, Maya Tanaka, Kaori Tanaka, and Kozo Hirata, "Effects of Prolonged Rucksack-Atrap pressure on Blood Flow and Pressure Sensation", Descente Sports Science, Vol.20,184—191,1999

[4] Clare Haselgrove, Leon Straker, Anne Smith, Peter O'Sullivan, Mark Perry, and Nick Sloan," Perceived school bag load, duration of carriage, and method of transport to school are associated with spinal pain in adolescents: an observational study", Australian Journal of Physiotherapy, vol.54,issue 3, 193—200,2008

[5] "Japan bag Association Randoseru Industry Association Randoseru Nomenclature "
[online]Available: http://www.Randoseru.gr.jp/check1.htm

[6] "Features of Seiban's Randoseru",
[Online]Available: https://www.seiban.co.jp/six/function.html

[7] "Fit-chan Randoseru Secret of feeling light"
[online]Available:
http://www.fit-chan.com/reason/durable/durable01.html

[8] Harumi Morooka, Tomoko Kawakami, and Hideo Morooka, "Discussion on School Bags Using Shoulder Strap Pressure for Reducing Body Load", SEN'I GAKKAISHI, vol.65 No.12, 325—331,Des 2009

[9] "harnessel 2015 model "
[Online]Available:
http://harnessel.jp/new_model_2015/index.html

[10] Hamish W.Mackie, Joan M. Stevenson, Susan A.Reid, and Stephen J.Legg," The effect of simulated school load carriage configurations on shoulder strap tension forces and shoulder interface pressure", Applied Ergonomics, vol.36, issue 2, 199—206,Mar 2005

[11] " Development of comfortable school for bags "
[online] Available: http://www.laponte.co.jp

[12] "the surface pressure distribution measurement system I-SCAN"
[Online]Available:http://www.nitta.co.jp/?post_type=sensor&p=7434&fnkey=product

[13] " The school health statistical survey reports 2013 fiscal national table"
[online]Available:http://www.e-stat.go.jp/SG1/estat/List.do?bid=000001052598&cycode=0

Recipe clustering based on Japanese Food Guide Spinning Top

Yasuhiro Tajima[1], Yoshihiro Suwa[2], Genichiro Kikui[1], Rikako Inoue[3] and Megumi Kubota[3]
[1]Department of Systems Engineering, Okayama Prefectural University, Soja, Okayama, Japan
[2]Graduate School of Comp. Sci. and Sys. Eng., Okayama Prefectural University, Soja, Okayama, Japan
[3]Department of Nutritional Science, Okayama Prefectural University, Soja, Okayama, Japan

Abstract— *Food recipe site is one of the hottest web service in recent years. Especially, a user posting style recipe site is the most popular service style. Such a recipe site contains thousands recipes posted by users, and the variation is very wide such that daily meal, take out foods, special conditioned meal for allergy and so on. On the other hand, Japanese government prepares an illustrated tool called "Japanese Food Guide Spinning Top" for selecting daily foods. There are some classes of foods and if you select foods from each class then daily food balance will be kept good. We propose an automatical method to classify a recipe to a class of the foods in Japanese Food Guide Spinning Top. Experimental evaluation is done and we obtain about 70% of accuracy as the result.*

Keywords: Rakuten Recipe, Japanese Food Guide Spinning Top

1. Introduction

Recipe sites are the hottest web service, now. There are many service sites and we can summarize them into the following types.

- User posting style : Service site is operated by some companies but recipe contents are constructed from users' post. Various and many recipes can be gathered but it is difficult to keep the service provider's policy for each recipes. There are many similar recipes and reliability is also low.
- Operated by an organization : one company or one organization controls all contents and recipes. For example, if a cooking school operates a recipe service site, then they can present nutrition values of each recipe. The balance of recipes can be controlled. For example, the site operator can select vegetable recipes as many as meat recipes.

Now, user posting style is more important for web technologies because there exists big data and remains many unorganized data. There are many related studies for user posting style recipe sites. In [2], the title of a recipe is evaluated whether it matches the recipe's content. In [3], using recipe clustering, subsutitutional ingredients are found from the recipe repository.

On the other hand, Japanese Food Guide Spinning Top is provided by Japanese government. This is a tool for selecting daily foods to balance nutrition. In this tool, there are 6 class of foods and if we take appropriate quantity from each class per day, then we can keep healthy life.

In this paper, we propose a mapping method from a recipe to the class on Japanese Food Guide Spinning Top. Then experimental evaluations are also done.

2. Japanese Food Guide Spinning Top

Japanese Food Guide Spinning Top[1] is an illustrated tool for daily food choice. This tool is provided by Japanese Ministry of Agriculture, Forestry and Fisheries. In the following, we call this tool the Guide for short.

Figure 1 is the illustration. This guidance consists of the following components.

[Spinning Top](at the left side of the figure)
The illustration expresses the balance of daily food. It represents the quantity image of daily eating. This spinning top is divided into following three dishes and three items.

1) Grain dishes
This dish provides carbohydrates. For example, rice, bread, noodles and so on.
2) Vegetable dishes
This dish provides vitamins, minerals or dietary fiber. For example, vegetables, tubers, corms, bulbs, mushrooms, seaweeds and so on.
3) Fish and Meat dishes
This dish provides proteins. For example, meat, eggs, fishes, beans and so on.
4) Milk
This item provides calcium. Milk and dairy products are classified into this item.
5) Fruits
Any fruits are classified into this item.
6) Sweets
We can also take little sweets.

These dishes and items are called the "class" in the Guide. Every class contain some example figures of foods. The center axis of the spinning top represents water. Healthy life can be kept since we make the spinning top rounding and standing.

[Target SV(serving) values](at the middle of the figure)
The SV value (which means SerVing value) is the target quantity for every class in the Guide. One SV means about one meal. Target values are set as follows.

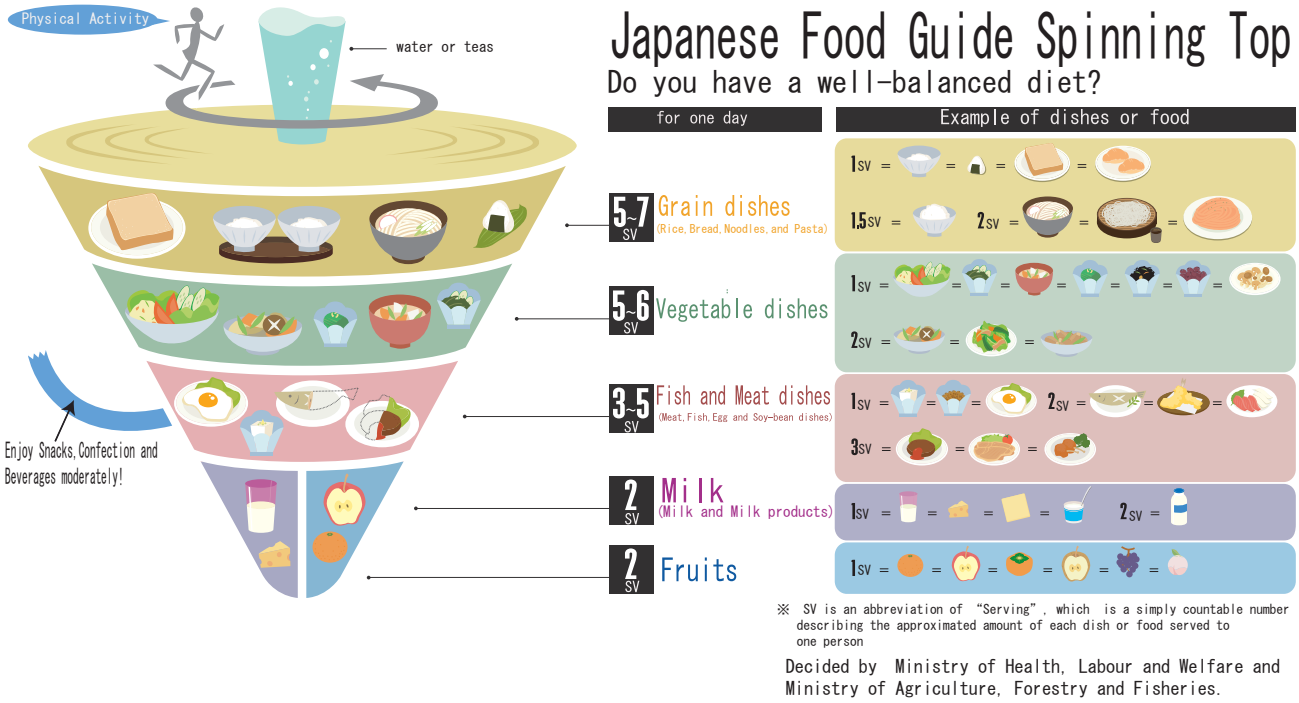


Fig. 1: Japanese Food Guide Spinning Top

- Grain dishes : 5 to 7 SV
  Here, 1 SV is about 40g of carbohydrates. Thus 5 to 7 SV per day means 200g to 280g carbohydrates to eat per day. In Japan, rice is served by a rice bowl which contains about 100g, and 100g of rice contains about 40g carbohydrates, then a regular size of rice bowl serves 1 SV.
- Vegetable dishes : 5 to 6 SV
  Here, 1 SV is about 70g of foods' net weight in this class.
- Fish and Meat dishes : 3 to 5 SV
  Here, 1 SV is about 6g of protein. It is hard to find the net weight of protein which is contained in a food. Thus, there are some examples to count this SV value. An egg dish has 1 SV. A fish dish has 2 SV. A meat dish has 3 SV. This is very rough counting method, but easy counting is prior to accuracy.
- Milk : 2 SV
  Here, 1 SV is about 100mg of calcium. When we use this tool, we regard a cup of milk as 1 SV.
- Fruits : 2 SV
  Here, 1 SV is about 100g of foods' net weight. In many case, we count 1 piece of fruit as 1 SV.
- Sweets does not have SV value, because little quantity is allowed to eat per day.

[Examples of dishes and SV value](at the right of the figure)
This area represents examples of dishes and example SV values of dishes. There are some figures of foods and whose SV values. The SV values are aimed to balance the quantity of daily eating but finding precise values is difficult for every dishes. Thus, such illustrated guide of dishes are useful to find rough SV values. Such roughness is more important to use this tool continuously.

With this illustrated tool, we can count the total SV values for each meal or the total value of every day. We can take

balanced meals by aiming the SV values of each dishes or items at the target values.

3. Classification of Recipes

3.1 Rakuten recipes

Rakuten Recipe is one of the most popular recipe website in Japan. And there are repositories of recipes for research or study use[5] called Rakuten data.

Fig. 2 is a sample page of Rakuten Recipe. There are following compositions.

[Recipe Title] : This title is set by the user who submit the recipe to Rakuten recipe site. So, it tends to more impressive word will be written by the author.

[Photos] : This photo is the dish after cooking. This is taken by the author of this recipe and usually dish quantity is as same as the following ingredients list.

[Ingredients List] : The list of ingredients also provided by the author of this recipe. Every entry of the list consists of a ingredient's name and its quantity. These name and quantity are also provided by the author of this recipe.

[Cooking Procedure] : Cooking procedure consists of the text of the process and its photo. Some steps does not have a photo and only text description is remained.

[Comments] : In this site, user can add a comment to the recipe which is called "making report" (in Japanese "Tsukurepo").

[Categories] : Every recipe has its three categories, that is big category, middle category and small category. The big category has the following nine names:

- Grain dish
- Side dish
- Snacks
- Takeout foods
- Sauce or Jam
- Drinks
- Season foods
- Local foods
- Special purpose

Middle and small categories are defined for every big category. For example, the middle category for Snacks is as follows.

- Snacks/Cakes
- Snacks/Foreign snacks
- Snacks/Japanese snacks
- Snacks/Others

Small category is defined for every middle category, then there are 733 small categories. The total number of middle categories is 61.

There are about 410 thousand recipes in Rakuten data.

3.2 Making correct classification

Using the category labels, we make a correct data of the Guide. The correct data is hand made and the following is how to make the correct data.

1) If the big category is identical to a class in the Guide, then all recipes in the big category is marked by the class name in the Guide. For example, the big category "Grain dish" can obviously be classified into "Grain dishes" class in the Guide.
2) If the big category can not be decided into one class in the Guide then, using middle category, we try to select a class in the Guide. For example, recipes with the middle category of "Snacks/Cakes" are classified into "Sweets" class in the Guide.
3) If we can not decide one class by the above step then, using the small category, we try to select a class in the Guide.
4) If we can not decide one class using small category then all recipes with the small category is classified int "Others" class.

With the above process, we can classify all recipes in Rakuten data into classes of the Guide.

Table 1 shows the number of recipes of each class in the Guide.

3.3 Classification method

Classification is done by Support Vector Machine(SVM for short)[4] in our experiments. The kernel is not used (=linear) and soft margin parameter is found by preliminary experiment.

The input vector of the SVM has the size of the vocabulary of learning data. It means that each element of an input vector is identical to a word of learning data. All texts in learning data are morphological analyzed by mecab[6]. The value of each element of an input vector is TF-IDF value where a recipe is a document.

For multi-label classification, we use one-versus-rest method with SVM. In this setting, we make n SVMs for n-class classification. Then, it is selected that the class whose SVM has the longest distance from the hyperplane. With this condition, we evaluate the performance of the classification method.

4. Experiments and Results

At first, we make input vectors from restricted morphemes. For the texts of cooking process and the table of ingredients, we restrict the following morphemes. The input vectors are constructed from one of these combinations.

- verb
- noun
- noun + verb
- noun + verb + adjective

Fig. 2: Rakuten Recipe sample page

Table 1: The number of recipes for each class in Japanese Food Guide Spinning Top

Grain dishes	Vegetable dishes	Fish and Meat dishes	Milk	Fruits	Sweets	Others
91436	134187	96967	3817	2143	44609	37819

Evaluation is done by 5-fold cross validation. All results are the average of these 5 tests. Table 2 shows the accuracy of the output of one-versus-rest method. Here, accuracy is calculated by the following equation.

$$accuracy = \frac{correct\ inferred\ recipes}{all\ test\ recipes}$$

"correct inferred recipes" is the number of recipes whose correct class corresponds to the inferred class.

Bigger size of input vector marks high performance from this result. Especially, the accuracy becomes over 70% when the size is bigger than 20000.

Next, we show the performance of each SVM which is two values classifier. In one-versus-rest method, there are 7 SVMs for each class of the Guide. Each of them classifies that the input recipe is in the class or not. In the previous experiment, we select the final class by the distance from the hyperplane. In the next tables, we show the performance of each SVM. Table 3, 4 and 5 show precision, recall and F-value of each SVMs, respectively.

Here, precision, recall and F-value is calculated by the following equations.

$$\begin{aligned} precision &= \frac{correct\ inferred\ recipes(each\ class)}{positive\ inferred\ recipes} \\ recall &= \frac{correct\ inferred\ recipes(each\ class)}{positive\ labeled\ recipes} \\ F &= 2/((1/recall) + (1/precision)) \end{aligned}$$

In this experiment, we are concerned with one target class. "correct inferred recipes(each class)" is the number of recipes which is correct inferred on the target class. "positive inferred recipes" is the number of recipes such that the SVM for the target class outputs the positive label. "positive labeled recipes" is the number of recipes whose correct class is the target class. For example, assume that the target class

Table 2: Accuracy of restricted morphemes

morphemes	input vector size	accuracy
verb	4961	0.5864
noun	21830	0.7658
noun + verb	25003	0.7722
noun + verb + adj	25410	0.7726

Table 3: Precision with restricted morphemes

morphemes	Grain	Vegetable	Fish and Meat	Milk	Fruits	Sweets	Others
verb	0.6293	0.5580	0.5778	0.4262	0.0000	0.6470	0.4944
noun	0.8480	0.7383	0.7439	0.6100	0.4036	0.7708	0.6714
noun + verb	0.8531	0.7476	0.7506	0.5820	0.4181	0.7767	0.6682
noun + verb + adj	0.8534	0.7485	0.7509	0.5939	0.4148	0.7770	0.6685

Table 4: Recall with restricted morphemes

morphemes	Grain	Vegetable	Fish and Meat	Milk	Fruits	Sweets	Others
verb	0.6316	0.7719	0.4966	0.0030	0.0000	0.6364	0.0950
noun	0.8631	0.8560	0.7466	0.1349	0.0437	0.8731	0.2370
noun + verb	0.8690	0.8586	0.7550	0.1524	0.0533	0.8744	0.2583
noun + verb + adj	0.8688	0.8596	0.7557	0.1533	0.0567	0.8743	0.2595

Table 5: F-value with restricted morphemes

morphemes	Grain	Vegetable	Fish and Meat	Milk	Fruits	Sweets	Others
verb	0.6304	0.6477	0.5341	0.0059	0.0000	0.6416	0.1409
noun	0.8555	0.7928	0.7452	0.2203	0.0787	0.8187	0.3502
noun + verb	0.8610	0.7992	0.7527	0.2406	0.0940	0.8227	0.3723
noun + verb + adj	0.8610	0.8002	0.7532	0.2428	0.0995	0.8228	0.3737

is Grain and every test data contains even numbers of Grain recipes, then "positive labeled recipes" for one test is 18287 (= 91436 / 5).

From these results, we find 0.56 F-value average when input vectors are made from noun, verb and adjective. The performances of Milk and Fruits are low because of few examples.

Next, we restrict text areas to construct input vectors. The following restrictions are evaluated.

- ingredients
- (cooking) process
- ingredients + (recipe) title
- process + title
- ingredients + process
- ingredients + process + title

Table 6 shows the accuracy of every condition. Obviously, text conditions which contain the recipe title get high performance than the others. From this, the recipe title name is useful for automatic classification even though upload users can select words in the title freely.

Table 7, 8 and 9 show precision, recall and F-value of each SVMs, respectively.

From these results, recipe title is also important to decide classes. Almost all conditions, if they contains the recipe title, the F-value is higher than that in the condition without the recipe title. Only on the condition that "ingredients + process + title" for Fruits, the F-value is smaller than that on "ingredients + process" for Fruits. This is also because of lack of learning examples.

5. Conclusions

We have shown a classification method for a recipe to a class on Japanese Food Guide Spinning Top. This is done by SVMs with one-versus-rest. The accuracy is about 0.7 to 0.8 and the title of the recipe is important to classify. For the future work, inferring SV value for a recipe is remaining. We can make the automatic food guide system if we can combining them.

Table 6: Accuracy of text restriction

text	input vector size	accuracy
ingredients	29377	0.7502
process	25410	0.7726
ingredients + process	53332	0.7885
process + title	34809	0.8119
ingredients + title	44251	0.8217
ingredients + process + title	61079	0.8090

Table 7: Precision with text restriction

text	Grain	Vegetable	Fish and Meat	Milk	Fruits	Sweets	Others
ingredients	0.8590	0.7184	0.7213	0.4624	0.3488	0.7504	0.5911
process	0.8534	0.7485	0.7509	0.5939	0.4184	0.7770	0.6685
ingredients + title	0.8829	0.8178	0.7981	0.6330	0.4745	0.8179	0.7270
process + title	0.8726	0.8031	0.7915	0.6682	0.4607	0.8099	0.7194
ingredients + process	0.8667	0.7678	0.7641	0.6150	0.4415	0.7890	0.6936
ingredients + process + title	0.8723	0.8024	0.7842	0.6585	0.4356	0.8038	0.7175

Table 8: Recall with text restriction

text	Grain	Vegetable	Fish and Meat	Milk	Fruits	Sweets	Others
ingredients	0.8426	0.8552	0.7371	0.0974	0.0508	0.8777	0.1424
process	0.8688	0.8596	0.7557	0.1533	0.0567	0.8743	0.2595
ingredients + title	0.9159	0.8718	0.8083	0.2743	0.1116	0.8983	0.4553
process + title	0.9087	0.8677	0.7903	0.2493	0.0595	0.8992	0.4317
ingredients + process	0.8901	0.8651	0.7756	0.1866	0.0554	0.8925	0.2843
ingredients + process + title	0.9032	0.8658	0.7896	0.2131	0.0495	0.8978	0.4271

Table 9: F-value with text restriction

text	Grain	Vegetable	Fish and Meat	Milk	Fruits	Sweets	Others
ingredients	0.8507	0.7808	0.7290	0.1601	0.0884	0.8091	0.2292
process	0.8610	0.8002	0.7532	0.2428	0.0995	0.8228	0.3737
ingredients + title	0.8991	0.8439	0.8031	0.3809	0.1798	0.8562	0.5597
process + title	0.8902	0.8341	0.7909	0.3624	0.1048	0.8522	0.5392
ingredients + process	0.8782	0.8135	0.7697	0.2848	0.0983	0.8376	0.4030
ingredients + process + title	0.8875	0.8329	0.7868	0.3206	0.0880	0.8482	0.5352

References

[1] N. Yoshiike, F. Hyashi, Y. Takemi, K. Mizoguchi and F. Seino, A new food guide in Japan : the Japanese Food Guide Spinning Top, Nutr. Rev. vol.65, no.4, pp.149–154, 2007.

[2] R. Takahashi, S. Oyama, H. Ohshima and K. Tanaka, Measuring relevancy of modifiers in the recipe names on the user-generated recipe websites, IEICE Trans. (in Japanese), vol.J94-A, no.7, pp.467–475, 2011.

[3] Y. Shidochi, I. Ide, T. Takahashi and H. Murase, Finding replaceable materials by cooking recipe mining, IEICE Trans. (in Japanese), vol.J94-A, no.7, pp.532–535, 2011.

[4] N. Cristianini and J. S. Taylor, An introduction to support vector machines and other kernel-based learning methods, Cambridge Univ. Press, 2000.

[5] The Rakuten Data Release provided by Rakuten, Inc.

[6] MeCab: Yet Another Part-of-Speech and Morphological Analyzer http://mecab.googlecode.com/svn/trunk/mecab/doc/index.html

Evaluating Elements of Communicative Stuffed-toy Device Describes Scripts on SNS

Haruka Mase[1], Tomoko Yonezawa[2], and Kazuki Joe[1]
[1]Dept. of Advanced Information and Computer Sciences, Nara Women's University, Nara, Japan
[2]Faculty of Informatics, Kansai University, Osaka, Japan

Abstract— *This paper investigates the elements of our proposed stuffed-toy device that is expected to work as an outlet target receptacle of a user's mental anguish. When the user physically interacts with the stuffed toy, a script of the stuffed toy is post on an SNS, such as Twitter. We aim for an indirect and unforced communication between the damaged user and other participants in SNS through the scripts of the stuffed toy device that indicates sympathetic understanding for user's mood recognition.*

Keywords: Stuffed-toy, Twitter, SNS, Rapport, Communication support

1. Introduction

There are many people under various strains in a stressful society today. In March 2011, we have received strong mental stress due to the Great East Japan Earthquake. It is considered that the method of solutions for those problems are counseling and healing programs. However, there are still additional problems and psychological burdens to have such programs; not only do they need time and money but it is also difficult to remove hesitation to have the programs especially for the first time. Moreover, it takes mental energy to find a suitable counselor for each person. To confirm such tendencies, we conducted a preliminary survey of the impression for counseling with questionnaire sheets. As results, it was confirmed that there were a lot of negative attitudes such as "I feel reluctance to get counseling," "I do not want to talk with strangers," and so on with only few responses with positive impressions.

In this paper, we propose a communicative stuffed-toy device that works as a emotional outlet for effusion of the user's psychological burdens. Our proposed device is expected to reduce the hesitation to understand the user's own mental situation by nonverbal effusion to artificial presence rather than by verbal explanation in a counseling session that requires self-discourse to another person. The proposed system has two types of feedbacks when the user touches on the stuffed-toy device, 1) the stuffed toy makes nonverbal motions by its head or arm and 2) it makes its own comment on an SNS such as Twitter[1]. The former interaction is expressed for the live-communicative healing of the user, and the latter interaction is a method for an indirect communication between the user and the family and friends of the user. The pull-type indirect representation without intrusion is possible to promote future proactive communications.

Additionally, in this paper, we conveyed the system evaluations focusing on the elements: surface materials, existence of the face, physical motions, and text messages. From the results of the experiment, we discuss whether the stuffed toy has the reduction effect of the user's stress and the effect as the receptacle of the user's emotional effusion.

[1]Twitter : https://twitter.com/

2. Related Works

Here, we discuss the related works on anthropomorphic agent-based counseling systems and communication systems.

ELIZA(DOCTOR)[1] is a text-based interactive counseling system with a computer in 1960s. ELIZA is a dialogue system that simulates a conversation between the psychoanalyst and a patient without a tangible body or virtual presence with any physical embodiment. When we consider verbal communication with an artificial presence, the conversation tends to become unnatural because of the mismatched keywords used to generate verbal replies from the system. ELIZA replies to the user's texts by repeating like a parrot without any dialogic analyses. Even such simple replies could make effective results.

To avoid the complex processes and the problems in the verbal communication and also to enrich the non-verbal and physical interactions, we adopted a stuffed animal as a medium covering the interface device. Stuffed toys do not make physical motions by themselves so that they have flexible characteristics as shown in their usages both as partners and as avatars of the user in playing house. Moreover, there are psychological therapies using stuffed toys as same as shown in animal therapy[2]. Consequently, we involved anthropomorphic physical interactions with nonverbal input from the user to take advantage of intuitive healing methods. The stuffed toy describes information of the user's interaction onto SNS such as Twitter to indirectly represent the user's close members of the family or friends how the user interacted with the stuffed toy.

On the other hand, the continuous interaction needs to establish a trustful relation, "rapport," between the user

Table 1: Experiences with stuffed toys

	Male (/123)	Female (/80)
Have spoken to	22.0%	63.8%
Have hugged	30.9%	77.5%

and the stuffed toy. In establishing rapport, the cooperative attitude, such as appropriate gaze as attentiveness and the positivite attitude are especially important[3]. In this paper, we adopt the nod gesture of the stuffed toy as a cooperative attitude in order to engage the more familiar rapport between the user and the stuffed toy.

Nakatani et al. introduced a stuffed-toy robot system that shows breathing expression like a living presence in order to produce a sense of affective feeling [4]. This study has focused on a presence like living beings. Contrary to their approach, we focus on tactile communication with rapport to provide the user a sense of peace of mind.

Nakagawa et al. developed Keepon [5] that is a communication robot which aims for drawing communicative behavior of children. Keepon aims at emotional expressions by nodding and tilting the head. The idea of the head gestures for the communication is similar to our device. We regarded the nod of the stuffed toy as a basic attitude of engaging rapport for promoting continuous interaction, and focused on the nonverbal contents of the communication for further indirect communication with other people.

Osumi et al. has proposed an automatic generation of a robot's motions from weblogs [6]. The purpose of this study has similar aspect to our challenge in supporting the indirect communication among people. However, from the viewpoint of the receptacle of emotional effusion, we consider the role of the stuffed toy as a communicative medium, and the robot is not expected to directly talk verbal content to the user nor to explicitly show the presence of the audience in the SNS media. From the viewpoint of both 1) the nonverbal interaction for therapeutic effusion and 2) the indirect communication between the user and other people, we designed two types of interactions corresponding to the user's input.

3. System Implementation

3.1 Preliminary survey with questionnaires

3.1.1 Purpose of the Preliminary Survey

We investigated various daily-uses of stuffed toys, such as the target of the user's affective emotions, feelings, and contexts of the interactions, to consider the possibility of affective communication using a stuffed toy. 203 people (123 males and 80 females) aged from 19 to 24 years old were the target of our survey.

3.1.2 Items of Questionnaires

We prepared a questionnaire survey focusing on tactile interaction and one-way utterance to the stuffed toy. The question items are as follows.

- Have you talked to stuffed toys?
- When do you talk to stuffed toys?
- Have you held on stuffed toys?
- What do you think the reason when you held the stuffed-toy / did not held it?

Table 2: Situations when the male respondents have talked to stuffed toys(Male)

positive situations	
happy	2
enjoyable	1
negatives ituations	
depressed	2
lonely	5
sad	3
anxiety	1
other situations	
spare time	6
emotional swing	2
use as cusion	1
need to move	1
sleeping	1
playing	4

Table 3: Situations when the male respondents have embraced stuffed toys(Male)

positive situations	
happy	1
negative situations	
depressed	4
tired	7
lonely	4
sad	1
anxiety	5
irritated	1
other situations	
sleeping	7
without meaning	3
need warming	1
need healing	3

Table 4: Situations when the male respondents have talked to stuffed toys(Female)

positive situations	
happy	7
enjoyable	7
negative situations	
tired	1
lonely	9
sad	10
painful	1
other situations	
sleeping	3
playing	9
greeting	1
dropped	4
without meaning	5

Table 5: Situations when the male respondents have embraced stuffed toys(Female)

positive situations	
happy	8
negative situations	
tired	1
lonely	17
sad	7
anxiety	3
irritated	1
other situations	
want to hug	6
sleeping	9
feel cute	8
playing	1
need healing	5

3.1.3 The Results of the Preliminary Survey

The results of the questionnaires survey are summarized in Table 1. As can be seen, the ratios of the people who have experiences of hugging or talking to the stuffed toys are different between the results for males and females. It is conjectured that the differences are caused by a stereotype image of girls in using stuffed toys and dolls in playing house, especially in the investigated targets in Japan.

Table 2 and Table 3 show the contexts of the male interviewees interacted with the stuffed toys. The verbal talks were to stuffed toys were made in various contexts, in contrast, the embraces or hugs were made while the male inteviewees were in negative contexts.

Table 4 and Table 5 show the contexts of the female interviewees interacted with the stuffed toys. The results

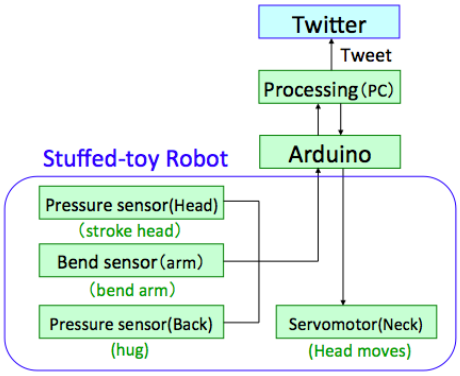


Fig. 1: Block diagram of the proposed system

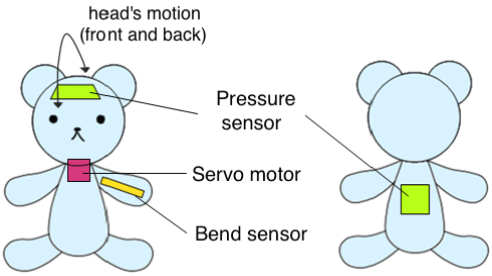


Fig. 2: Hardware structure of the system

were similar to the male interviewees with few tendencies of negative motivations.

In total, we could find the strong tendency of interaction under negative contexts. It is considered that people interact with, especially embrace, the stuffed toys when they have stresses from negative feelings or thoughts. From these results, the possibility of the stuffed toys is expected to become a role as an acceptant of the user's strong emotions.

3.2 System

3.2.1 System concept

The basic concept of our stuffed-toy system is built to give the device two aspects of communication roles; the first role is the acceptant of the user's strong emotions, and the second role is the medium as an indirect communicator with other people. An embodied stuffed toy enables physical and nonverbal interactions from the user without caring the other person. The recorded logs of the interaction can also indirectly inform simple situation of the user in SNS community so that her/his family and friends can roughly understand the user's state.

3.2.2 System summary

In this research, we adopted a bear-type stuffed toy as a physical embodiment of the device. Figure 1 shows

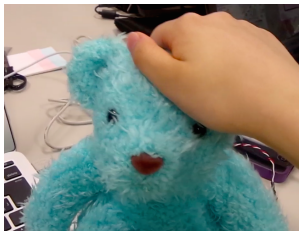


Fig. 3: Example use (1): (make-believe play by patting its head).

Fig. 4: Example use (2): make-believe play by bending its arm

Fig. 5: Example use (3): hugging it as a partner

the process flow of the proposed system. The stuffed toy includes three sensors and one servomotor in its body under the fur skin. The user's inputs such as stroking, embracing, or bending the arm of the stuffed toy are automatically recognized and the stuffed toy makes nods or other motions corresponding to the user's input. At the same time, a message related to the user's contexts and interactions is automatically sent to SNS server as a comment on Twitter.

3.2.3 Hardware Structure

Figure 2 shows the hardware structure of the system. We assumed three types of user's inputs:stroking or patting on the head of the stuffed toy (Figure 3), bending the arm of the stuffed toy to make some gestures (Figure 4), and embracing the stuffed toy (Figure 5). Figures from 6 show sensor values in the stuffed toy. In order to establish rapport between the user and the stuffed-toy device, we designed the motion of the device's head to move back and forth (nods) by a servo motor in the neck of the stuffed toy. This motion makes the user feel as though the stuffed toy nodded corresponding to his/her inputs.

3.2.4 Software Structures

The physical inputs from the user to the stuffed toy are detected from On/Off signals of the sensors by each threshold value although there should be various patterns and levels of delicate inputs. In order to produce reasonable and understandable reaction of the stuffed toy, we adopted a simple judgement of the inputs using threshold values. In the case when at least one of the sensors detect a value over

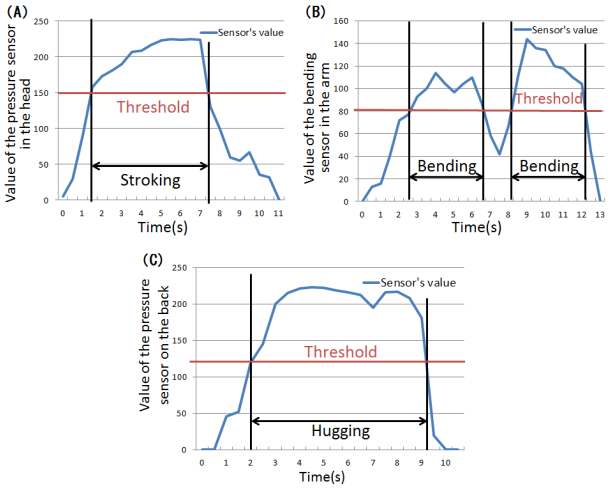


Fig. 6: Example values of the sensors

Table 6: The scripts of the stuffed toy in Twitter

context	actions		
	hug	pat on the head	bend the arm
morning	Good morning, thanks for hug. Should be surely a good day today!	Good morning, I'll cheer for you today.	Thanks for playing with me! Speak to me anytime.
noon	Thanks for hug. What's up? Please talk anything to me.	Thank you for patting me. It is necessary to have a break sometimes.	Thanks for playing with me! How are you today?
evening	Thanks for hug. I'm going to be along with you tomorrow too.	Thanks for your patting. What's up today?	Thanks for playing a lot with me! Let's play again tomorrow.
stronger input than usual	What's up? I hear anything, so, please try to talk anything.	What's up? I'll listen to you anytime, so please talk more and more.	What's up? I can play with you anytime, so don't hesitate!

the threshold, the AVR (Arduino[2]) sends the values of the sensors to PC (Processing[3]) by serial communication, and Processing selects an appropriate script for the comment in SNS based on the assumed context of the interaction from the scripts database (Figure 7). Based on the results of the preliminary surveys, we focused on the stuffed-toy's scripts to express concern of it for the user. The script database have twelve patterns of scripts for the stuffed toy on Twitter corresponding to the types of interaction, strength, and timing as shown in Table 6.

3.2.5 User Feedbacks in Demonstration Exhibition

Some participants in Maker Faire Tokyo 2013[4] experienced our proposed stuffed-toy device with the same functions, while the contents of the comments were different from Table 6. Some participants told us familiar feedbacks such as "The proposed stuffed-toy device is more popular than a normal stuffed animal." On the other hand, there

[2]Arduino : http://www.arduino.cc/
[3]Processing : https://www.processing.org/
[4]Maker Faire Tokyo 2013: http://makezine.jp/event/mft2013/

Fig. 7: Automatic Tweets of the stuffed toy

were several people who indicated negative opinions such as "It is scary." or "I have some hesitation of use because the interactions are uploaded automatically."

4. Evaluations of Stuffed-toy Device

Focusing on the direct interaction between the stuffed toy and the user, we evaluated the possibility of the proposed system for the user's emotional effusion.

4.1 Setting for experiment description

Purpose: We evaluated the following four verifications:

- Whether the stuffed-toy device can decrease the user's stress.
- Whether the user regards the stuffed-toy device as a communication partner.
- Whether the stuffed-toy has a role as an accepter of feelings.
- Whether it is possible to form the rapport.

Participants: Twenty-five people participated in this experiment. These participants are undergraduate or graduate students aged from 19 to 24 years old with sixteen male participants and nine female participants.

Factors in the experiment: We set the following factors based on the hypothesis above.

A The surface material of the device
B The arrangement of facial parts of the device
C The gestural motion of the device
D Text message

Level :

A Cloth / styrene board
B Face-like arrangement of the eyes and the mouth in a surface as a face. / Abnormal arrangement of the facial parts. (See Figure 9)
C With nods / without nod
D Existence / non-existence of a text message

Conditions: A branching table of the conditions is shown in Table 7. Two standards were set up to 4 factors of everything and the experiment was prepared by total of 16 conditions.

Hypotheses:

A When the surface material of the stuffed-toy device is cloth, the user feel relaxed and relieved with familiar emotion toward the stuffed toy.

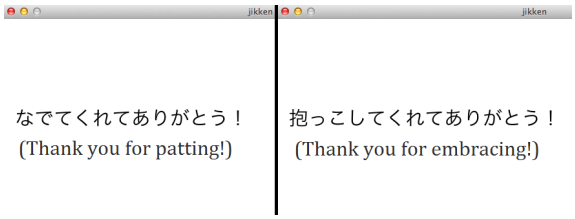


Fig. 8: Screen of the character message(two types)

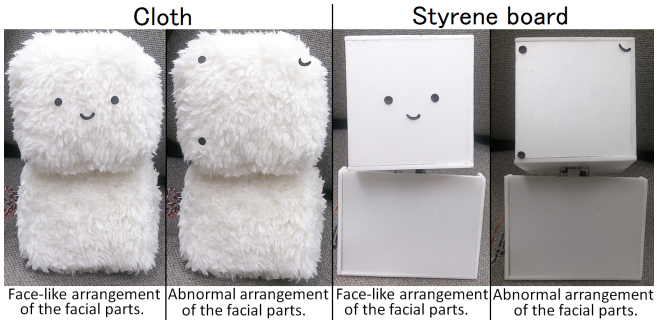


Fig. 9: The device in the experiment

B A robot with a face enables to make eye contact and to express emotions on its face. From the expressiveness of the face, the user become possible to can trust the robot (the engagement of rapport).

C When the stuffed-toy device nods, the user regards the robot as a trustworthy and communicative partner. In addition, the user feels easier to effuse her/his own emotion rather than the device without nod.

D When the user receives a text message from the robot, the user interprets it as that the stuffed-toy device is trying to communicate with the user.

Procedures: This experiment was prepared as a within-subjects design. Before the experiment, each participant had a POMS-SF test in order to measure the state of her/his stress. The participants were instructed to pay attention not only to the device but also the monitor, which shows a text message on the screen in the condition with the message.

In advance of the experiment, an experimenter directed the participant to pat on the robot's head and to embrace it during the experiment session. The participant can touch the device as she/he likes after the directed inputs. The participant's hand motions were recorded with a web-camera. After each session with a touch and embrace of the device the participant evaluated the evaluation items. The experiment sessions were randomized for counter-balance.

The robot for the experiment use: Robots used by this experiment is indicated on Figure 9.

Evaluation Items 1: subjective evaluations:

1) You felt at ease.
2) The robot seemed to try to tell you something.
3) You became an unpleasant feeling.
4) You were relieved.
5) The robot reacted to your action.
6) The robot seemed to try to communicate with you.
7) You could communicate with the robot easily.
8) The robot seemed to mind nothing.
9) You felt familiar feeling to the robot.
10) You felt that the robot was reliable.
11) The communication with the robot was natural.
12) The robot seemed to have an intention.
13) You felt like talking to the robot.
14) You touched the robot strongly.

Table 7: Conditional branch table

			message ○	message ×
Cloth	normal face	nod ○	Condition1	Condition2
		nod ×	Condition3	Condition4
	abnormal face	nod ○	Condition5	Condition6
		nod ×	Condition7	Condition8
Styrene board	normal face	nod ○	Condition9	Condition10
		nod ×	Condition11	Condition12
	abnormal face	nod ○	Condition13	Condition14
		nod ×	Condition15	Condition16

Table 8: The POMS-SF scores of the participants

The degree of stress	Value	The number of people
Low	23〜48	17
Middle	49〜73	5
High	74〜98	3

Evaluation Items 2: POMS-SF test POMS-SF test is a reduction edition of POMS test (the degree of stresses measurement check). As the value of the result becomes big, the degree of stress becomes high. POMS-SF test was performed to all participants first by this experiment. As a result, the lowest numerical value is 23 and the maximum value is 98 of participants. Table 8 shows the one which classified the numerical value into three classes according to the degree of the stress and totaled one in this area.

4.2 Result

Analyses of variance (ANOVA) with repeated measurement among the conditions(*p<.05) are shown in Table 9. The results of the means opinion scores (MOS) are shown in Figure 10 and 11.

The evaluation items 1, 4, 9 and 13 showed significances by all factors. Moreover, from the result of the evaluation item 3, the significant difference was found by the factor of material, face, and text message. Accordingly, It is suggested that the hypothesis A was confirmed.

From the results for the evaluation item 2, the significant difference was found by the factors of nod and text message in particular with the significant interaction between them. It is suggested that the robot's intention of the communication could be expressed by at least one modality of the expression. Moreover, the significant difference in the evaluation item 5 was confirmed by the factors of face, nod,

and text message with an interaction between the nod factor and the message factor. The item 8 showed significances by all factors with an interaction between the nod factor and the character factor. Additionally, the evaluation item 12 showed significant difference by the factors of face, nod, and message with the interaction between the nod factor and the message factor. From these results, the hypothesis B was confirmed.

Moreover, the results for the evaluation items 6, 7, 10 and 13 showed significant differences by the all factors so that the hypotheses C and D were confirmed.

The evaluation item 14 did not show any significance.

5. Discussion

Now, we discuss the effectiveness of the stuffed-toy device and two types of communication possibilities. From the results of four factorial analyses of variance, it was revealed that the appearance of the stuffed toy device and the interactions with the user are important from the evaluations for “whether the user felt at ease,” “whether relationship of mutual trust can be built between the user and the stuffed toy device(formation of rapport),” and “whether the stuffed toy device can be a communication partner" as the ANOVA results.

From the results for the evaluation item 1, 3, 4, 9 and 13, the users could become relieved and familiar to the stuffed toy by the soft material of the robot’s covering cloth. Additionally, it was revealed that the users tend to become easy to make emotional effusion by the soft material. However, the interactions between the face and nod factors and between the face and message factors were confirmed by all items mentioned above; therefore, it is suggested that some motion of the stuffed toy are also important as well as having a normal face.

The result for the evaluation items 6, 7, 10 and 13 showed that the facial appearance of the stuffed toy enabled eye contact with the user and the nodding of the stuffed toy represented the reliable and communicable partner. Additionally, from the result of the evaluation item 6, it is conjectured that a rapport was established between the user and the robot by adopting the material and engaging gestures of the stuffed toy. It is also considered that the representation of the text message enriched the user’s feeling of the robot’s effort to communicate with the user. Moreover, many interactions were confirmed by the factors about the appearance. For these reasons, the appearances of the stuffed toy could elevate the possibility of the user’s relieved and reliable communication. The importance of the nod and message factors was confirmed by both the interactions between the nod factor and the other factor in the evaluation item 8 and the interactions between the message factor and the other factor in the evaluation item 12.

From the result of the evaluation item 3, it is suggested that the user’s unpleasant feeling would be caused by uncomfortable touch or the inappropriate appearance of the anthropomorphic presence. The appearances of the stuffed toy without any message are considered to draw the negative feeling of the user as though the reaction of the toy were not really showing earnest attitude.

From the results for the evaluation item 2, 5, 8 and 12, it is suggested that the text message with concrete contents make the user interpret as though the artificial presence had its own intentions. Moreover, the results of evaluation item 8 and 12 showed the importance of the material of the device for expressing its intentions. Thus, the animal-like or living-being-like appearances could affect the user’s interpretation of the anthropomorphic presence.

Figure 10 and 11 showed that the condition 16 was the highest score in the evaluation item 3. Additionally, The conditions without nods nor messages were highly scored in the evaluation item 8. In contrast, the condition 1 was the highest score in all evaluation items except for the items 3, 8, and 14.

From these results, we could confirm the effectiveness of our proposed device in interpretation of anthropomorphic communication. We should develop further evaluations of not only the direct communication but also the indirect communication with the other people through the SNS text message of the stuffed toy.

6. Conclusion

In this paper, we proposed a communicative stuffed-toy system as an acceptant of the emotional effusion of a user’s psychological burdens. The proposed system consists of a stuffed-toy device, tactile sensors, and servo motors as actuators with network connection through a PC. The device captures, the physical and nonverbal inputs from the user not only for communicative interaction between the stuffed toy and the user but also for the indirect communication through posted messages between the user and the other people such as her/his remote family.

The experiment on the effectiveness of the proposed stuffed-toy device revealed that the user’s stress is able to be reduced by using the stuffed-toy. Moreover, the relationship between the user and the stuffed-toy device has been grown by its nods and indicated messages in SNS such as Twitter. Finally the our aiming rapport, that is the mutual trust, was established at least in the feeling of the user. Thereby, we found out that the user regards the stuffed-toy toy device as a communication partner in emotional interaction that needs mutual trust.

In future work, we should consider to perform the user’s gestural recognition which shows emotional feelings. The system also needs to understand the environment surrounding user by using such as sound recognition. Moreover, further evaluations for stress reduction should be conducted in future.

Table 9: The results of four-factor ANOVAs

	material(A)		face(B)		nod(C)		message(D)		
	F(24)	p	F(24)	p	F(24)	p	F(24)	p	interaction
Q1	85.888	< 0.01*	35.266	< 0.01*	53.275	< 0.01*	51.826	< 0.01*	BC*
Q2	1.854	0.19	6.060	0.02*	265.962	< 0.01*	89.363	< 0.01*	CD*
Q3	35.552	< 0.01*	16.358	0.01*	2.047	0.17	9.152	0.01*	BD*
Q4	71.14	< 0.01*	24.24	0.01*	33.62	< 0.01*	38.44	< 0.01*	BC*
Q5	2.489	0.13	8.197	0.01*	318.500	< 0.01*	151.034	< 0.01*	CD*
Q6	12.04	0.01*	20.54	0.01*	220.94	< 0.01*	98.08	< 0.01*	CD*, ABC*, ACD*
Q7	37.883	< 0.01*	30.438	< 0.01*	147.975	< 0.01*	75.013	< 0.01*	AC*, BC*
Q8	6.083	0.02*	19.989	0.01*	79.976	< 0.01*	18.416	0.01*	AC*, BC*, CD*
Q9	62.49	< 0.01*	27.48	< 0.01*	71.44	< 0.01*	66.46	< 0.01*	BC*, BD*
Q10	27.12	< 0.01*	49.31	< 0.01*	74.33	< 0.01*	24.00	0.01*	BD*
Q11	14.852	0.01*	29.538	< 0.01*	108.859	< 0.01*	49.596	< 0.01*	
Q12	2.94	0.1+	19.89	0.01*	186.90	< 0.01*	46.26	< 0.01*	AD*, BD*, CD*, ABC*
Q13	16.97	0.01*	32.47	< 0.01*	40.84	< 0.01*	16.07	0.01*	BD*
Q14	0.518	0.48	0.170	0.68	0.376	0.55	3.827	0.06+	

+ p <.10, * p <.05

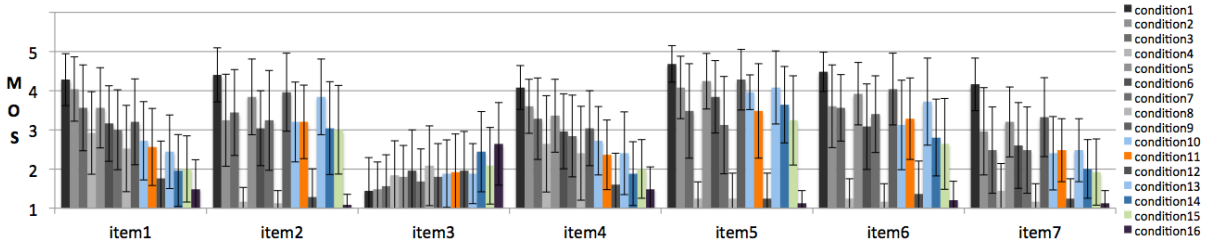


Fig. 10: ANOVA result graph - 1

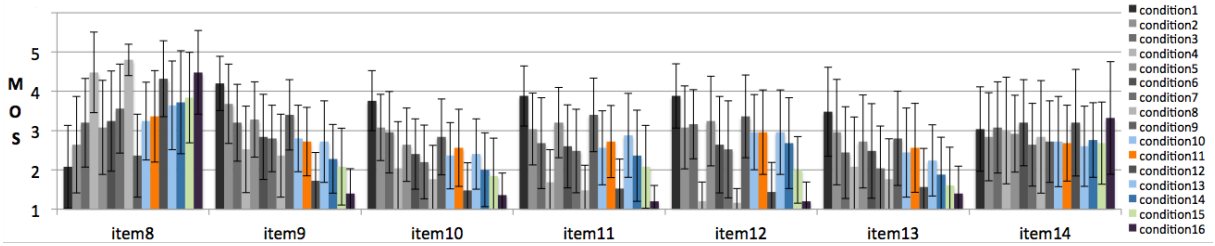


Fig. 11: ANOVA result graph - 2

Acknowledgment

This research was supported in part by JSPS KAKENHI 15H01698, JSPS KAKENHI 25700021, and JSPS KAKENHI 24300047. The authors would like to thank the participants in the experiment.

References

[1] J.Weizenbaum: "ELIZA-A Computer Program For the Study of Natural Language Communication Between Man and Machine," Communications of the ACM, Vol.9, No.1, pp.36–45, 1966.

[2] Tomoko Yonezawa, Hirotake Yamazoe, Akira Utsumi, Shinji Abe: "Attractive, Informative, and Communicative Robot System on Guide Plate as an Attendant with Awareness of User's Gaze," Paladyn. Journal of Behavioral Robotics, Vol.4, Issue 2, pp.113–122, 2013.

[3] Tickle-Degnen, L., Rosenthal, R.: "The Nature of Rapport and Nonverbal Correlates," Psychological Inquiry, Vol.1, No.4, pp.285–293, 1990.

[4] Yukari Nakatani, Tomoko Yonezawa: Breatter: "A Simulation of Living Presence with Breath that Corresponds to Utterances," HRI2014, pp.254–255, 2014.

[5] Hideki Kozima, Marek Piotr Michalowski, Cocoro Nakagawa: "Keepon: A playful robot for research, therapy, and entertainment," International Journal of Social Robotics, Vol.1, No.1, pp.3–18, 2009.

[6] Toshihiro Osumi, Kenta Fujimoto, Yuki Kuwayama, Masato Noda, Hirotaka Osawa, Michita Imai, Kazuhiko Shinozawa: "BlogRobot: Mobile Terminal for Blog Browse Using Physical Representation," International Conference on Social Robotics, pp.96–101, 2009.

[7] Tomoko Yonezawa, Brian Clarkson, Michiaki Yasumura, Kenji Mase, "Context-aware sensor-doll as a music expression device," CHI'01 Extended Abstracts on Human Factors in Computing Systems, pp. 307–308, 2001.

A study on Non-Correspondence in Spread between Objective Space and Design Variable Space and Application to Genetic Search

Tomohiro Yoshikawa, Toru Yoshida, Toshihiro Kishigami
Dept. of Computational Science and Engineering
Nagoya University
Furo-cho, Chikusa-ku, Nagoya 466-8603, Japan

Abstract— *Recently, a lot of studies on Multi-Objective Genetic Algorithm (MOGA), in which Genetic Algorithm is applied to Multi-objective Optimization Problems (MOPs), have been reported actively. MOGA has been also applied to engineering design fields, then it is important not only to obtain high-performance Pareto solutions but also to analyze the obtained Pareto solutions and extract some knowledge in the problem. In order to analyze Pareto solutions obtained by MOGA, it is required to consider both the objective space and the design variable space. In this paper, we define "Non-Correspondence in Spread" between the objective space and the design variable space. We also try to extract the Non-Correspondence area in Spread with the index defined in this paper. Moreover, we apply the defined index to genetic search to obtain Pareto solutions that have different design variables one another having similar fitness values. This paper applies the above index to the trajectory designing optimization problem and extracts Non-Correspondence area in Spread int the acquired Pareto solutions. This paper also shows that robust Pareto solutions can be acquired by the genetic search with the index.*

Keywords: Non-Correspondence, Objective Space, Design Variable Space, Distributed Area, Multi-objective Optimization Problem

1. Introduction

Genetic Algorithm (GA) is expected to be effective for solving Multi-objective Optimization Problems (MOPs), which maximizes or minimizes multiple objective functions at the same time. Recently, Multi-Objective Genetic Algorithm (MOGA), applying GA to MOPs, are getting much attention and a lot of studies have been reported[1]. Generally, it is difficult to obtain the optimized solution satisfying all objective functions because of their trade-offs. Then, it is necessary to obtain Pareto solutions which are not inferior to other solutions in at least one objective function.

In recent years, it is reported that MOGA is applied to engineering design problems in the real-world due to the improvement of computing performance[2][3][4]. In the engineering design problems, it is required not only to obtain high performance Pareto solutions using MOGA but also to analyze and extract design knowledge in the problem. And in order to analyze Pareto solutions obtained by MOGA, it is required to consider both the objective space and the design variable space.

Obayashi obtained Pareto solutions for aircraft configuration problem by MOGA and tried to analyze the obtained Pareto solutions through the visualization of the relationship between fitness values and design variables using Self Organizing Map (SOM)[2]. Kudo *et al*. proposed a visualization method that visualized the geometric distance between data in the design variable space based on their relationship in the objective space, and they analyzed the relationship between the fitness values and the design variables in the conceptual design optimization problem of hybrid rocket engine[5].

In this paper, we analyze obtained Pareto solutions considering the objective space and the design variable space, and we especially focus on "Non-Correspondence" between two spaces. In this study, we have introduced 3 patterns of Non-Correspondence between the objective space and the design variable space.

- Non-Correspondence in Sequence
- Non-Correspondence in Spread
- Non-Correspondence in Linear Relationship

We have already reported on the Non-Correspondence in Sequence[6]. In this paper, we define "Non-Correspondence in Spread" and propose the quantitative index to extract Non-Correspondence area in Spread. Non-Correspondence area in Spread is the area where solutions are distributed densely in the objective space but are distributed widely in the design variables space, and vice versa.

This paper applies the proposed method to the trajectory designing optimization problem known as DESTINY (Demonstration and Experiment of Space Technology for INterplanetary voYage)[7] provided by Japan Aerospace Exploration Agency (JAXA). We apply NSGA-II (Non-dominated Sorting Genetic Algorithm-II)[8] to this problem. We also analyze the extracted Non-Correspondence area in Spread in the obtained Pareto solutions. We also apply the defined index to genetic search to obtain Pareto solutions that have different design variables one another having similar fitness values.

2. Non-Correspondence in Spread

2.1 Definition of Non-Correspondence in Spread

In this paper, we focus on Non-Correspondence in Spread. The area with Non-Correspondence in Spread, called Non-Correspondence area in Spread, is defined as the area where solutions are distributed densely in the objective space but are distributed widely in the design variables space, and vice versa (Hereinafter we call simply "Non-Correspondence area."). The former means that there are a lot of design patterns with similar performance and the later means that the design variables are sensitive, i.e. the small change of design variables causes the large change of fitness values. For designers, the former is especially important because they can select design variables from some design patterns having similar performance in consideration of the cost of design or the difficulty level of design and the later is because they have to choose design variables very carefully. Figure 1 shows an example of Non-Correspondence area. In Fig. 1, data 5-6-7-8 are distributed widely in the design variable space compared to the distribution of the objective space.

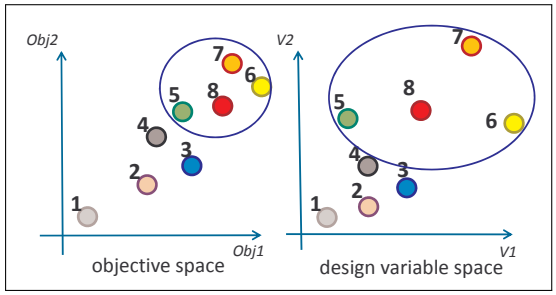


Fig. 1: Non-Correspondence area in Spread

2.2 Index for Non-Correspondence Area in Spread

Here, we define the quantitative index for Non-Correspondence in Spread to extract the Non-Correspondence area. The index is calculated in the following procedure.

1) Define the neighborhood radius ϵ (eq. (1)) in the objective space or the design variable space.
2) Extract the individuals as target individuals within radius ϵ from individual i.
3) Calculate the center of gravity of the target individuals.
4) Calculate the index for Non-Correspondence in Spread v_i according to eq. (2).

By the above procedure, the index v_i is calculated for each individual. The neighborhood radius ϵ is defined by eq. (1). In eq. (1), η denotes the parameter that defines the neighborhood radius, f_{lmax}, f_{lmin} mean the maximum and the minimum fitness values, which are normalized, in the Pareto solutions for objective function l, and M_f is the number of objective functions. x_{lmax}, x_{lmin} mean the maximum range and the minimum range of design variables l, and M_d is the number of design variables. If the neighborhood is defined in the objective space, the upper equations in eq. (1) and eq. (2) are employed and otherwise the lower equations are employed to calculate the value of index v_i. In eq. (2), d_{dcik} is the normalized Euclidean Distance between target individual k and the center of gravity in the design variable space, d_{fcik} is that in the objective space, N is the number of the target individuals and v_i is the index for individual i. Individuals with large indexes are distributed densely in the objective space / design variable space and distributed widely in the design variable space / objective space.

$$\epsilon = \begin{cases} \frac{\sqrt{\sum_{l=1}^{M_f}(f_{lmax}-f_{lmin})^2}}{\eta} \\ \text{(Neighborhood was defined in the objective space.)} \\ \frac{\sqrt{\sum_{l=1}^{M_d}(x_{lmax}-x_{lmin})^2}}{\eta} \\ \text{(Neighborhood was defined in the design variable space.)} \end{cases} \quad (1)$$

$$v_i = \begin{cases} \frac{1}{N}\sum_{k=1}^{N}(d_{dcik}) \\ \text{(Neighborhood was defined in the objective space.)} \\ \frac{1}{N}\sum_{k=1}^{N}(d_{fcik}) \\ \text{(Neighborhood was defined in the design variable space.)} \end{cases} \quad (2)$$

2.3 Application to Genetic Search

By applying the index defined above to genetic search, it is aimed to obtain solutions whose fitness values are robust to the change of design variables. In this paper, we apply the index value instead of Crowding Distance that is one of the distinguish mechanism in NSGA-II. We introduce the new search criterion R according to eq. (4) . In eq. (4), CD_i is Crowding Distance of individual i and v_i is the index value of Non-Correspondence in Spread defined by eq. (2). As the aim of 2.2 is to extract the Non-Correspondence area, v_i is calculated using the distance between target individuals and the center of gravity of them. On the other hand, v_i is calculated according to eq. (3) using the normalized Euclidean Distance d_{dik} between the individual i and the target individual k because the index value is introduced into each individual as search criterion. In addition, this paper aims to obtain solutions with the robustness to the change of fitness values in regard to the change of design variables, then the neighborhood radius is defined in the objective space.

$$v_i = \frac{1}{N}\sum_{k=1}^{N}(d_{dik}) \quad (3)$$

$$R_i = CD_i * v_i \quad (4)$$

The larger the value of CD or v is, the larger the value of R is. Thus it is expected that this method can consider the diversity in the objective space and the robustness of fitness values.

3. Experiment

In this paper, we applied the above calculation to the trajectory designing optimization problem "DESTINY" provided by JAXA and analyzed the obtained Pareto solutions.

3.1 Trajectory Designing Optimization Problem

The aim of this problem is to reach the moon as early as possible with less fuel and to reduce the degradation of the solar array panel of the spacecraft due to the damage by the radiation of the Van Allen belt. As shown in Fig. 2, the spacecraft is launched by Epsilon Rocket and put elliptical orbits around the earth. Once being put in orbit, the spacecraft is released and accelerates with Ion Engine until it reaches the moon. The spacecraft firstly aims to gain the altitude of perigee and switches to gain the altitude of apogee on the way, then it gradually moves closer to the moon.

This paper tries to optimize the trajectory designing of the spacecraft until it reaches the moon ((1),(2) in Fig. 2). The objective functions, the design variables, and the range of each design variable in this problem are shown in TABLE 1, TABLE 2, and TABLE 3, respectively. $V6$ is used in the case of optimization for 6 objective functions. As shown in TABLE 1, this problem can be expanded to six objective optimization problem. This paper deals with 5 objective functions $Obj1$, $Obj2$, $Obj3$, $Obj4$, $Obj5$ in TABLE 1.

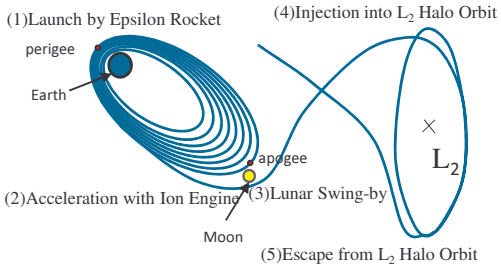


Fig. 2: Consept of DESTINY

3.2 Experimental Condition

NSGA-II was applied to the problem described above and 2000 Pareto solutions were obtained. We employed SBX[9] for the crossover and Polynomial Mutation[10]. Crossover rate was 1.0, mutation rate was 0.2, population size was 715, and generation was 100.

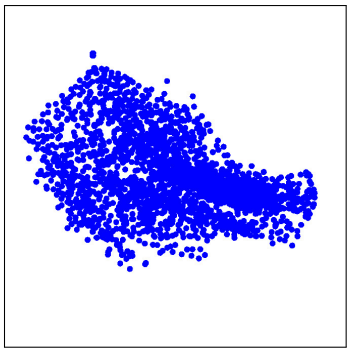
Figure 3 shows the visualization result of the distribution of obtained Pareto solutions in (a)the objective space and (b)the design variable space by Multi-Dimensional Scaling (MDS)[11].

Table 1: Objective Functions

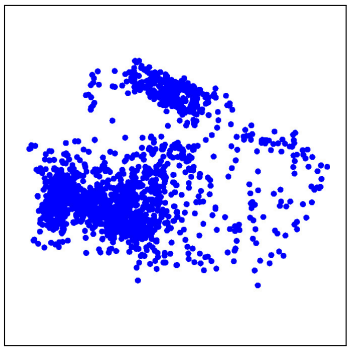
$Obj1$	time to reach altitude of 20000km	Min
$Obj2$	IES (Ion Engine System) operation time	Min
$Obj3$	the time to reach the Moon	Min
$Obj4$	the maximum eclipse time	Min
$Obj5$	the time to reach an altitude of 5000km	Min
$Obj6$	Initial mass of the spacecraft	Max

Table 2: Design variables

$V1$	: Launching date
$V2$	: Launching time
$V3$	: Switching apogee-perigee date
$V4$	: Range of IES operation time in perigee rise phase
$V5$	: Range of IES operation time in apogee rise phase
$V6$	: Initial mass of spacecraft



(a) objective space



(b) design variable space

Fig. 3: Distribution of Pareto Solutions

Table 3: Ranges of design variables

$V1$	2017/1/1-2017/12/31
$V2$	00:00:00-23:59:59
$V3$	90-365[days]
$V4$	0-180[degrees]
$V5$	0-180[degrees]
$V6$	350-450[kg]

3.3 Extraction of Non-Correspondence Area in Spread

The result of the index values for Non-Correspondence in Spread calculated by eq. (2) for obtained 2000 Pareto Solutions, in which the neighborhood was defined in the objective space, are shown in Fig. 4. Neighborhood radius ϵ was set as $\eta = 8$ in eq. (1). The parameter of neighborhood radius ϵ was not sensitive and the results were not much changed by the difference of ϵ in the experiments of this paper. The individuals in Fig. 4 are sorted in descending order of the index v_i. The vertical axis shows the value of the index v_i and the horizontal axis shows the individual label.

We focused on the top 50 individuals with large index values. Figure 5 shows the result of visualization of the distribution in which these 50 individuals are colored by red on the result of the objective space and the design variable space shown in Fig. 3. As shown in Fig. 5, the individuals with red color are distributed widely in the design variable space compared to the distribution in the objective space. We extracted 2 individuals in these 50 individuals and the fitness values and design variables of them are shown in TABLE 4.

In TABLE 4, each fitness value in the second and the third rows is normalized by the maximum and the minimum fitness values of the obtained Pareto solutions into the range of [0,1], and each design variable is normalized by the feasible ranges shown in TABLE 3 into [0,1]. In TABLE 4, though A and B have similar fitness values each other, the design variables are widely different. For example, the launching dates are March and December, the launching times are 1 in the midnight and 8 in the morning, and $V3$ and $V5$ are also different. In this area, there were some individuals that design variables are widely different with similar fitness values.

The result of the index values in eq. (2), in which the neighborhood was defined in the design variable space, are shown in Fig. 6. Neighborhood radius ϵ was set as $\eta = 8$ in eq. (1). Figure 6 shows the value of index v_i for each individuals same as Fig. 4.

Figure 7 shows the result of the visualization of the distribution of the top 50 individuals with large index values. As shown in Fig. 7, the individuals with red color are distributed widely in the objective space compared to the distribution in the design variable space. TABLE 5 shows the extracted 2 individuals C and D in Fig. 7 in the same way with TABLE 4. In TABLE 5, though C and D have similar design variables each other, the fitness values are widely different. In this area, there were some individuals that fitness values were very sensitive to the change of design variables. Thus it is required for the designer to choose or design very carefully a Pareto solution in this area.

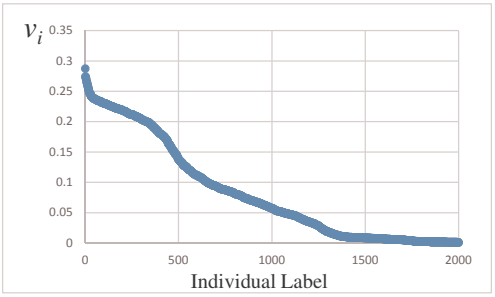
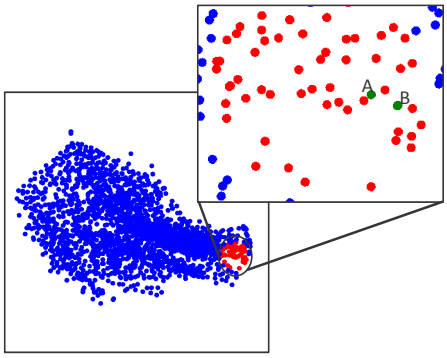
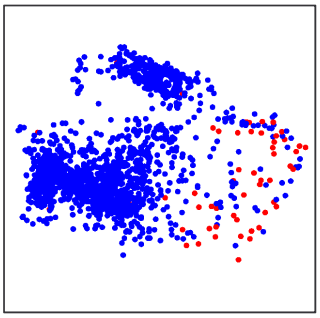


Fig. 4: Value of Index v_i in eq. (2) for each Individual (Neighborhood : Objective Space)



(a) objective space



(b) design variable space

Fig. 5: Distribution of Pareto Solutions for Non-Correspondence Area (Neighborhood : Objective Space)

3.4 Acquirement of Robust Pareto Solutions

In the experimental condition here, population size was 100, and generation was 100. We compared the proposed

Table 4: fitness values and design variables of selected individuals (A, B)

	Normalized Value		Actual Value	
	A	B	A	B
$Obj1$	0.006	0.011	1434.70	1437.75
$Obj2$	0.846	0.910	8545.60	8713.77
$Obj3$	0.035	0.0005	401.08	395.65
$Obj4$	0.097	0.167	1.524	2.009
$Obj5$	0.018	0.085	217.71	221.07
$V1$	0.201	0.916	2017/3/15	2017/12/1
$V2$	0.051	0.336	01:13:47	08:4:14
$V3$	0.977	0.313	358	175
$V4$	0.999	1.000	179.94	180.00
$V5$	0.818	1.000	147.99	180.00

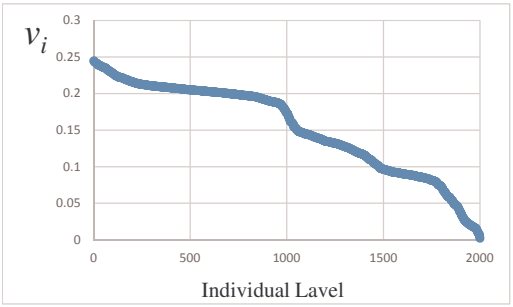
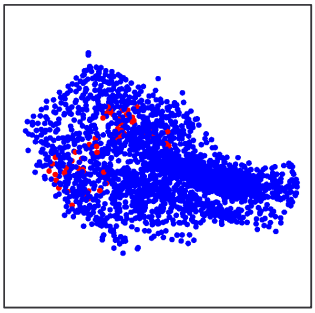


Fig. 6: Value of Index $v_i$in eq. (2) for each Individual (Neighborhood : Design Variable Space)

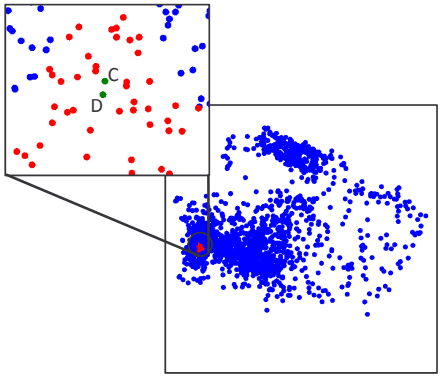
Table 5: fitness values and design variables of selected individuals (C, D)

	Normalized Value		Actual Value	
	C	D	C	D
$Obj1$	0.660	0.857	1801.06	1911.56
$Obj2$	0.226	0.061	6891.16	6452.99
$Obj3$	0.660	0.890	497.49	533.10
$Obj4$	0.156	0.694	1.938	5.689
$Obj5$	0.290	0.385	231.23	236.01
$V1$	0.750	0.750	2017/10/1	2017/10/1
$V2$	0.382	0.385	09:10:59	09:14:16
$V3$	0.038	0.038	100	100
$V4$	0.999	0.985	179.95	177.37
$V5$	0.864	0.841	155.53	151.43

method described in 2.3 and NSGA-II as the conventional method. Note that the difference between the proposed method and the conventional method was only the difference between Crowding Distance (CD_i) and R_i by eq. (4). Figure 8 shows the visualization result of the distribution of obtained Pareto solutions in the objective space by MDS. In Fig. 8, the color of solutions indicates the value of index v, in which red color means large value and green color means small value.



(a) objective space

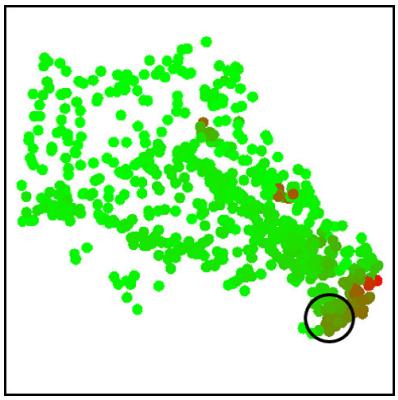


(b) design variable space

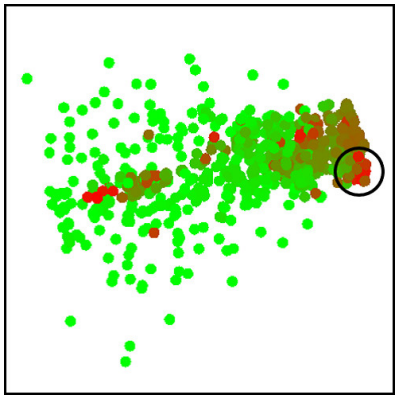
Fig. 7: Distribution of Pareto Solutions for Non-Correspondence Area (Neighborhood : Design Variable Space)

As shown in Fig. 8, Pareto solutions by the proposed method have many solutions with large value of v compared to the solutions by the conventional method. We extracted the solutions with large value of v (in the circle in Fig. 8(b)) from the obtained Pareto solutions by the proposed method. Next, we extracted the solutions having same fitness values in the conventional method and compared the values of design variables. The maximum and minimum fitness values in the extracted solutions are shown in TABLE 6.

We focused on Start Date and Time ($V1$ and $V2$), and showed the result of comparison in terms of Start Date and Time in Fig. 9. In Fig. 9, the horizontal axis shows the label of each solution and the vertical axis shows the value of each design variable. Red plots indicate Start Date and Time of solutions by the proposed method with the range of fitness values in TABLE 6 and blue plots indicate those of solutions by the conventional method. As shown in Fig. 9, the proposed method could obtain the solutions with Start Date and Time which the conventional method could not do (from December to February, from 23 o'clock to 2 o'clock).



(a) Result of conventional method



(b) Result of proposed method

Fig. 8: Distribution of Pareto Solutions (Objective Space)

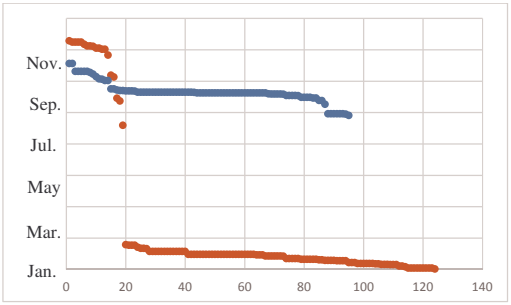
This result shows that the proposed method could obtain a lot of solutions having same fitness values and different Start Date and Time.

Table 6: Maximum and Minimum of Fitness Values

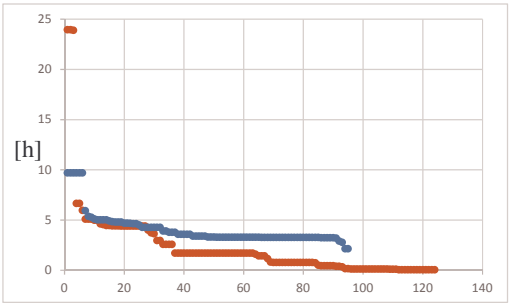
	Max	Min
$Obj1$	1544.51	1442.90
$Obj2$	8364.64	7999.78
$Obj3$	438.76	424.75
$Obj4$	1.893	1.144
$Obj5$	223.87	218.014

4. Conclusion

In this paper, we defined Non-Correspondence in Spread between the objective space and the design variable space. We proposed the quantitative index to extract Non-Correspondence area in Spread. This paper applied the proposed method to the trajectory designing optimization problem known as DESTINY provided by JAXA and analyzed the extracted Non-Correspondence area in Spread in the obtained Pareto solutions. This paper showed that the



(a) Start Date（$V1$）



(b) Start Time（$V2$）

Fig. 9: Start Date and Time

Pareto solutions having widely different design variables with similar fitness values could be extracted. This paper also applied the defined index to genetic search and obtained a lot of Pareto solutions having similar fitness values but different Start Date and Time. For the future work, we will apply the proposed method to other problems with more objective functions or higher dimensional design variables and study Non-Correspondence in Linear Relationship.

5. Acknowledgment

This research is partially supported by High Performance Computing Initiative Field 4: Industrial Innovation R and D Topic 4 [Design innovation with mutiobjective design exploration][12].

References

[1] K. Deb, *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.

[2] S. Obayashi, "Multiobjective design optimization of aircraft configuration (in japanese)," *The Japanese Society for Artificial Intelligence*, vol. 18, pp. 495–501, 2003.

[3] K. Deb, "Unveiling innovative design principles by means of multiple conflicting objectives," *Engineering Optimization*, vol. 35, no. 5, pp. 445–470, 2003.

[4] K. H. Akira Oyama, Yasuhiro Kawakatsu, "Application of multiobjective design exploration to trajectory design of the next-generation solar physics satellite," *Japanese Society for Evolutionary Computation*, 2010.

[5] F. Kudo and T. Yoshikawa, "Knowledge extraction in multi-objective optimization problem based on visualization of pareto solutions," *WCCI 2012 IEEE World Congress on Computational Intelligence*, pp. 860–865, 2012.
[6] T. Yoshida and T. Yoshikawa, "An extraction of non-correspondence area between objective space and design variable space based on order correlation of distance relation（in japanese）," *The Japanese Society for Artificial Intelligence*, 2013.
[7] A. L´ opez, A. Oyama, and K. Fujii, "Evaluating two evolutionary approaches to solve a many-objective space trajectory design problem," *The Japanese Society for Evolutionary Computation*, 2012.
[8] K. Deb, *A Fast and Elitist Multiobjective Genetic Algorithm : NSGA-II*. 2002.
[9] K. Deb and R. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 1, no. 9, pp. 115–148, 1994.
[10] K. Deb and M. Goyal, "A combined genetic adaptive search (geneas) for engineering design," *Computer Science and Informatics*, vol. 26, pp. 30–45, 1996.
[11] J. W. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Transactions on computers*, vol. 18, no. 5, pp. 401–409, 1969.
[12] http://www.ciss.iis.u-tokyo.ac.jp/supercomputer/about/.

Quantitative Evaluation of Reconstructed Image with Filtered Back Projection Bayes Method

Nodoka IIDA[1] Hayaru SHOUNO[1] Muneyuki SAKATA[2] Yuichi KIMURA[3]
[1] Graduate School of Informatics and Engineering, University of Electro-Communications,
Chofugaoka 1-5-1, Chofu, Tokyo, JAPAN
[2] Tokyo Metropolitan Institute of Gerontology,
35-2 Sakae-cho, Itabashi-ku, Tokyo, JAPAN
[3] Faculty of Biology-Oriented Science and Technology, Kinki University,
930, Nishimitani, Kinosawa, Wakayama, JAPAN

Abstract— *We compared a Bayesian reconstruction method with conventional filtered back projection (FBP) method in positron emission tomography (PET). In the medical scene, PET scanning plays an important role for functional diagnosis such like measuring the cerebral glucose metabolisms. Tomographic image is reconstructed from the observed data, which is the recieved signals from administrated-radioactive substances existing in the target tissue, by a PET scanner. Various reconstruction methods have been proposed, and we focus on a Bayesian approach of our previous method, that is, introducing a Gaussian Markov random field (GMRF) as a prior and Gaussian observations as likelihood function. We evaluate reconstruction performances of our method in a region of interest (ROI) based approach and compare with the one of FBP method. In the result, we obtain our Bayesian approach might be able to suppress fluctuation of the observation noise.*

Keywords: Positron emission tomography (PET), Filtered back projection (FBP), Bayes inference, Quantitative evaluation

1. Introduction

In the field of medical imaging, the positron emission tomography (PET) plays an important role for diagnosis. Observed data, or projection data can be obtained by a PET scanner which collects the distribution of radioactive substances, called a tracer, which emits positron, administrated in an target. Tomographic image is reconstructed from the projection data. Radon transform is usually used in mathematical formulation.

On the other hand, the quality of reconstructed image is always influenced with the ratio of signal and noise (S/N) in the observation. Generally, the larger S/N ratio brings, the better reconstructed images become. It is desired to obtain a high quality image as much as possible from observed data including much noise for reducing radiation exposure, or observed signals.

A lot of algorithms and methods for image reconstruction are proposed [1][2]. The filtered back projection (FBP) method is one of the most famous and standard reconstruction method as is introduced in PET scanner imaging systems.

On the contrary, image restoration method using Bayes inference has been discussed [3]. Thus, in our previous work, we extend the FBP method with introducing Bayes inference method[4][5]. Hereafter we call the method as FBP-Bayes method. Even though our method is expected in improvement in image quality compared with FBP method, there has not been any comparative experiments.

In the field of computer vision, PSNR(Peak Signal-to-Noise Ratio), which means the ratio between the power of a peak signal and the noise mixed in the whole image, or amount of degradation in image quality, is used for quantitative evaluation of restoration method. However, it is needed to know how much does the reconstructed image indicate physical quantity in the original space correctly in PET images. This is because the final goal of PET reconstructed image is to know how much does the radioactive substance in the region of interest (ROI) exist in the topical position of the target object. Therefore, PSNR itself does not play an important role.

In this paper, we compare images reconstructed by FBP method introduced in a commercial PET console and images reconstructed by proposed FBP-Bayes method, quantitatively.

2. Formulation

2.1 Radon Transform

Briefly, the Radon transform assumes that the observed signals are transmitted through the target object. Fig. 1 shows the schematic diagram of the Radon transform. We describe

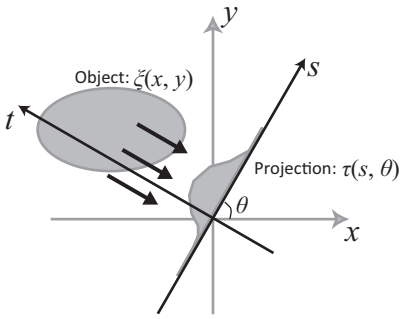


Fig. 1: Schematic diagram of the Radon transform. Detectors are aligned on the s axis, which has an angle described as θ.

the target object density as the function (x, y) coordinate, and assume that the detectors are aligned along the s axis that is rotated in θ degree. We can thus denote the relationship between the (x, y) and (s, t) coordinates as a rotation:

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{1}$$

We describe the density of the target as $\xi(x, y)$; that is, detectors are aligned on the s axis, so that we describe the observation $\tau(s, \theta)$, or a sinogram, as the following formulation, called Radon transform,

$$\tau(s,\theta) = \int dt\ \xi(x,y) = \int dt\ \xi\left(x(s,t),\ y(s,t)\right). \tag{2}$$

2.2 Conventional FBP Method

This reconstruction method is mainly formulated on the frequency domain, so we introduce the 2-dimensional Fourier transform of the reconstruction image $\sigma(x, y)$ and its inverse transform pair as

$$\tilde{\sigma}(\tilde{x},\tilde{y}) = \iint dxdy\ \sigma(x,y)\ e^{-2\pi \mathrm{j}(x\tilde{x}+y\tilde{y})} \tag{3}$$

$$\sigma(x,y) = \iint d\tilde{x}d\tilde{y}\ \tilde{\sigma}(\tilde{x},\tilde{y})\ e^{2\pi \mathrm{j}(x\tilde{x}+y\tilde{y})}, \tag{4}$$

where the $(\tilde{x}, \tilde{y})$ coordinate represents the space frequency.

Meanwhile, we can apply a 1-dimensional Fourier transform for the s of the observed data $\tau(s, \theta)$ as $\tilde{\tau}(\tilde{s}, \theta)$. The $\tilde{\tau}(\tilde{s}, \theta)$ satisfies the following relationship, which is called a projection theorem:

$$\tilde{\tau}(\tilde{s},\theta) = \int ds\ \tau(s,\theta) e^{-2\pi \mathrm{j} s\tilde{s}} \tag{5}$$

$$= \tilde{\xi}(\tilde{s}\cos\theta, \tilde{s}\sin\theta). \tag{6}$$

The FBP method is derived as a transformation of the from Cartesian coordinate $(\tilde{x}, \tilde{y})$ into the polar coordinate $(\tilde{s}, \theta)$ in the inverse Fourier transform (4):

$$\sigma(x,y) = \iint d\tilde{x}d\tilde{y}\ \tilde{\sigma}(\tilde{x},\tilde{y})\ e^{2\pi \mathrm{j}(x\tilde{x}+y\tilde{y})} \tag{7}$$

$$= \int_0^{\pi} d\theta \int_{-\infty}^{\infty} d\tilde{s}\ |\tilde{s}|\ \tilde{\sigma}(\tilde{s}\cos\theta, \tilde{s}\sin\theta)\ e^{2\pi \mathrm{j} s\tilde{s}}. \tag{8}$$

Omitting the observation noise, we can substitute eq.(6) into the part of eq. (8):

$$g(s,\theta) = \int d\tilde{s}\ |\tilde{s}|\ \tilde{\sigma}(\tilde{s}\cos\theta, \tilde{s}\sin\theta)\, e^{2\pi \mathrm{j} s\tilde{s}},$$

$$= \int d\tilde{s}\ |\tilde{s}|\ \tilde{\tau}(\tilde{s},\theta)\, e^{2\pi \mathrm{j} s\tilde{s}}. \tag{9}$$

Eq.(9) means the inverse Fourier transform of the projection $\tilde{\tau}(\tilde{s}, \theta)$ through the filter $|\tilde{s}|$. Thus, the reconstructed image is obtained by the integral of the filtered projection $g(s, \theta)$:

$$\sigma(x,y) = \int_0^{\pi} d\theta\ g(x\cos\theta + y\sin\theta, \theta), \tag{10}$$

where $s = x\cos\theta + y\sin\theta$ from eq.(1). We call this reconstruction method as the FBP method.

2.3 Stochastic Model

In this section, we introduce the observation noise as the stochastic model into the FBP method. Of course, we should consider Poisson noise for observation in a realistic model; however, solvable model is also important for understanding the reconstruction process.

Hence, in this study, we introduced additive Gaussian noise for observation on the signal $\xi(x, y)$. When we consider the Gaussian noise $n_p(x, y)$ on the image $\xi(x, y)$, the observation through the Radon transform $\tau(s, \theta)$ can be described as

$$\tau(s,\theta) = \int dt\,(\sigma(x,y) + n_p(x,y)) \tag{11}$$

$$= \int dt\sigma(x,y) + N_p(s,\theta), \tag{12}$$

where $N_p(s, \theta) = \int dt n_p(x, y)$. Therefore, we treat $N_p(s, \theta)$ as the Gaussian noise. In manner of the conventional image restoration method proposed by Tanaka & Inoue, we also introduce the energy function $H_{\mathrm{n}}(\boldsymbol{\tau}\,|\,\boldsymbol{\sigma})$ as following[7] [8]:

$$H_{\mathrm{n}}(\boldsymbol{\tau}\mid\boldsymbol{\sigma}) = 4\pi^2\gamma \int_0^{\pi} d\theta \int ds\ \left(\tau(s,\theta) - \int dt\,\sigma(x,y)\right)^2. \tag{13}$$

The hyper-parameter γ represents a precision parameter that is proportionate to the inverse of the variance of the Gaussian noise $N_p(s, \theta)$; that is, the large γ indicates a good S/N ratio in the observation. We can thus denote the observation process as

$$p(\boldsymbol{\tau}\mid\boldsymbol{\sigma}) \propto \exp(-H_{\mathrm{n}}(\boldsymbol{\tau}\mid\boldsymbol{\sigma})). \tag{14}$$

To reconstruct an image from noisy data, we also adopt the prior distribution in this study. We treat the energy function $H_{\mathrm{pri}}(\boldsymbol{\sigma})$ of the prior as

$$H_{\mathrm{pri}}(\boldsymbol{\sigma}) = \beta \iint dx\, dy\, ||\nabla\sigma(x,y)||^2 \tag{15}$$

$$+ 4\pi^2 h \iint dx\, dy\, \sigma(x,y)^2. \tag{16}$$

In Eq. (16), the first term corresponds to the Markov random field (MRF) like constraint; that is, neighboring pixel values should be similar to the target pixel, and the second term that corresponds to that pixel value should not take such a large value. The hyper-parameters β and h control the strength of each constraint. The prior probability can thus be described as the following when we adopt the polar coordinate in the frequency domain:

$$p(\boldsymbol{\sigma}) \propto \exp(-H_{\mathrm{pri}}(\boldsymbol{\sigma})). \tag{17}$$

From Equations (14) and (17), we can derive the posterior probability with Bayes theorem

$$p(\boldsymbol{\sigma} \mid \boldsymbol{\tau}) = \frac{p(\boldsymbol{\tau} \mid \boldsymbol{\sigma})p(\boldsymbol{\sigma})}{\sum_{\boldsymbol{\sigma}} p(\boldsymbol{\tau} \mid \boldsymbol{\sigma})p(\boldsymbol{\sigma})}. \tag{18}$$

2.4 Image Reconstruction method

We adopt the marginalized posterior mean $\langle\sigma(x,y)\rangle$ for the image reconstruction solution. The posterior mean can be denoted as

$$\langle\sigma(x,y)\rangle = \int_0^{\pi} d\theta \int_{-\infty}^{\infty} d\tilde{s}\, |\tilde{s}|\, \langle\tilde{\sigma}_{\tilde{s},\theta}\rangle\, e^{2\pi \mathrm{j}\tilde{s}(x\cos\theta + y\sin\theta)}. \tag{19}$$

Thus $\{\langle\tilde{\sigma}_{\tilde{s},\theta}\rangle\}$ represents an average set of Fourier expressions, is required to obtain the mean pixel value over the posterior $\langle\sigma(x,y)\rangle$. The average set can be denoted as

$$\langle\tilde{\sigma}_{\tilde{s},\theta}\rangle = \sum_{\boldsymbol{\sigma}} \tilde{\sigma}_{\tilde{s},\theta}\, p(\boldsymbol{\sigma} \mid \boldsymbol{\tau}). \tag{20}$$

This solution, called the posterior mean (PM), provides identical result as the MAP does, and it also provides the maximizer of the posterior marginals (MPM) solution of Eq. (18).

2.5 Hyper-parameter inference

To reconstruct an appropriate tomography image with our Bayesian inference, we need to assign proper values to the hyper-parameters β, h, and γ. The hyper-parameters β and h control the strength of constraints, while γ controls the fidelity of the observation. We infer these hyper-parameters by using maximization of marginal log-likelihood, which is sometimes called the type II maximum likelihood method.

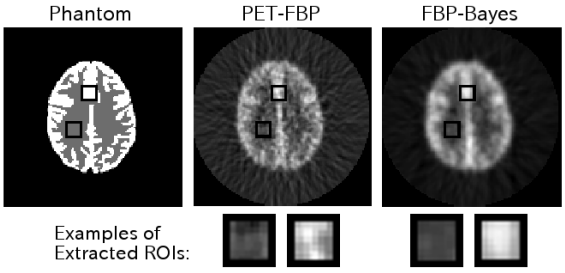


Fig. 2: Example of ROI extraction: ROIs are extracted from reconstructed images corresponding to that of the places in digital phantom. We focus on standard deviation and mean of each ROI; smaller change of the values by time means accuracy is better than the other.

The marginal log-likelihood denoted as the linear combination of log partition functions is

$$\ln p(\boldsymbol{\tau} \mid \beta, h, \gamma) = \ln Z_{\mathrm{post}}(\beta, h, \gamma) - \ln Z_{\mathrm{n}}(\gamma) - \ln Z_{\mathrm{pri}}(\beta, h), \tag{21}$$

where

$$Z_{\mathrm{pri}}(\beta, h) = \sum_{\boldsymbol{\sigma}} e^{-H_{\mathrm{pri}}(\boldsymbol{\sigma} \mid \beta, h)} \tag{22}$$

$$Z_{\mathrm{n}}(\gamma) = \sum_{\boldsymbol{\tau}} e^{-H_{\mathrm{n}}(\boldsymbol{\tau} \mid \boldsymbol{\sigma}, \gamma)} \tag{23}$$

$$Z_{\mathrm{post}}(\beta, h, \gamma) = \sum_{\boldsymbol{\sigma}} e^{-H_{\mathrm{pri}}(\boldsymbol{\sigma} \mid \beta, h) - H_{\mathrm{n}}(\boldsymbol{\tau} \mid \boldsymbol{\sigma}, \gamma)}. \tag{24}$$

The partition functions $Z_{\mathrm{pri}}(\beta, h)$, $Z_{\mathrm{n}}(\gamma)$, and $Z_{\mathrm{post}}(\beta, h, \gamma)$ correspond to the prior distribution, observation channel, and posterior distribution, respectively.

For maximization of marginal log-likelihood (21), we adopt a naive gradient method corresponding to the hyper-parameters β, h, and γ.

3. Quantitative Evaluation Method for Region of Interests

Quantitative evaluation of PET image, which means correspondence between pixel intensity and tracer density, in the original space is important for diagnose. For practical use for the diagnose of the PET image, we use following evaluation method. First, square shaped areas, whose distributions are equable, were extracted from digital phantom image as region of interest (ROI). In the ROI, from its homogeneity, the tracer density might be uniform distribution in ideal. Fig. 2 shows the ROIs in the each corresponding positions. The left shows the ROIs on the phantom image, which means the ideal reconstruction image. The middle one shows the conventional PET-FBP image with ROIs on the same locations in the Phantom. The right shows FBP-Bayes method

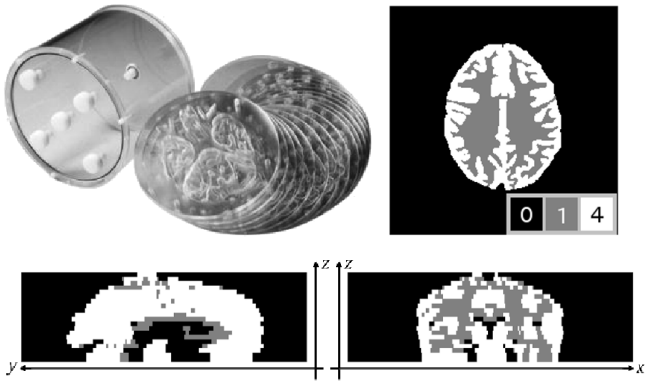


Fig. 3: Hoffman 3-D Brain phantom: It is a 3 dimensional model of the normal brain with 20 pieces. In the model, each of the gray matter part and the white matter part uptakes the tracer with the ratio of 4:1.

with ROIs in the manner of the middle one. The bottom part shows the magnified ROIs on corresponding images. Because the bandwidth gets narrower when using FBP-Bayes method compared to PET-FBP method (see Fig.5), mean of ROIs extracted from each method cannot be compared with each other. Therefore, we calculated the mean and the standard deviation (SD) of each ROI, and regarded the ratio of them, which means SD/mean, as the indicator of the image quality. If an image which has the smaller SD/mean, the image has the better quality. We applied 5 kinds of sizes of ROI: 4×4 [pixels], 6×6 [pixels], 8×8 [pixels], 10×10 [pixels], and 12×12 [pixels], covering all over the target domain (uptake ratio 1 and 4) of the images. In order to align the whole object location, which is called "co-register" procedure, we apply a software called Statistical Parametric Mapping (SPM; the Wellcome Trust Centre for Neuroimaging) between the digital phantom and each reconstructed image in the pre-processing.

We applied 5 kinds of sizes of ROI: 4×4 [pixels], 6×6 [pixels], 8×8 [pixels], 10×10 [pixels], and 12×12 [pixels], covering all over the target domain (uptake ratio 1 and 4) of the images.

4. Reconstruction and Evaluation

In the reconstruction experiment, we compared each of reconstructed images with corresponding slice in the Hoffman 3-D brain phantom. Fig.3 shows an example of the Hoffman 3-D Brain phantom. The Hoffman phantom consists of 20 slices, and each slice simulates the shape of a normal brain. Each slice consists of gray and white matter parts, and each part uptakes the tracer with ratio of 4:1. Thus, the pixel value in the cross-sectional image of Hoffman 3-D Brain phantom, which is called digital phantom,was 0, 1, and 4 ideally.

23.1 MBq of ^{11}C carfentanil were put in Hoffman phantom (see Fig.3) which was observed at Tokyo Metropolitan Institute of Gerontology (PET model:SET-2400W(Shimadzu,Tokyo,Japan)). We went 120 minutes of observations by 3 minutes per frame. We call each observation as "sequence."

As Fig. 4 shows, since we did not perform attenuation correction, the radiation is attenuated by half-life of 20.3 minutes. That is, signal intensity became weak and the reconstructed image became noisy.

For comparison, we also obtain the reconstruction images with FBP method introduced in the scanner console. Here, the applied FBP method, hereafter we refer as PET-FBP method, is just a little different from the original method proposed by Ramachandran and Lakshminarayanan [6]. The original FBP method contains a filter such like $|\tilde{s}|$ in Eq.(9), which enhance high frequency component in the meaning of the spatial frequency. Thus, the high frequency noise component also enhances. The purpose of FBP filter design is to suppress the high frequency noise component. Fig.5 shows FBP filters in the spatial frequency expression. The horizontal axis shows the spatial frequency, and the vertical one shows filter strength. As shown in the figure, the shapes of filters are different from each method. The standard FBP shows the original FBP curve $|\tilde{s}|$[6]. The filter of PET-FBP method, which is designed under a radiological technologist instruction, is configured as a butterworth filter (cutoff: 1.25 cycle/cm, order:2). Our Bayes FBP filter is controlled hyperparameters β, γ, and h for each observation. Thus, each filter curve is different under the condition of observation sequence and position.

4.1 Reconstructed Images

Fig. 6 shows examples of the reconstructed images of PET-FBP and FBP-Bayes method compared with digital phantom images. Their sizes are all 128×128 [pixels]. Each pixel size nearly equals 2 [mm]. As the signal intensity get weak as time goes by, the reconstructed images of PET-FBP method seem to get noisier, and reconstructed images of FBP-Bayes method seem to get smoother.

In order to compare and evaluate the two methods more accurately, we propose a quantitative image comparison method written below.

4.2 Results

Figures 7, 8, and 9 show transitions of (mean of standard deviation) divided by (mean of mean) by time in each size of ROIs. As mentioned in previous section, we applied 5 kinds of sizes of ROI. However, we introduce 3 of them in this paper: 4×4, 8×8, and 12×12 ROI. From these results, in

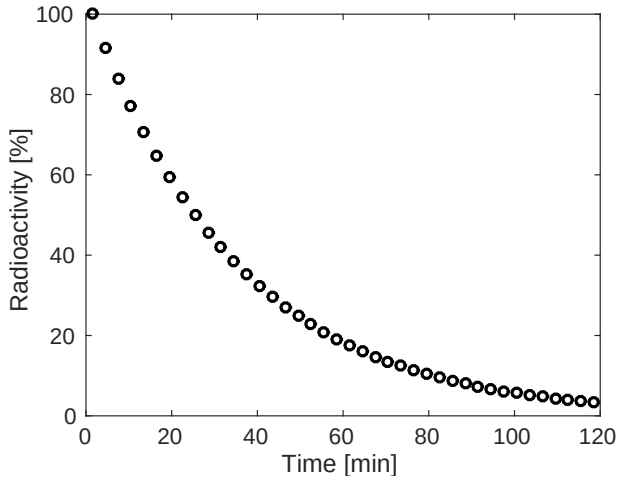


Fig. 4: Radioactive decay of $^{11}\mathrm{C}$ isotope.

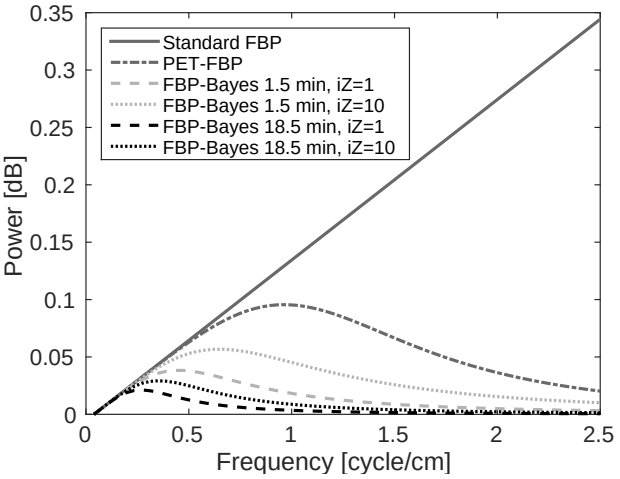


Fig. 5: Difference of frequency filter among the original FBP, PET-FBP, and FBP-Bayes which depends on hyper-parameters β, h, and γ.

any size of ROI, standard deviation of ROIs change smaller and values themselves are smaller using FBP-Bayes method than PET-FBP. Therefore, we are able to affirm that FBP-Bayes method can reconstruct images more accurate than PET-FBP method.

5. Conclusion

We evaluated images reconstructed by real observation which introduces FBP method, referred as PET-FBP method in this paper, and images reconstructed by FBP-Bayes method using a quantitative-image evaluation method we proposed. FBP-Bayes method is formulated such like the MRF-like distribution $p(\boldsymbol{\sigma})$ for the prior, and also the

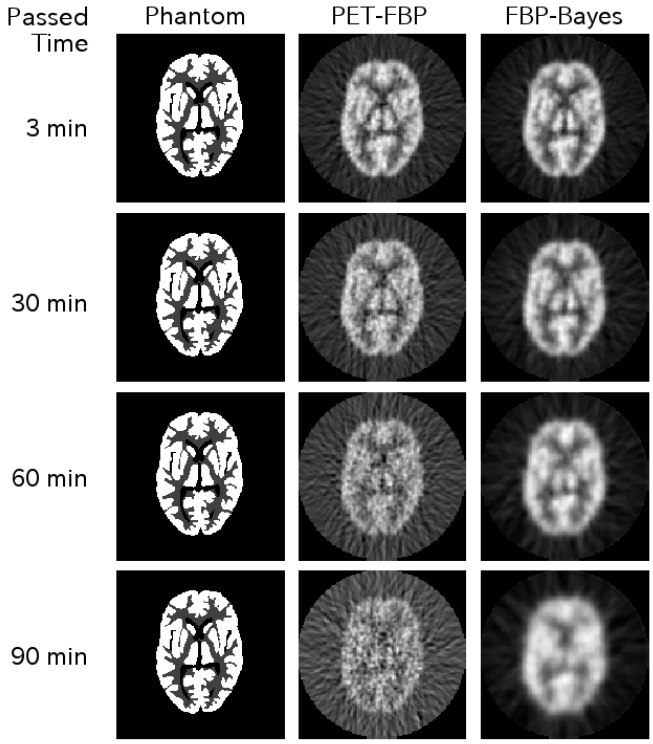


Fig. 6: Reconstructed images compared to digital phantom image. Observations were carried out 40 times in 120 minutes, where reducing radiation exposure were occurring; signal intensity get weak and the images get noisy.

observation process as $p(\boldsymbol{\tau}|\boldsymbol{\sigma})$ by assuming the process was carried out through the Gaussian channel.

To evaluate PET-FBP method and FBP-Bayes method, we proposed a quantitative image evaluation method using standard deviation of extracted ROIs. Smaller change of standard deviation of ROI indicated better accuracy in reconstructing when using FBP-Bayes method than when using the conventional PET-FBP method. However, the mean of ROI was not stable, which means there is room for improvement the FBP-Bayes method.

References

[1] L. A. Shepp and Y. Vardi: Maximum Likelihood Reconstruction for Emission Tomography, IEEE Trans. Med. Imag.Vol.1, pp.113–122, (1982).

[2] P. J. Green: Bayesian Reconstructions from Emission Tomography Data Using a Modified EM Algorithm, IEEE Trans. Med. Imag. 9 pp.84–93, (1990).

[3] K. Tanaka: Statistical-Mechanical Approach to Image Processing, J.Phys.A: Math. Gen., 37, pp. R81–R150, (2002).

[4] M. Yamasaki, H. Shouno, and M. Okada: A Bayesian Hyperparameter Inference for Radon-Transformed Image Reconstruction, Intl. J. Biomed. Imag. ID870252, (2011)

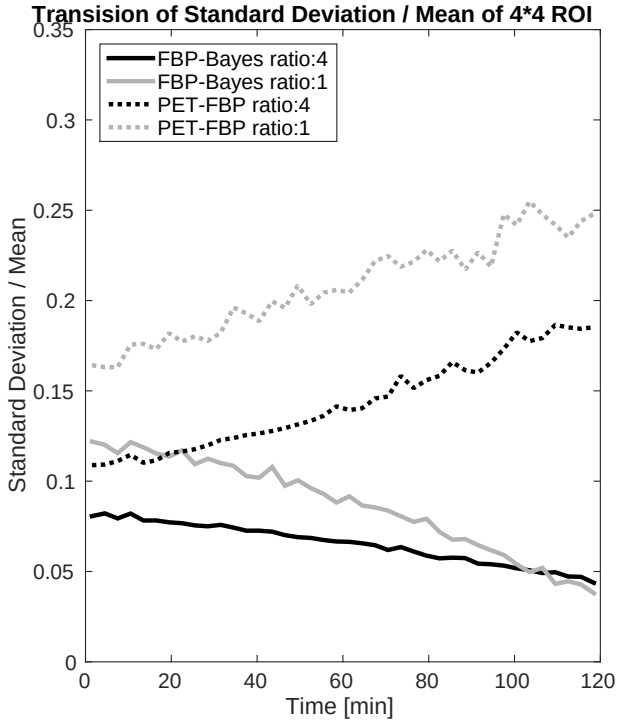


Fig. 7: Transition of standard deviation divided by mean of 4×4 pixel ROI.

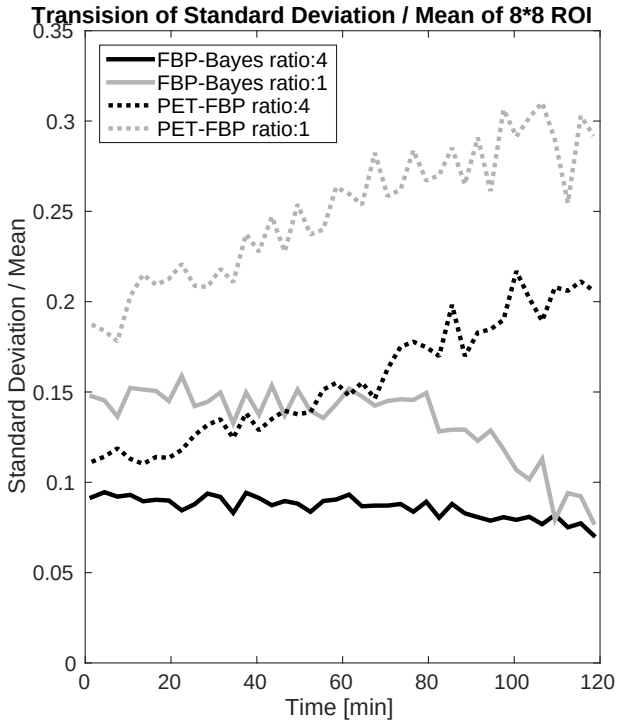


Fig. 8: Transition of standard deviation divided by mean of 8×8 pixel ROI.

[5] M. Yamasaki, H. Shouno, and M. Okada: Bayes Tomography Image Reconstruction using 4-Dimensional Markov Random Field Prior, IEICE Trans. D-II, Vol.96-D(4), pp.791–802, (2013) (in Japanese)

[6] G. N. Ramachandran and A.V.Lakshminarayanan: Three-Dimensional Reconstruction from Radiographs and Electron Micrographs, Proc. Natl. Acad. Sci.68, pp.2236–2240, (1971).

[7] K. Tanaka, H. Shouno, M. Okada, and D. M. Titterington: Accuracy of the Bethe approximation for hyperparameter estimation in probabilistic image processing, J. Phys. A: Math. Gen. 37, pp.8675–8695 (2004).

[8] J. Inoue and K. Tanaka: Dynamics of Maximum Marginal Likelihood Hyper-Parameter Estimation in Image Restoration: Gradient descent vs. EM algorithm, Phys. Rev. E 65, 016125, (2002).

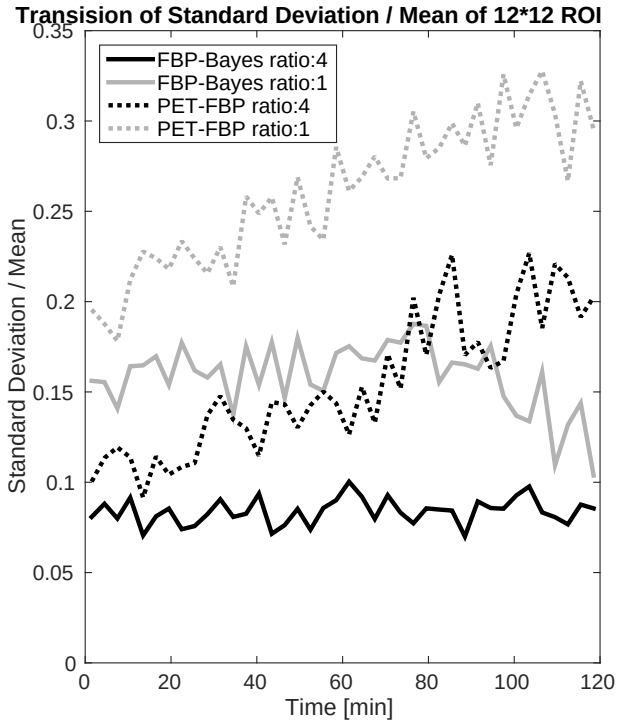


Fig. 9: Transition of standard deviation divided by mean of 12×12 pixel ROI.

SESSION

POSTER PAPERS

Chair(s)

TBA

A CPU and GPU Heterogeneous Processing of Multimedia Data by using OpenCL[1]

Heegon Kim, Sungju Lee, Yongwha Chung, Daihee Park
Dept. of Computer and Information Science, Korea University, Sejong City, Republic of Korea

Abstract - *In recent times, it has become possible to parallelize many multimedia applications using multicore platforms such as CPUs and GPUs. In this paper, we propose a parallel processing approach for a multimedia application by using both the CPU and GPU. Instead of distributing the parallelizable workload to either the CPU or GPU, we distribute the workload simultaneously to both by using OpenCL. Based on our experimental results, using both the CPU and GPU, we confirm that the proposed parallel processing approach provides better performance than typical parallel processing approaches on account of maximal utilization of the given resources.*

Keywords: CPU, GPU, Heterogeneous Computing, OpenCL

1 Introduction

As multicore processors are now widely available, parallel processing approaches have been developed for many applications. In this paper, we focus on parallelizing multimedia applications by using both the CPU and GPU. Especially, OpenCL [1] has been defined as a standard for heterogeneous parallel computing. It provides a cross-platform framework for writing software able to run on different kinds of devices, from multicore CPUs to GPUs. That is, a parallel program written with OpenCL can be executed on either a CPU or GPU [2]. Generally, it is true that a GPU can provide better performance than a CPU for multimedia applications. However, current multicore CPUs are also powerful processors, and thus, when used together with a GPU, the total execution time can be reduced.

We propose a load balancing approach that can overcome the performance limits of either CPU-only or GPU-only execution. We first attempt to achieve parallelism in a typical multimedia application (i.e., photomosaic application [3]) using OpenCL, and measure its execution time on the CPU and GPU, respectively. Then, we partition the parallelized workload into two parts, based on the relative performance of the GPU over the CPU and some parallel overhead. Finally, we assign the GPU-portion of the workload to the GPU by using a non-blocking command, and then assign the remaining parallel portion to the CPU without waiting for a result from the GPU. By reducing the idle time on either the CPU or GPU, we maximally overlap the GPU execution with the CPU execution.

2 Heterogeneous processing of Photomosaic

A GPU has many cores with low clock frequency, whereas a CPU has few cores with high clock frequency. Because a GPU is based on a Single Instruction, Multiple Data (SIMD) architecture with many cores, it can efficiently compute the same operations over large images with no data dependency, as required by the photomosaic application. That is, the photomosaic application can perform better using a GPU than a CPU. A large number of studies of GPU-equipped environments using only GPU-based parallel processing have been published [4-5]. However, a current multicore CPU is also a powerful processor, and thus, when used together with a GPU, the total execution time can be reduced.

To generate the photomosaic based on a CPU-GPU heterogeneous computing environment, we employ OpenCL [1]. Figure 1 shows the OpenCL code for generating the photomosaic using both the CPU and GPU. After initializing the CPU and GPU, they are each assigned a workload (i.e., some row pixels of the image), respectively. Because of the speed discrepancy between the CPU and GPU, we assign the workload carefully such that the possible idle time on either the CPU or GPU should be minimized. Then, the OpenCL *clEnqueueReadBuffer* function with non-blocking mode is used for execution of the code on the GPU. Finally, the OpenCL *clFinish* function is used to synchronize between the CPU and GPU.

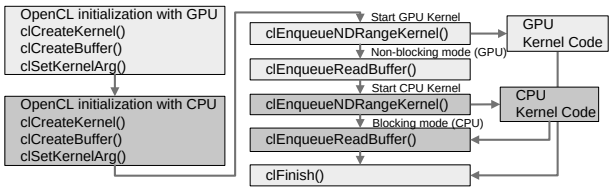


Figure 1. OpenCL code for the photomosaic

The amount of workload assigned to each processor is determined by the *clCreateBuffer* function and the *clSetKernelArg* function. In this paper, we propose a simple, but effective load balancing approach in order to reduce the possible idle time on either the CPU or GPU. We first execute the OpenCL code for CPU-only (i.e., assign 100% of the workload to CPU) and GPU-only (i.e., assign 100% of the workload to GPU) cases. Using the speed discrepancy between the CPU and GPU, we set the initial workload to each processor. Note that the non-blocking mode of the execution also incurs some overhead on the host (i.e., CPU).

[1] "This paper is being submitted as a poster".

Thus, we start searching for the optimal load distribution by comparing the execution time of the initial load distribution (i.e., determined by the speed discrepancy between the CPU and GPU) with that of the "CPU-less" load distribution (i.e., some of the initial CPU load is assigned to the GPU). If the CPU-less case provides faster execution time, this comparison is repeated until no more improvement can be found. As we will explain in the next Section, we can determine the optimal load distribution with very few comparisons.

Note that, determining the actual execution time on either the CPU or GPU analytically is a very difficult problem. Because the OpenCL code can be run on both the CPU and GPU, we can easily measure, with very few comparisons and fine tune the workload distribution, in addition to handling the speed discrepancy between the CPU and GPU (i.e., CPU-only and GPU-only cases).

3 Experimental Result

For evaluating the proposed approach, we used two platforms. *Platform 1* was composed of an AMD Phenom II X4 955 CPU (4 cores) and a GeForce GTX 285 GPU (336 cores). *Platform 2* was composed of an Intel Core i5-2500 CPU (4 cores) and a GeForce GTX 560 GPU (336 cores). The target image size was 3072×2048, and the image was partitioned into 64×64 blocks of data for the photomosaic. That is, we need to assign each of the 32 (= 2048/64) row blocks to each processor. In order to obtain the speed discrepancy between the CPU and GPU, we first executed the photomosaic for CPU-only and GPU-only cases. For platform 1, the GPU performs 2.3 times better than the CPU, whereas on platform 2, CPU performs 1.7 times better than GPU. Therefore, from the total of 32 row blocks, we set the initial load distribution CPU:GPU = 10:22 on platform 1 and CPU:GPU = 20:12 on platform 2.

Then, we compared the execution time of this initial distribution with that of CPU-less distribution. In our experiment for the minor tuning of the load distribution, we decreased the CPU workload in each comparison (i.e., the first iteration compared the workload ratio 10:22 with 9:23 for platform 1, and compared the workload ratio 20:12 with 19:13 for platform 2). Because the workload ratio 9:23 was faster than 10:22 on platform 1, we compared the workload ratio 9:23 with 8:24 in the next iteration. Similarly, since the workload ratio 19:13 was faster than 20:12 on platform 2, we compared the workload ratio 19:13 with 18:14 in the next iteration. The comparison was terminated when improvements cannot be obtained. We derived the optimal load distribution for this example (i.e., workload ratio of CPU:GPU = 7:25 for platform 1 and 18:14 for platform 2).

Note again that our approach does not need to measure all the cases. In addition to CPU-only and GPU-only (for the initial load distribution), only the cases of workload ratio from 10:22 to 6:26 (i.e., five workload ratio cases) were measured on platform 1, and only the cases of workload ratio from 20:12 to 17:15 (i.e., four workload ratio cases) were measured on platform 2, in order to derive the optimal load distribution.

Finally, Table 1 compares the speedup of CPU-only, GPU-only, and the proposed approach.

Table 1. Speedup comparison

	Speedup	
	Platform 1	Platform 2
CPU-only (CPU:GPU = 32:0)	16.8	19.5
GPU-only (CPU:GPU = 0:32)	38.5	11.7
Proposed approach	45.0 (CPU:GPU = 7:25)	26.0 (CPU:GPU = 18:14)

4 Conclusions

We have proposed an efficient heterogeneous parallel processing approach to reduce the CPU idle time in a multimedia application. The approach using OpenCL, involves using both the CPU and GPU, and decreases total execution time resulting in better performance.

Experiments with the use of both the CPU and GPU for parallel processing have demonstrated that our parallel processing approach can provide a speedup of 45 (17% better performance than the typical parallel processing approach using GPU-only) on platform 1 and a speedup of 26 (222% better performance than using GPU-only) on platform 2. Our approach exploits the main advantage of OpenCL (i.e., portability across platforms) and derives the optimal load distribution without using complicated scheduling techniques.

5 Acknowledgement

This research was supported by Basic Science Research Program(through the NRF funded by the Ministry of Education, Science and Technology, No. 2012R1A1A2043679).

6 References

[1] J. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science and Engineering*, Vol. 12, No. 3, pp. 66-73, 2010.

[2] R. Gaetano and B. Pesquet-Popescu, "OpenCL Implementation of Motion Estimation for Cloud Video Processing," in *Proc. of International Symposium on Multimedia Signal Processing*, pp. 1-6, 2011.

[3] R. Silvers and M. Hawley, *Photomosaics*, Henry Holt, New York, 1997.

[4] J. Cao, X. Xie, J. Liang, and D. Li, "GPU Accelerated Target Tracking Method," *Advances in Intelligent and Soft Computing*, vol. 128, pp. 251-257, 2012.

[5] D. Davendra and I. Zelinka, "GPU Based Enhanced Differential Evolution Algorithm: A Comparison between CUDA and OpenCL," *Intelligent Systems Reference Library*, vol. 38, pp. 845-867, 2013.

On the Scalability of Parallel Quicksort: A Case Study on Distributed vs. Shared-Memory Models

David Paulius
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida, USA
davidpaulius@mail.usf.edu

Marc Boumedine
Computational and Computer Sciences Department
College of Science and Mathematics
University of the Virgin Islands
St. Thomas, U.S.V.I
mboumed@uvi.edu

ABSTRACT/POSTER PAPER — The increasing availability of parallel systems affords undergraduates opportunities to experience firsthand the potential benefit and pitfalls of parallel programming. However, a clear understanding of the underlying architecture is critical to achieve the anticipated speed up of serial programs. To this end, analyzing performance metrics is essential for tuning parallel implementations and determining what improvement can be expected by scaling out or increasing the number of processes [1-4]. Without considering important aspects such locality, load balancing and communication overheads associated with each available parallel system, students are having a hard time obtaining the intended performance and determining which approach is suitable to solving the problem at hand. Finding the best mapping between the parallel implementation and the parallel platform available is hard to achieve without clear guidance or practical experiences. Given some legacy code, a first approach consists of looking at parallel programming patterns such as divide-and-conquer or decomposition techniques. This pattern has been used widely to design a number of efficient algorithms for a variety of applications [5]. Students are already familiar with the approach of recursively partitioning a problem into smaller sub-problems. Identification of sub-problems which can be solved independently is an intuitive choice for achieving speed up. Even if this is not always the case, this approach allows students to quickly implement embarrassingly parallel programs using message passing and shared-address space programming paradigms. Ultimately, performance metrics are collected and contrasted with the serial and parallel runtime implementations. Sources of overhead affecting the performances of the algorithms are examined and discussed. This practical approach can be repeated on various classical problems (matrix multiplication, sorting, numerical integration and the gravitational N-body) and extended to new problems by applying various parallel programming patterns

This case study walks through the steps of solving the problem of speeding up sorting serial problems from legacy code for both shared memory and distributed memory models on a small system multi-core cluster called *Bucc*. Bucc is a small cluster available to undergraduate students at the University of the Virgin Islands who are exposed to high-performance computing applications and are interested in solving computational science problems. Bucc consists of a head node with 2 Intel Westmere, 2.40GHz, six-core, processors and five twin compute nodes, each with two Intel 2.40GHz six-core processors. It is configured with 2 GB of RAM per core. The interconnects are InfiniBand 40Gbps.

This study focuses specifically on the sorting task for both the distributed and shared-memory models on the Bucc system. Sorting is an recurrent critical problem for organizing and processing efficiently information in many fields such as combinatorial searching, optimization, simulation, information retrieval, scientific computing etc. Sorting is also used as a core utility in many libraries. Because of its essential role computer scientists have devoted many studies on sorting techniques and have been focusing on finding the fastest serial and parallel implementations. The *quicksort* algorithm is one of the most popular algorithms known for its time complexity of $O(N \log N)$. As a divide-and-conquer algorithm, it uses the notion of *pivoting* to divide the data set into two halves or subsets that are recursively solved; there are several ways of determining the pivot such as finding the median, selecting the middle value, or simply selecting a random value in the data set. Selecting the pivot is important as it ultimately affects the time taken to sort and merge all the results together. The *quicksort* algorithm has the worst case of being $O(N^2)$, which can render it as slow as the selection and bubble sorting algorithms. Due to its divide-and-conquer nature, intuitive parallel implementations are usually implemented using message passing interface libraries (MPI) and shared-addess space libraries (OpenMP). Other hybrid programming models are also considered but not examined in this study.

The naive parallel formulation of the serial quicksort consists of sorting a global array A of n element on p processes. Each process is given a sub-array which is a sequence of n/p items. The sub-array is partitioned recursively around a pivot using a compare and split routine. The algorithm terminates when the sub-array can not be partitioned anymore. Fig. 1 show the serial quicksort runtime of Bucc. The performance and scalability of parallel programs (MPI and OpenMP) are discussed. OpenMP quicksort spawns t threads each of them handle a partition of the array. Each thread uses the serial *qsort()* from the C libraries [6]. Once all partitions have been sorted, the sub-arrays are then merged together. The

runtimes were recorded during various experiments by varying the size of the data size and the number of processes (see Figs. 2-5). The array sizes were increased incrementally (1000 x 2^N items, where *N= 0,...,12).* The MPI experiments shown on Figs. 2-3 were derived with *p* processes (p=1,...,128). Figs. 4-5 show OpenMP implementation results by increasing the number of threads from 1 to 512. The data sets were randomly generated with integers using the C's *drand()* function for better generalization purposes. Runtimes were recorded and plotted on Figs. 1-5. Common performance metrics such as *speedup* and *efficiency* were computed on Bucc. Speedup is the performance gain of parallel processing versus sequential processing. It is defined as [7-8]:

$$S(p)= T(N,1)/T(N,p)$$

where *T(n,1)* is the runtime on one processor executed on *size N* input, *T(n,p)* is the runtime on *p* processors executed on *size* N input.

Efficiency of a parallel program is a measure of how much of available processing power is being used.

$$E(p) = S(p)/ p$$

where *S(p)* is the parallel runtime on *p* processors

This metric provides an accurate measure of the true efficiency of a parallel program compared to CPU usage (redundant calculations and idle times are included) [1-3].

Fig.6 shows speedup results for the MPI and OpenMP implementations over the serial quicksort. Overall, the results for the *OpenMP implementation* show significant performance improvement on over the MPI implementation on relatively large problem size (N= 4,096,000), where 16 threads yield a ~700% time performance increase (see Fig. 7). Both parallel implementations improve performance when item sizes are greater than 32,000. However, the naive parallel implementations have no gain for small problem sizes (*N < 32,000*) on the Bucc architecture. Figs 6-7 show that the current quicksort implementations are not scalable on Bucc. For some experiments, the parallel versions solve the problem with a superlinear speed up for (OpenMP with 8, 16 and 32 threads). On the other hand, the execution with 2 and 4 threads results in a sublinear speedup (see Fig. 7). Based on these results, it is clear that the experiments conducted with the current parallel implementations must be re-examined in order to identify sources of parallel overhead such as load imbalancing, intranode and internode communication overhead, the partitioning of the array and the size of the cache available at each compute node. The pivot selection is also critical in avoiding idle processes/threads and must be considered in revised versions.

This case study highlighted the challenges undergraduate students are facing during the design and development of parallel applications. Students were tasked to parallelize the serial quicksort algorithm on Bucc cluster. Naive parallel algorithms were implemented using OpenMP and MPI. These implementations were compared in terms of speed up, efficiency and scalability. Results have shown that the parallel naive implementations can be improved significantly by reviewing the designs and implementations as well as taking into account the underlying architecture. In the future we want to combine the benefits of MPI and OpenMP code with a hybrid programming model and examine speedup and scalability metrics on Bucc.

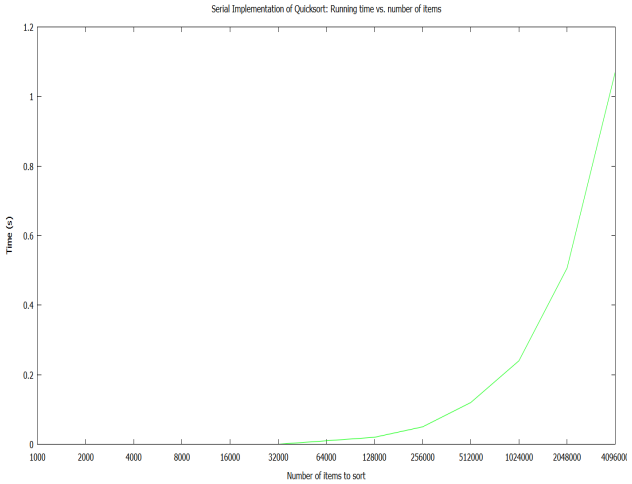


Fig. 1: Serial Quicksort Running time vs. problem sizes

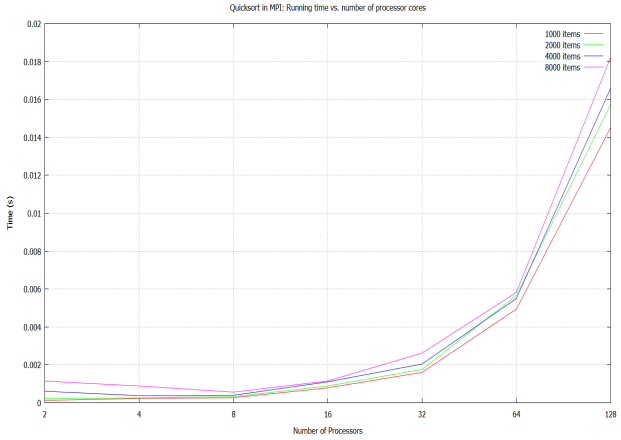


Fig. 2: MPI Quicksort runtimes on Bucc for small size sets

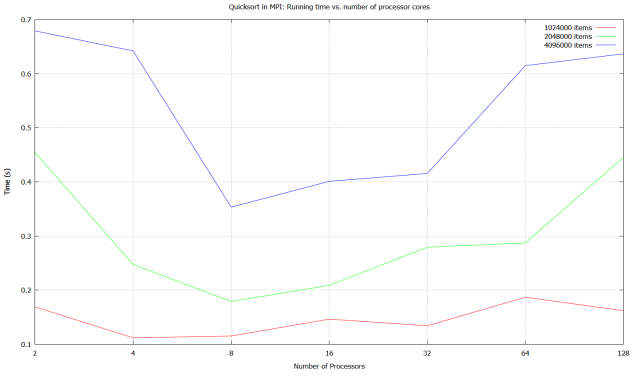


Fig. 3: Quicksort MPI implementation on Bucc for large data sets

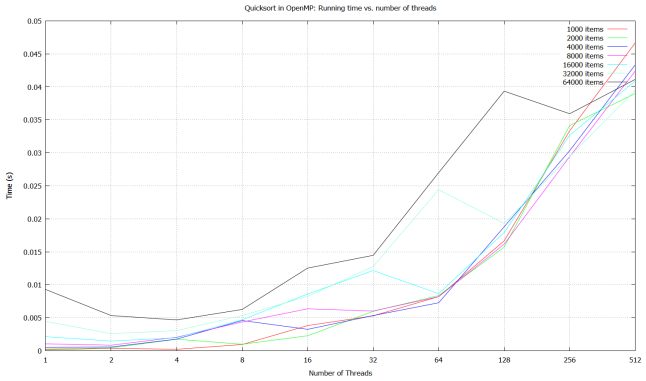


Fig. 4: Quicksort OpenMP implementation on Bucc

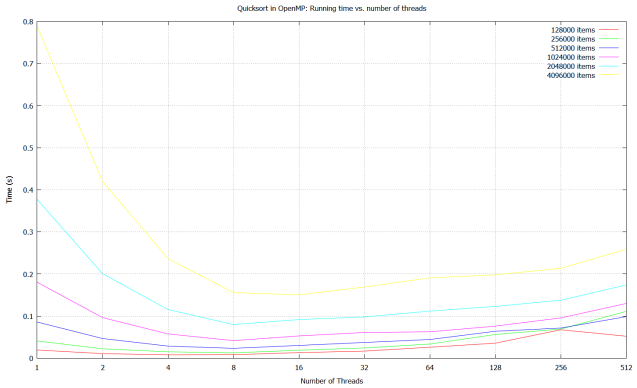


Fig. 5: Quicksort OpenMP implementation on Bucc

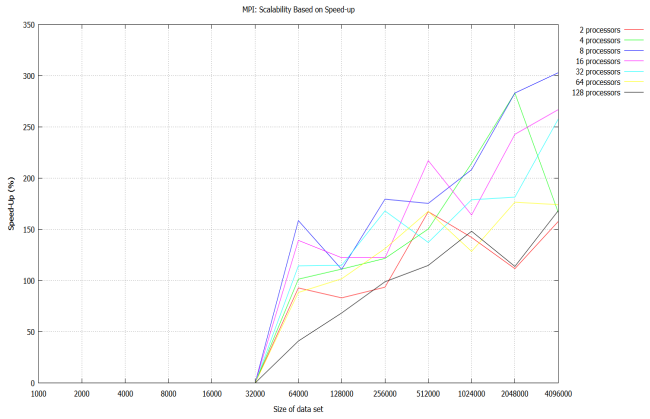


Fig. 6: MPI implementation Speed up

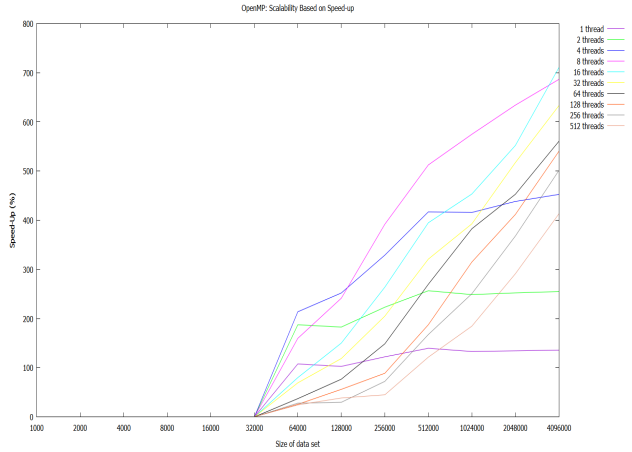


Fig.7:OpenMP Speed up (%)

REFERENCES

[1] Rajput, I., Kumar, B. and Singh, T. 2012. *Performance Comparison of Sequential Quick Sort and Parallel Quick Sort Algorithms,* http://research.ijcaonline.org/volume57/number9/pxc3883363.pdf

[2] Foster, I. *1995. Designing and Building Parallel Programs,* http://www.mcs.anl.gov/~itf/dbpp/

[3] Miller, R. and Boxer, L. 2013. *Algorithms Sequential and Parallel: A Unified Approach, 3rd Edition.*

[4] *Grama A., Gupta A., Karypis G. , Kumar V.(2003) Introduction to Parallel Computing: Design and Analysis of Algorithms, Second Edition, Addison Wesley.*

[5] Berna L. Massingill, Timothy G. Mattson, and Beverly A. Sanders "Reengineering for Parallelism: An Entry Point for PLPP (Pattern Language for Parallel Programming) for Legacy Applications"Proceedings of the Twelfth Pattern Languages of Programs Workshop (PLoP 2005), 2005. http://www.idris.fr/eng/turing/turing-mapping-eng.htmlD

[6] Wiesland, C. 2012. *Hypercube-Quicksort-MPI,* https://github.com/utahwithak/Hypercube-Quicksort-MPI/blob/master/QuickSort/main.c

[7] Monteiro, M. 2010. *Parallel Programming: Parallel Algorithms Sorting,* http://paginas.fe.up.pt/~apm/CPAR/docs/cpar12.pdf

[8] Kumar, V. and Gupta, A. 1993. *Analyzing Scalability of Parallel Algorithms and Architectures,* Journal of Parallel and Distributed Computing, volume 22, pgs. 379-391.

SESSION

PARALLEL PROCESSING ALGORITHMS, SYSTEMS, APPLICATIONS, AND RELATED ISSUES

Chair(s)

TBA

Distributed Caching Using the HTCondor CacheD

Derek Weitzel, Brian Bockelman,
and David Swanson
Computer Science and Engineering
University of Nebraska – Lincoln
Lincoln, Nebraska 68588
Email: dweitzel@cse.unl.edu

***Abstract*—A batch processing job in a distributed system has three clear steps, stage-in, execution, and stage-out. As data sizes have increased, the stage-in time has also increased. In order to optimize stage-in time for shared inputs, we propose the CacheD, a caching mechanism for high throughput computing. Along with caching on worker nodes for rapid transfers, we also introduce a novel transfer method to distribute shared caches to multiple worker nodes utilizing BitTorrent. We show that our caching method significantly improves workflow completion times by minimizing stage-in time while being non-intrusive to the computational resources, allowing for opportunistic resources to utilize this caching method.**

I. Introduction

Large input datasets are becoming common in scientific computing. Unfortunately for campus researchers, the staging time of the datasets to computational resources has not kept pace with the increase in dataset sizes. The typical large dataset workflow may consist of thousands of individual jobs, each sharing input files.

The campus resources made available to researchers are shared; therefore, the researchers have the limitation of not having access to install programs on the clusters. Previous work [1] built an overlay on top of campus resources to create a virtual, on-demand pool of resources for task execution. We expand the capabilities of this virtual pool to include data caching and novel transfer methods to enable big data processing.

An excellent example of a big data workflow is that of the bioinformatics application: BLAST [2]. Each BLAST query requires an entire reference database, which can range in size from a few kilobytes to many gigabytes. The workflow to run a BLAST query requires a large stage-in time in order to make the reference database available. Additionally, the databases are frequently updated with new entries.

Users in the past have copied the database using various methods. The naïve method includes copying the database for each job. Storing the database on a shared filesystem has the same effect as copying the database for each job, since the database must be transferred to the execution node for each job. We propose caching the database on the node for subsequent executions.

We find that the BLAST workflow described above is common among large data researchers.

Bosco [1] is a remote submission tool that can create overlay virtual pools designed for campus resources. In previous work, Bosco allowed campus researchers to submit high throughput jobs to high performance clusters. We extend Bosco to include data caching and novel data transfer methods.

We limit our design and analysis to a campus cluster computing environment. Our solution is unique in that it is designed to run opportunistically on the campus computing resources. Additionally, they do not require administrator intervention in order to create a virtual, on-demand pool of resources.

II. Background and Related Work

Data caching on distributed systems has been used many times and at many levels. Caching can be done on the storage systems and on the execution hosts, as well as well as in within the infrastructure separating the two.

Some distributed filesystems use local caches on the worker nodes. GPFS [3] has a read-only cache on each worker node that can cache frequently accessed files. It is designed for a fast, shared filesystem and is recommended when file access latency is a concern. It is not recommended for large files since internal bandwidth to the local disk is assumed to be less than the bandwidth available to the GPFS shared filesystem. GPFS file transfers are typically done over high speed interconnects which can provide high bandwidth for large files. These interconnects are not typically available to a user's jobs for transferring data from a remote source.

HTTP caching is used throughout the web to decrease latency for page loads and to distribute requests among servers. In high throughput computing, a forward proxy is commonly used to cache frequent requests to external servers. The forward proxy caches files requested through it, and will respond to subsequent requests for the same file by reading it from memory or its own disk cache.

The HTTP forward proxy caching does have limitations. The HTTP protocol was designed and is used primarily for websites. Websites have very different requirements from high throughput computing. The data sizes are much smaller. Software designed as forward proxies, such as Squid [4], are optimized for web HTTP traffic, and therefore do not handle large data file sizes optimally. Further, the Open Science Grid (OSG) [5] sites typically only have one or possibly a few squid caches available to user jobs. They are not designed to scale to large transfers for hundreds of jobs, our target use case.

Parrot [6] is another application that will cache remote files when using certain protocols. Parrot uses interposition [7] to capture and interpret IO operations by an unmodified binary application. The interposition allows Parrot to provide a transparent interface to remote data sources. Parrot caches some of those sources such as HTTP with GROW-FS, a filesystem using HTTP. Parrot caches an entire file to the local storage. Parrot must download directly from the source the first time it is requested, exhausting WAN bandwidth quickly for large files.

CernVM-FS [8] provides a filesystem over the HTTP protocol. It integrates into the worker node system using the FUSE [9] interface. The CernVM-FS local node client caches files on the node, as well as using Squid to cache files at the site. Again, since it uses the HTTP, it's not designed to cache large files. Neither the Squid caches nor the web servers optimally transfer large files, nor are they designed for large data sizes. Further, CernVM-FS requires administrator access in order to install and configure, a privilege that campus users do not have.

XrootD [10] is designed for large data access, and it has even been used for WAN data transfers [11] using a federated data access topology. There has been some work in creating a caching proxy for the XrootD [12]. The caching proxy is designed to cache datasets on filesystems near the execution resources. The caching proxy requires installation of software and the running of services on the cluster. Unprivileged campus users will be unable to run or install these services.

We define local caching as saving the input files on the local machine and making them available to local jobs. Local caching is different from site caching, which is done in the OSG by Squid caches. We define site caching as when data files are stored and available to jobs from a closer source than the original. In most cases on the OSG, the site cache is a node inside the cluster that has both low latency and high bandwidth connections to all of the execution hosts.

We use distributed transfer to mean transfers that are not from a single source. In our case, we will be using BitTorrent [13], in which a client may download parts of files from multiple sources. Additionally, the client may make available to other clients parts of the files that have already been downloaded.

BitTorrent is a transfer protocol that is designed for peer-to-peer transfers of data over a network. It is optimized to share large datasets between peers. The authors of [14] and [15] discuss scheduling tasks efficiently in peer-to-peer grids and desktop grids. Their discussion does not take into account the network bottlenecks that are prevalent in campus cluster computing.

In [16], the authors use scheduling, caching, and BitTorrent in order to optimize the response time for a set of tasks on a peer-to-peer environment. They build the BitTorrent and caching mechanisms into the middleware which is installed and constantly running on all of the peers. They do not consider the scenario of opportunistic and limited access to resources. Their cluster size is statically set, and therefore may not see the variances that users of campus clusters may see.

III. Implementation

The HTCondor CacheD is a daemon that runs on both the execution host and the submitter. For our purposes, a cache is defined as an immutable set of files that has metadata associated with it. The metadata can include a cache expiration time, as well as ownership and acceptable transfer methods.

The CacheD follows the HTCondor design paradigm of a system of independent agents cooperating. Each CacheD makes decisions independently of each other. Coordination is done by CacheDs communicating and negotiating with each other.

Each caching daemon registers with the HTCondor Collector. The collector serves as a catalog of available cache daemons that can be used for replication.

In addition to the CacheD, a transfer plugin is used to perform the cache transfers in the job's sandbox. The plugin uses an API to communicate with the local CacheD to request local replication requests to the local host. After the cache is transferred locally, the plugin then downloads the cache to the job's working directory.

Expiration time is is used for simple cache eviction. A user creates a cache with a specific expiration time. After a cache has expired, a caching server may delete it to free space for other caches. The expiration may be requested to be extended by the user

The CacheD supports multiple different forms of transferring data. Using HTCondor's file transfer plugin interface, it can support pluggable file transfers. For this paper, we will only use the BitTorrent and Direct transfer methods. The BitTorrent method uses the libtorrent library to manage BitTorrent transfers and torrent creation. The Direct method uses an encrypted and authenticated stream to transfer data from the source to the client.

An important concept of the caching framework is a cache originator. The original daemon that the user uploaded their input files to is the cache originator. The cache originator is in charge of distributing replication requests to potential nodes, as well as providing the cached files when requested.

The caching daemons interact with each other during replication requests. A cache originator sends replication requests to remote caching daemons that match the replication policy that is set by the user. The remote caching daemon then confirms that the cache data can be hosted on the server. The remote cache then initiates a file transfer in order to transfer the cached data from the origin to the remote CacheD.

The receiving CacheD can deny a replication request for many reasons, including:

- The resource does not have the space to accommodate the cache.
- The resource may not have the necessary bandwidth available in order to transfer the cache files.
- The resource does not expect to be able to run the user's jobs and has determined that the cached files will not be used.

The ability of the receiving CacheD to deny a replication request follows HTCondor's independent agent model.

The policy expression language is modeled after the matchmaking language in the HTCondor system [17]. The caching daemon is matching the cache contents to a set of resources; therefore, it is natural to use HtCondor's same matchmaking language that is used to match jobs to resources. Once a resource is determined to match the cache's policy expression, the caching daemon will contact the resource's caching daemon in order to initiate a cache replication. The caching daemon on the remote resource is an independent agent that has the ability to deny a caching replication even after matchmaking is successful.

Libtorrent is built into the CacheD to provide native BitTorrent functionality. The CacheD is capable of creating torrents from sets of files in a cache, as well as downloading cache files using the BitTorrent protocol. Since this is a distributed set of caches, we will not use a static torrent tracker. Rather, we will use a Distributed Hash Table [18] and local peer discovery [19] features of the BitTorrent protocol. This ensures that there are no single points of failure.

A. Creation and Uploading Caches

The user begins using the caching system by uploading a cache to their own CacheD, which then becomes the cache originator. This is very similar to a user submitting a job to their own HTCondor SchedD. Using the cache's metadata, the CacheD decides whether to accept or reject the cache. If the CacheD accepts the cache, it stores the metadata into resilient storage. The user then proceeds to upload the cache files to the CacheD.

The CacheD stores the cache files into its own storage area. Once uploaded, the CacheD takes action to prepare the cache to be downloaded by clients. This includes creating a BitTorrent torrent for the cached files.

Numerous protections are used in order to ensure proper usage of the CacheD. The upload size is enforced to the size advertised in the metadata. The client cannot upload more data to the CacheD than was originally agreed upon during cache creation. Further, the ownership of the cache is stored in the metadata, and is acquired by authenticating with the client upon cache creation. Only the owner may upload and download files from the cache directly.

A client may mark a cache as only allowing certain replication methods. This can be useful if a user wishes to keep data private. BitTorrent doesn't offer the authorization framework to ensure privacy of caches. Users may mark the cache as only allowing DIRECT replications, which are encrypted and authenticated.

B. Downloading Caches

When a job starts, the CacheD begins to download the cache file. The cache is identified by a unique string that includes the cache's name and the cache's originator host. The flow of replication requests is illustrated in Figure 1. The replication requests originate from the file transfer plugin, which sends the replication request to the node local CacheD. The node local CacheD then sends the replication to its parent or the origin cache. The propagation of replication requests are modeled after well-known caching mechanisms such as DNS.

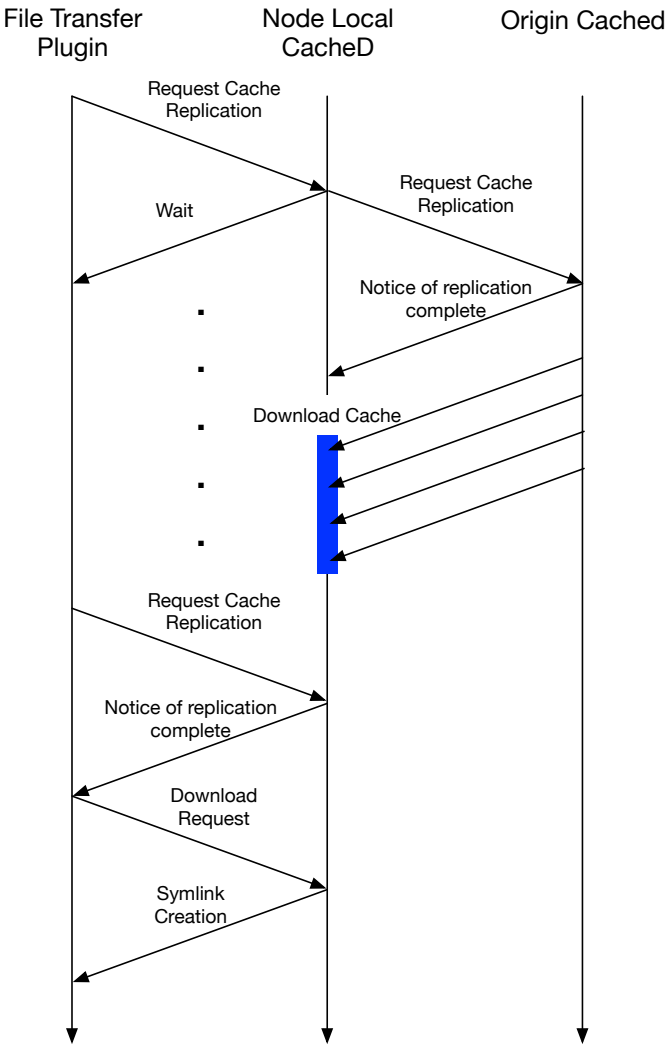


Fig. 1. Flow of Replication Requests

1) The plugin contacts the node local CacheD daemon on the worker node. It requests that the cache is replicated locally in order to perform a local transfer.
2) The node local CacheD responds to the file transfer plugin with a "wait" signal. The file transfer plugin polls the node local CacheD periodically to check on the replication request.
3) The local CacheD daemon propagates the cache replication request to its parent, if it exists. If the CacheD does not have a parent it contacts the cache originator in order to initiate a cache replication.
4) If the cache is detected to be transferable with BitTorrent, the download begins immediately after receiving the cache's metadata from the parent or origin.

5) Once the cache is replicated locally, the plugin downloads the files from the local CacheD.

Each download is negotiated for the appropriate transfer method between the parent and the client. Between parent and client CacheD's, the cache's individual replication preferences are honored. Between a CacheD and the transfer plugin, an additional protocol is offered: symbolic link (symlink).

If the transfer plugin successfully authenticates with a local CacheD, transfer methods are negotiated. If supported, the symlink method may be chosen. The symlink transfer method allows near instant transfer of the cache from the CacheD to the plugin. A symlink is created by the CacheD in the job's working directory pointing to the cache directory. This symlink method eliminates transferring the cache to each job.

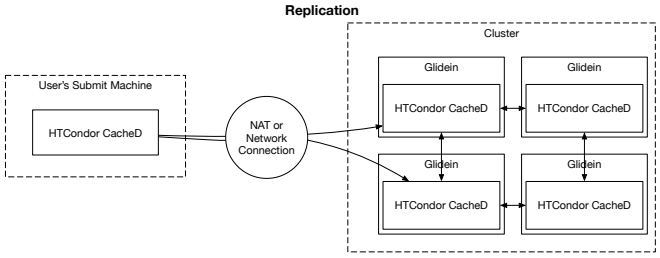


Fig. 2. Cache Replication Showing Bottleneck

In Figure 2, you can see a traditional configuration of a cluster. The configuration shows that there is a Network Address Translation bottleneck or a network bottleneck between the submit machine and the execution nodes. The bottleneck limits the bandwidth between the submit machine and the execution nodes.

C. Parenting of CacheDs

During testing of the CacheD, it was apparent that BitTorrent increases the IO queue on the host server significantly, degrading the IO performance for all jobs on the server. This increased IO queue leads to competition between BitTorrent-enabled CacheD's on the same host. In order to address the increased IO queue, each CacheD will designate a single daemon on the host that downloads the files through BitTorrent. All other CacheDs will then download the cache from the parent using Direct file transfer mechanisms.

IV. Results

A. Experimental Design

To evaluate our solution, we will run a BLAST benchmark from UC Davis [20]. We chose a BLAST benchmark due to many factors. BLAST is used frequently on campuses, but used infrequently on clusters due to the size of the database. BLAST has very large databases that are required by each job. This makes it difficult to use on distributed resources since each job requires significant data. BLAST databases are frequently updated, making them poor candidates for static caching, but good candidates for short-term caching, for which our CacheD specializes.

The BLAST database distributed with the benchmark is a subset of the Nucleotide NR database. In our tests, we will use a larger subset of the NR database in order to demonstrate the efficiency of our solution.

For researchers, the time to results is likely the most important metric. The stage-in time of data can be a large component of the entire workflow time. We will measure the time for stage-ins as well as the average stage-in time.

We designed two experiments that represent our experience on campus infrastructure. In the first experiment, we will allow 100 simultaneous jobs to start at the same time and measure the average download time versus the number of distinct nodes. This experiment also includes the download time for child caches. We chose 100 jobs somewhat arbitrarily in order to completely fill all of the nodes we were allocated on the cluster.

In the second experiment, we compare the total stage-in time for a variable number of jobs while number of distinct nodes remains constant at 50. This will show that the cache is working to eliminate transfer times when the files are already on the node. Further, it will compare HTCondor's File Transfer method versus the CacheD's two transfer methods: BitTorrent and Direct.

When the number of jobs is fewer than 50, each job must download the cache since there are 50 nodes available for execution. When the number of jobs is more than 50, all jobs that run after the initial download use a cached version of the data.

In our experiments, each job will use the CacheD to stage-in data to the worker nodes. The jobs will be submitted with glideins created by Bosco [1] and the Campus Factory [21]. Bosco allows for remote submission to campus resources while the Campus Factory allows for on-demand glidein overlay of remote resources. The Campus Factory is used in order to create and run glideins which, in turn, run the CacheD daemon. Bosco was used in order to submit to multiple campus resources simultaneously.

These two experiments were conducted on a production cluster at the Holland Computing Center at the University of Nebraska–Lincoln (UNL).

B. Results

We completed 41 runs of the BitTorrent versus Direct transfer experiments on the UNL production cluster. We first confirmed our suspicion that the Direct transfer method would result in a linear increase in the average stage-in time to transfer the cache as we increased the number of distinct nodes. Conversely, we found that the BitTorrent transfer method did not significantly increase the average stage-in time as we increased the number of distinct nodes. The BitTorrent transfer method was faster than the Direct in all experiments.

Figure 3 shows that the BitTorrent transfer method is superior to Direct for all experiments that were run. Since multiple CacheDs on the same node will parent to a single CacheD, the number of distinct nodes is the dependent variable. After the parent cache downloads the cache for the node, then each

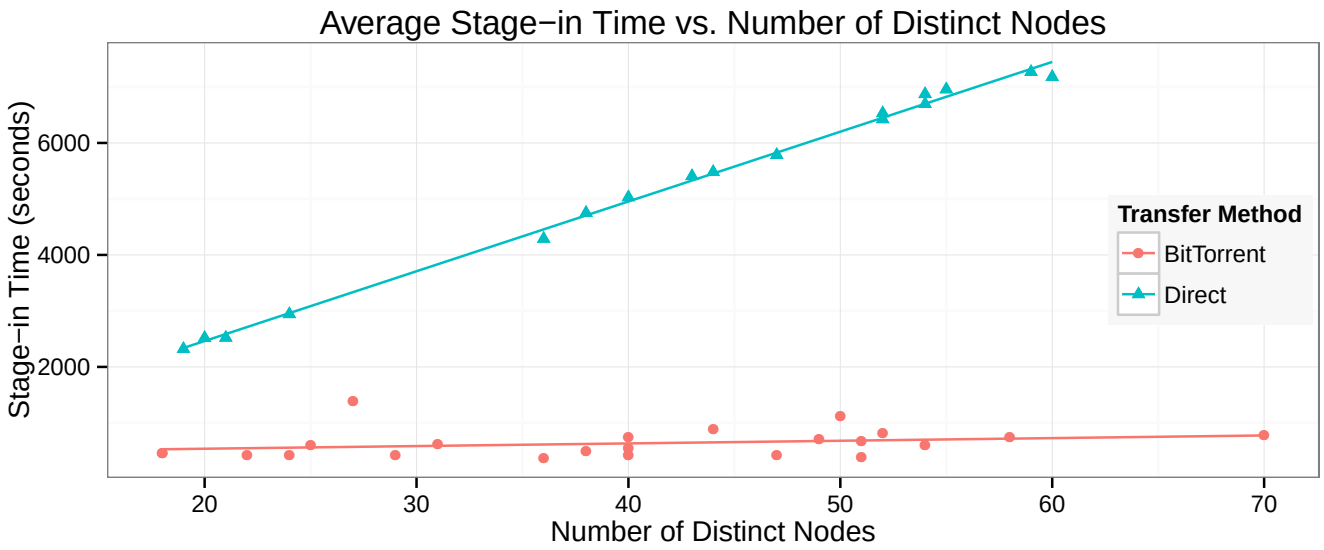


Fig. 3. Comparison of Direct and BitTorrent Transfer Methods with Increasing Distinct Node Counts

child cache will download from the parent using the Direct transfer method.

The Direct method of transfer follows a linearly increasing time to download the cache files. This can be explained by bottlenecks of the transfers between the host machine and the execution nodes. The increase in number of distinct nodes increases the stage-in time for any individual node.

The average download times for BitTorrent stage-ins are also shown in Figure 3. The stage-in time does not significantly increase as the distinct nodes increases. This meets our expectations. We expect this trend to continue as the number of distinct nodes increases since BitTorrent can use peers to speed up download time.

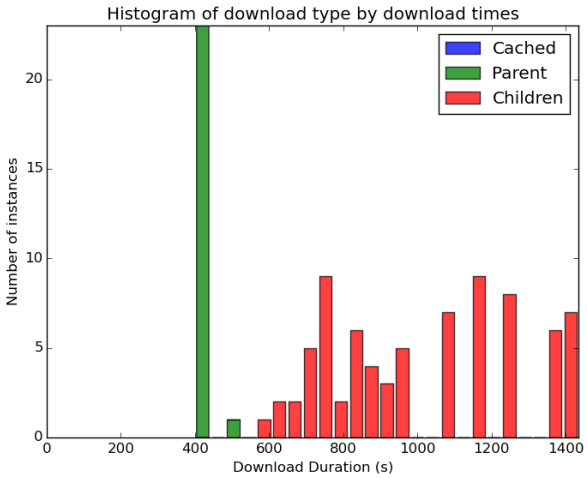


Fig. 4. Historgram of Transfers Mode vs Download Times

To better illustrate how parenting affects the download time of a cache, we show a histogram of the different modes in Figure 4. The figure shows that while the parents download first, and nearly at all the same time, the children take a variable amount of time to download. This variability can be attributed to the number of children on a node. The more children downloading the cache at the same time, the slower each download will take.

For our second experiment, we calculated the total stage-in time for a variable number of jobs.

When we limit the number of nodes to 50, we can clearly see the effect of the caching by varying the number of jobs. In Figure 5, both the Direct and BitTorrent transfer methods have a natural bend at about 50 jobs. This correlates to when the CacheD has on-disk caches of the datasets, and the transfer to the job's sandbox is nearly instantaneous.

The HTCondor file transfer method has a shorter stage-in time for low numbers of distinct nodes than the Direct method. This can be explained by the increased overhead that the CacheD introduces when transferring datasets. After all 50 nodes have the dataset cached locally, the Direct transfer method becomes more efficient than the HTCondor file transfers.

V. Conclusions

We have presented the HTCondor CacheD, a technique to decrease the stage-in time for large shared input datasets. Our experiments proved that the CacheD decreases stage-in time for these datasets. Additionally, the transfer method that the CacheD used can significantly affect the stage-in time of the jobs.

The BitTorrent transfer method proved to be a efficient method to transfer caches from the originator to the execution hosts. In fact, the transfer time for jobs did not increase as the number of distinct nodes requesting the data increased. Any

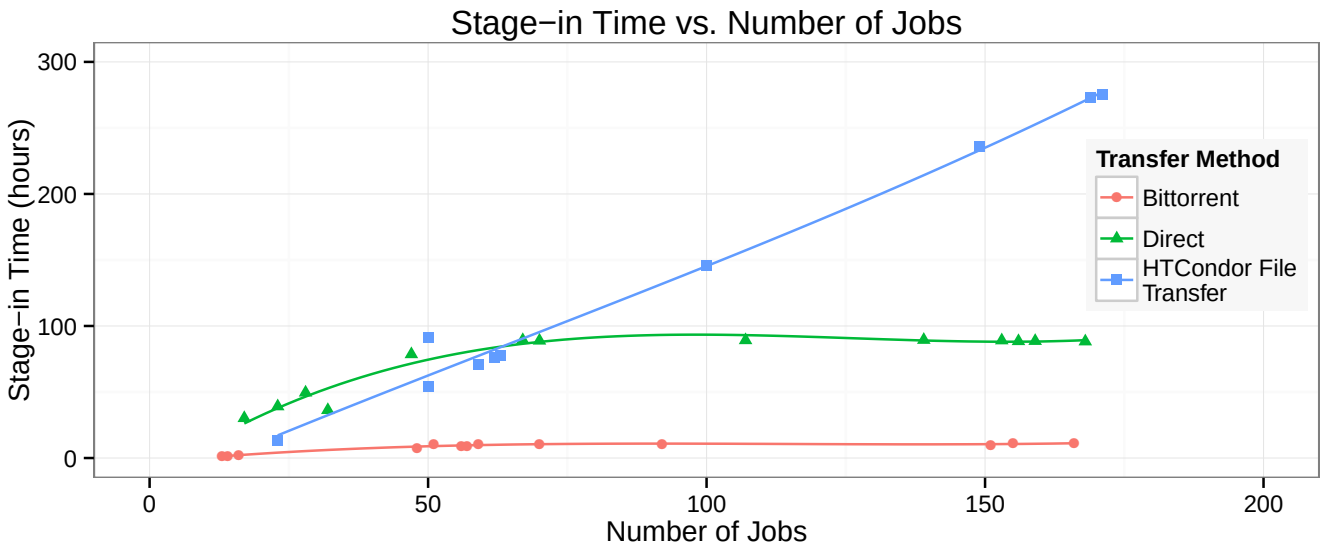


Fig. 5. Transfer Method vs Number of Jobs

bottlenecks that surround the cluster are therefore irrelevant using the BitTorrent transfer method.

In the future we plan to investigate incorporating job matchmaking with cache placement. The HTCondor Negotiator could attempt to match jobs first against resources that have the input files before matching against any available computing resources.

Acknowledgment

This research was done using resources provided by the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science.

References

[1] D. Weitzel, I. Sfiligoi, B. Bockelman, J. Frey, F. Wuerthwein, D. Fraser, and D. Swanson, "Accessing opportunistic resources with bosco," *Journal of Physics: Conference Series*, vol. 513, no. 3, p. 032105, 2014.

[2] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psi-blast: a new generation of protein database search programs," *Nucleic acids research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[3] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters." in *FAST*, vol. 2, 2002, p. 19.

[4] Squid. (2015) Squid: optimizing web delivery. [Online]. Available: http://www.squid-cache.org/

[5] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein *et al.*, "The open science grid," in *Journal of Physics: Conference Series*, vol. 78, no. 1. IOP Publishing, 2007, p. 012057.

[6] D. Thain and M. Livny, "Parrot: An application environment for data-intensive computing," *Scalable Computing: Practice and Experience*, vol. 6, no. 3, pp. 9–18, 2005.

[7] ——, "Multiple bypass: Interposition agents for distributed computing," *Cluster Computing*, vol. 4, no. 1, pp. 39–47, 2001.

[8] J. Blomer, P. Buncic, and T. Fuhrmann, "Cernvm-fs: delivering scientific software to globally distributed computing resources," in *Proceedings of the first international workshop on Network-aware data management*. ACM, 2011, pp. 49–56.

[9] M. Szeredi *et al.*, "Fuse: Filesystem in userspace," *Accessed on*, 2010.

[10] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky, "Xrootd-a highly scalable architecture for data access," *WSEAS Transactions on Computers*, vol. 1, no. 4.3, 2005.

[11] L. Bauerdick, D. Benjamin, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, M. Ernst, R. Gardner, A. Hanushevsky, H. Ito *et al.*, "Using xrootd to federate regional storage," in *Journal of Physics: Conference Series*, vol. 396, no. 4. IOP Publishing, 2012, p. 042009.

[12] L. Bauerdick, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, J. Dost, I. Sfiligoi, A. Tadel, M. Tadel, F. Wuerthwein *et al.*, "Xrootd, disk-based, caching proxy for optimization of data access, data placement and data replication," in *Journal of Physics: Conference Series*, vol. 513, no. 4. IOP Publishing, 2014, p. 042044.

[13] B. Cohen, "The bittorrent protocol specification," 2008.

[14] B. Wei, G. Fedak, and F. Cappello, "Scheduling independent tasks sharing large data distributed with bittorrent," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, 2005, pp. 219–226.

[15] ——, "Towards efficient data distribution on computational desktop grids with bittorrent," *Future Generation Computer Systems*, vol. 23, no. 8, pp. 983–989, 2007.

[16] C. Briquet, X. Dalem, S. Jodogne, and P.-A. de Marneffe, "Scheduling data-intensive bags of tasks in p2p grids with bittorrent-enabled data distribution," in *Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks*. ACM, 2007, pp. 39–48.

[17] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*. IEEE, 1998, pp. 140–146.

[18] J. Dinger and O. P. Waldhorst, "Decentralized bootstrapping of p2p systems: a practical view," in *NETWORKING 2009*. Springer, 2009, pp. 703–715.

[19] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1. ACM, 2007, pp. 301–312.

[20] G. Coulouris. (2015) Blast benchmark. [Online]. Available: http://fiehnlab.ucdavis.edu/staff/kind/Collector/Benchmark/Blast_Benchmark

[21] D. Weitzel, "Campus grids: A framework to facilitate resource sharing," Master's thesis, University of Nebraska, 2011.

A Queueing Model of Hybrid Parallel Pipelines PDPTA'15

Fahad Khalid*, Lena Herscheid, Andreas Polze
Hasso Plattner Institute for Software Systems Engineering
14482 Potsdam, Germany
fahad.khalid, lena.herscheid, andreas.polze@hpi.uni-potsdam.de

Abstract— *Hybrid parallel pipelines are suited for many scientific algorithms, which can be sped up significantly by using CPU and GPU hardware. However, the complexity of modern applications and hardware makes it hard to optimally configure hybrid parallel pipelines. To this end, analytical modeling approaches are needed.*

We present a novel Queuing Theory model of hybrid parallel pipelines, which can be used for performance prediction and optimization. Based on this model, we propose an algorithm for automatically identifying the bottleneck stage and tuning its degree of parallelism. Empirical evidence from various pipeline configurations demonstrates the validity of our model.

Keywords: parallel pipeline, optimization, hybrid architectures, queueing theory, queueing model

1. Introduction

Emerging hybrid parallel architectures (including both CPUs and accelerators such as *graphics processing unit*s (GPUs)) make huge performance gains possible, if the hardware is used efficiently and idle times are avoided. However, the problem of partitioning work cleverly over different processors while also keeping data access and transfer latencies low, is a hard one to solve.

In this context, *pipeline parallelism* is a pattern suited to many scientific applications. The computation on data items is performed in a sequence of pipeline *stages*. Each pipeline stage can be parallelized internally, by exploiting data or task parallelism. Since the different stages can operate on different items independently of each other, computation can be overlapped, leading to performance gains.

Within software pipelines, the question arises of how compute resources should be distributed across pipeline stages, and how the issues of load imbalance and data transfer bottlenecks should be addressed. The tuning of such parameters is a complex task depending on various system parameters. To this end, analytical models enable predicting the performance of different pipeline configurations.

Queueing Theory [1] is a mathematical modeling tool for processes, where pending items in *queues* are processed by *servers*, whose throughput is expressed as stochastic distributions. A *queueing network* consists of different servers, linked together by their input and output queues. Queueing networks allow for the computation of the overall throughput, utilization metrics and the expected number of waiting items. They are therefore a powerful tool also for predicting the performance of parallel pipelines. We present a Queueing Theory model of hybrid parallel pipelines and use it to guide performance optimizations. In particular, we show how GPU stages can be represented realistically in such a queueing network. Our model can be employed to tune the assignment of threads to pipeline stages.

2. Related Work and Our Contribution

Foremost, our work builds upon that of Navarro et al. [2], [3]. In their research, a queueing network model is presented for multithreaded CPU-only applications. This queueing model, which consists of an exponentially distributed queue for each pipeline stage, is then used to optimize the number of tokens (i.e., work items) in the system and the number of threads assigned to each pipeline stage.

Analytical Performance Models for software pipelines: Gonzalez et al. [4] propose an analytical model for software pipelines which can be used to optimally assign pipeline stages to a ring of processors.

Wei et al. [5] analytically minimize the communication overhead in accelerator-based architectures using *integer linear programming* (ILP). This minimization problem has also been tackled by model checking data flow graphs [6].

Analytical Performance Models for GPUs: The performance of hybrid parallel architectures is hard to predict, as it depends on multiple platform and application dependent parameters. There have been multiple attempts at analytically characterizing the performance of such hybrid architectures.

The *GPURoofline* model [7], as a GPU-focussed extension of the roofline model [8], enables the prediction of a theoretical upper bound on kernel performance, and the evaluation of optimizations, which improve the communication to computation ratio. The *boathull* model [9] builds upon GPURoofline, additionally including host to accelerator data transfer. Algorithms are classified by the number of parallel work units, the amount of computation,

data I/O, and memory access patterns. Prior to the actual development of kernel source code, the boathull model can be used to make performance predictions.

GROPHECY [10] is a framework for predicting GPU performance by means of CPU code skeletons. Several parameters regarding the code layout (the number of distinct code blocks, and tasks assigned to each thread) and regarding the workload (the number of memory accesses and instructions per thread), are used to project GPU performance without having to write actual GPU code.

Low-level models, which are based on the analysis of kernel source code or even GPU assembly, have also been discussed [11], [12], [13].

Optimizations for Software Pipelining: Pipeline parallelism has been identified as a commonly occurring pattern [14], lending itself to performance optimizations by overlapping different pipeline stages. Software Pipelines exhibit a special distribution of workload, and are characterized with special benchmarks such as `ferret` and `dedup` [15]. Various approaches to auto-tuning software pipelines have been discussed recently.

On-the-fly pipeline parallelism [16] determines the pipeline structure dynamically during runtime. Integrated into a Cilk-based work stealing system, this approach has demonstrated good performance in pipeline benchmarks.

Feedback-directed pipeline parallelism [17] is the auto-tuning of core-to-stage allocation by repeatedly finding the slowest pipeline stage and gradually providing it with more resources, until no more performance is gained. This hill climbing approach, has been shown to improve performance in CPU-only software. It can be integrated into software libraries such as TBB.

Load-balanced pipeline parallelism [18] tries to extract fine-grained pipeline parallelism from loops by analyzing the program graph. Different types of intra- and cross-iteration dependencies are identified and barriers for serial stages are inserted automatically. The approach is implemented in a compiler and does not yet tackle GPU architectures.

Parallel-Stage Decoupled Software Pipelining (PS-DSP) [19], an extension of *Decoupled Software Pipelining* (DSWP) [20], is the extraction of pipeline parallelism from loops in the presence of loop-carried dependencies, for compiler-based automatic code transformations. A more recent extension to this work is *speculative DSP* [21], which speculatively ignores infrequent dependencies.

Van Der Wijngaart et al. [22] have shown how fine grained software pipelines (at the instruction level) can be modeled and optimized analytically.

Stream Programming Models: Another vast body of related work studies how streams of data can be modified efficiently using a sequence of compute kernels. The focus of stream programming has recently shifted from multimedia applications to other data-intensive computations. It has been acknowledged that the stream programming style lends itself to pipeline parallel optimizations [23]. A catalogue of stream processing optimizations is presented in [24].

Thies et al. [25] presented an annotation-based tool for C programs, which facilitates the representation of coarse grained software pipelines. Based on Valgrind and a custom runtime for the resulting pipeline application, this approach has proven helpful for various streaming applications.

There have been multiple approaches to mapping streaming programs, often written in a dedicated language, to multicore architectures [26], [27]. An extension of the *OpenMP* standard, which incorporates language constructs for expressing streams and pipeline parallelism, has been proposed [28]. Recently, GPUs have also become a target platform for the execution of streaming applications [29].

2.1 Research Gap and Our Contribution

Past research has been done on predictive analytical models, as well as on optimization approaches for certain architectures, often using runtime information. Our contribution lies in the intersection of these two research areas. We first present an analytical Queueing Theory model of pipeline parallelism, which can predict the throughput of different pipeline configurations. On the basis of this model, we also demonstrate how to optimize away the bottleneck stage of a pipeline automatically. The predictions from our model can be used to avoid the time-consuming effort of manually trying out numerous pipeline configurations.

In contrast to previous research, in particular that of Navarro et al. [2], our model describes hybrid GPU-CPU systems, where pipeline stages can run either on accelerator hardware, or on the host. Thus, we contribute a novel way of modeling hybrid parallel pipelines both for performance prediction, and for optimization.

3. The Concept of Hybrid Pipeline

We define a hybrid parallel pipeline as an implementation of the parallel pipeline pattern in such a way that the at least one pipeline stage is executed on a conventional CPU, and at least one other pipeline stage is executed on an accelerator architecture. In this paper, we assume the accelerator architecture to be a Graphics Processing Unit (GPU). Throughout the rest of the paper, we will refer to the CPU as *Host*, and the GPU as *Device*.

We further assume that *Device* memory is separate from the *Host* memory. Therefore, in order for the *Device* to process data, the data must first be transferred from the *Host* to the *Device*, and finally the results transferred from the *Device* to the *Host*.

The total computation time can be reduced by overlapping kernel execution and data transfer. This is achieved by

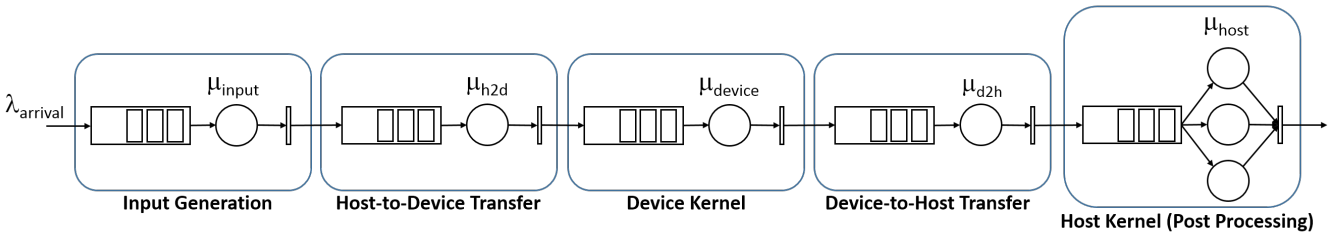


Fig. 1: A typical hybrid pipeline. Stages include data transfer to and from the *Device*, a computation kernel on the *Device*, and a Post Processing and Input stage on the *Host*. Each stage can be modeled as an $M/M/c$ queue, where $c = 1$ for *Device* and serial *Host* stages. The service rates of the different stages μ_i determine the overall throughput and bottleneck stage.

using asynchronous (non-blocking) routines for data transfer between *Host* and *Device*.

Figure 1 depicts a typical hybrid parallel pipeline. Following is a description of the stages:

1) *Input*: The purpose of this stage is to generate input data that is processed by the following stages. This stage can either comprise reading input files, or pre-processing already read data. Here we assume that this stage is executed on the *Host*, and that it is possible to utilize more than one *Device* threads to process this stage.
2) *H2D*: This stage represents the asynchronous transfer of input data from *Host* to *Device*. We assume that this stage can only be processed by one *Host* thread, since there is only one channel available for transfer of data from *Host* to *Device*.
3) *Device Kernel*: This stage represents the *Device* kernel. We assume that the *Device* cannot process more than one kernel at a time, and therefore only requires one *Host* thread. In this case, parallelism is within the *Device* kernel; multiple *Device* kernels cannot be executed in parallel.
4) *D2H*: This stage represents the asynchronous transfer of result data from *Device* to *Host*. We assume that this stage can only be processed by one *Host* thread, since there is only one channel available for transfer of data from *Device* to *Host*.
5) *Post Processing*: This stage represents a *Host* kernel that is used to post-process results generated by the *Device*. We assume that it is possible to utilize multiple *Host* threads to process this stage.

3.1 Item flow through the pipeline

Here we use the above mentioned description of pipeline stages to present a hypothetical scenario for the flow of items through the pipeline.

1) The *Input* stage generates multiple items; one item is generated per *Host* thread used for this stage.
2) Items generated by the *Input* stage are queued in the *H2D* stage. This stage transfers input data from *Host* to *Device* for one item at a time in a *first come first served* manner.
3) Items forwarded by the *H2D* stage are queued in the *Device Kernel* stage. This stage executes the *Device* kernel for one item at a time in a *first come first served* manner.
4) Output items generated by the *Device Kernel* stage are queued in the *D2H* stage. This stage copies output data from *Device* to *Host* for one item at a time in a *first come first served* manner.
5) The *Post Processing* stage can use multiple threads to process multiple output items in parallel, queued by the *D2H* stage.

4. Analytical Model of Hybrid Parallel Pipelines

We use Queuing Theory to model a hybrid parallel pipeline as a network of queues. In this Section, we first define the relevant parameters of the queuing model, and then map these to the hybrid parallel pipeline example discussed earlier.

The following parameters are used for each queue in the model.

- $T_{arrival}$ = inter-arrival time, i.e., time duration between the arrival of two successive items in the queue
- $T_{service}$ = service time, i.e., time it takes for the service to process one item
- c = number of servers

The above mentioned definitions can be used to derive the following quantities:

- *Arrival rate*, defined as item arrival per unit time,

$$\lambda = \frac{1}{T_{arrival}} \tag{1}$$

- *Service rate*, defined as items serviced per unit time,

$$\mu = \frac{1}{T_{service}} \tag{2}$$

For each queue, we assume exponential distributions for both $T_{arrival}$ and $T_{service}$. This assumption implies that the service and arrival rates are, at some point, in a steady state

and do not vary much. Such a queue can be represented as a $M/M/c$ queue in Kendall's notation. Each stage in the pipeline of Figure 1 is a $M/M/c$ queue. For *H2D*, *Device Kernel*, and *D2H* stages, we assume $c = 1$. For *Input* and *Post Processing* stages, $c \geq 1$. The complete pipeline is modeled as a network of $M/M/c$ queues.

Let λ_i be the arrival rate for stage i, and μ_i be the service rate for stage i. Then,

$$\lambda_i = \mu_{i-1} \tag{3}$$

i.e., arrival rate at stage i is equal to the service rate at stage $i-1$.

We define throughput of the stage as,

$$\frac{number\ of\ processed\ items}{time} = service\ rate = \mu \tag{4}$$

Then, given that service rate for the slowest stage defines the upper bound on the overall pipeline throughput, we define the overall pipeline throughput as,

$$X = min\,(M) \tag{5}$$

where, $M = \{\mu_s, \forall s \in S\}$, and S is the set of all stages.

In order to compare the predictions of the model with experiment results, we define a quantity that is derived from the above mentioned parameters, but can also be measured directly. This metric is time, i.e., the total time taken by the pipeline. Using Little's Law, it is defined as,

$$T = \frac{n}{X} \tag{6}$$

where n is the total number of items processed by the pipeline.

4.1 Optimizing Pipeline Parameters

In the above mentioned model, arrival times and service times are measured using serial execution of the pipeline stages. Then, a degree of freedom exposed in the model that can be used to optimize total throughout, is the number of threads per parallel *Host* stage, i.e., c.

As mentioned earlier, the total throughput of a pipeline is constrained by the throughput of the slowest stage. Therefore, we can optimize total throughput using c if and only if the slowest stage is a *Host* processing stage which can be modeled as a queue with multiple servers. *Input* and *Post Processing* stages of Figure 1 fit this criterion.

Let us call the slowest stage in the pipeline the *bottleneck stage*. For simplicity, we assume that only one of the stages can be the bottleneck stage. Then, we can identify the bottleneck stage as,

$$S_{bottleneck} = \{S_i \in S | \mu_{s_i} = min\,(M)\} \tag{7}$$

Now, let us assume that $S_{bottleneck}$ is a *Host* processing stage that satisfies the condition: $c \geq 1$. Then let us define a new parameter called *traffic intensity* as,

$$\rho = \frac{\lambda}{c\mu} \tag{8}$$

Using the *stability condition* for a $M/M/c$ queue, we have,

$$\frac{\lambda}{c\mu} < 1 \tag{9}$$

This implies that if $c = 1$, and $\frac{\lambda}{\mu} > 1$, the optimal number of threads for $S_{bottleneck}$,

$$c_{bottleneck} \geq \frac{\lambda}{\mu} \tag{10}$$

This provides us with the lower bound on $c_{bottleneck}$. In order to compute the optimal value, we can simply measure the value for c such that,

$$\frac{\lambda}{\mu} \leq c \leq c_{max} \tag{11}$$

where, c_{max} is the maximum number of threads available for $S_{bottleneck}$.

The number of *live items*, n_{live}, is another variable that affects pipeline throughout. It can be defined as the number of items that can be processed simultaneously over the entire pipeline. In our model, we assume $n_{live} = max\{c, N\}$, where N is total number of pipeline stages.

Algorithm 1 summarizes the process of finding the optimal number of servers per stage.

Algorithm 1: Algorithm for finding the optimal number of threads per server to maximize pipeline throughput. $c_{min\ \mu}$ is the c corresponding to the lowest μ determined within the foreach loop.

Input : P: Set of parallel stages
M: Throughputs of all pipeline stages
Output: Optimal value of c for bottleneck stage

```
S_bottleneck = {S_i ∈ S | μ_{s_i} = min(M)};
if S_bottleneck ∈ P then
    c_start = λ/μ;
    foreach c: from c_start to c_max do
        Store μ corresponding to c;
    end
    c_optimal = c_{min μ};
end
```

5. Empirical Analysis

In this Section, we use two different hybrid parallel pipelines to evaluate the optimization algorithm presented earlier. We begin by providing essential information about the test environment, and proceed with descriptions of the two use cases. We compare the results of executing the pipelines with the results predicted by the queuing model.

5.1 Test Environment

The empirical measurements discussed in this paper were conducted on an Intel Nehalem EX architecture based quad-core Xeon E5520 CPU with 4 cores, each of which supports 2 hardware threads. The machine contains 17GB of RAM. The installed operating system is Ubuntu 12.04 LTS. As an additional device, the machine contains an NVIDIA GTX 680 GPU with 4GB of device memory, which supports compute capability 3.0. All tests were run using CUDA driver version 5.5. Our tested code can also be used with CUDA 5.0, which is the earliest version supporting callback functionality. The compilers we used are GCC version 4.6.3 and NVCC version 5.5.

5.2 Use Case I – Toy Pipeline

As the first use case, we use a $5-stage$ toy pipeline. The purpose of this use case is to be able to experiment with different pipeline configurations, in order to thoroughly test the queuing model. Real life examples tend to be much less flexible, since the purpose of such codes is to solve specific problems. Our toy pipeline framework, however, makes it possible for us to test the following different scenarios: 1) *Post Processing* constitutes the bottleneck stage, 2) *Input* constitutes the bottleneck stage, and 3) *Device Kernel* eventually becomes the bottleneck stage.

Stages in the toy pipeline are based on the concepts described in Section 3. The implementation allows us to run the pipeline with different λ and μ values, configurable for each stage. It is also possible to configure c for the *Input* and *Post Proccessing* stages. The pipeline has been implemented using the Hybrid Pipeline Framework (HyPi) [30].

5.2.1 Results

Table 1 presents results for three test scenarios. The topmost entry shows a setup where the *Post Proccessing* stage comprises the pipeline bottleneck. We see that the queuing model correctly identifies the bottleneck stage. We further observe that the optimization algorithm correctly identifies the optimal value of c for the bottleneck stage. Similar results are presented in the second table entry, a scenario where the *Input* stage constitutes the bottleneck.

Table 1: Comparison of $S_{bottleneck}$ and $c_{optimal}$ values predicted by the model, with values measured from the simulation runs. Each row indicates one of the scenarios mentioned in Section 5.2.1. The predicted values are computed as floating point values, which are rounded up to the nearest integer.

$S_{bottleneck}$		$c_{optimal}$	
Predicted	Measured	Predicted	Measured
PP	PP	$\lceil 1.43 \rceil = 2$	2
Input	Input	$\lceil 1.35 \rceil = 2$	2
PP, Input	PP, Input	$\lceil 1.35 \rceil$, $\lceil 3.78 \rceil$	2, 4

The last entry in Table 1 presents results for a scenario where we first optimize c for the *Post Proccessing* stage, which shifts the bottleneck to the *Input* stage. We then optimize c for the *Input* stage, which results in the *Device Kernel* forming the final bottleneck stage. This scenario runs the optimization algorithm multiple times, until there is no further opportunity for optimization.

5.3 Use Case II – Combinatorial Candidate Generation

In this Section, we present results of applying the queuing model to a real simulation from the domain of Computational Biology. *Combinatorial Candidate Generation* is the most compute intensive part of the Nullspace [31] algorithm for enumerating elementary flux modes in metabolic networks. It has been shown in previous work [32] that using a hybrid parallel pipeline is an effective strategy to improve the performance of *Combinatorial Candidate Generation*.

A metabolic network comprises *metabolites* – chemical compounds – and *reactions*. A path in the network consists of one or more *substrate* metabolites being converted into one or more *product* metabolites. Such a system can be modeled as a node-weighted directed hypergraph, where nodes represent metabolites and edges represent reactions. The Nullspace [31] algorithm is used to enumerate all Elementary Flux Modes (EFMs) in the network, where an EFM is a minimal subnetwork that operates at equilibrium. For the sake of brevity, we refer the reader to [33] for a detailed description of the Nullspace algorithm. Here, we only present a brief description of how the combinatorial candidate generation algorithm is mapped on to a hybrid parallel pipeline.

Following is a description of the $3-stage$ hybrid parallel pipeline used for the combinatorial candidate generation algorithm:

1) *Generate*: This stage is implemented as a *Device* kernel. The kernel implements a combinatorial algorithm, which takes as input two bit-matrices. A bitwise-OR operation is performed on all possible combinations of columns of the two matrices. On each vector resulting from the bitwise-OR operation, a *popcount* operation is performed. If the *popcount* value is greater than a certain predefined threshold τ, a result bit corresponding to the combination vector is set in the result vector.
2) *D2H*: Due to the combinatorial nature of the *Device* kernel, the result vector can be very large, even for small input matrices. This stage is responsible for the asynchronous transfer of the result vector from *Device* to *Host*.
3) *Map*: This stage is executed on the *Host*. It parses the result bit-vector generated by the *Device* kernel, and maps all set bits to the respective column indices. This is a memory-intensive operation with low arithmetic intensity, and is therefore implemented as a *Host* stage.

5.3.1 Results

As mentioned in Section 3, the *Device Kernel* and *D2H* stages are implemented as serial stages due to the inherently serial nature of the stages. This holds true for the *Generate* and *D2H* stages in this pipeline. Therefore, *Map* is the only stage with the possibility to optimize the number of parallel threads, c.

Results of the simulation with different c values are plotted in Figure 2. We observe that the queuing model accurately predicts the pipeline performance. Also, $c_{optimal}$ calculated using Algorithm 1 is in agreement with the simulation results.

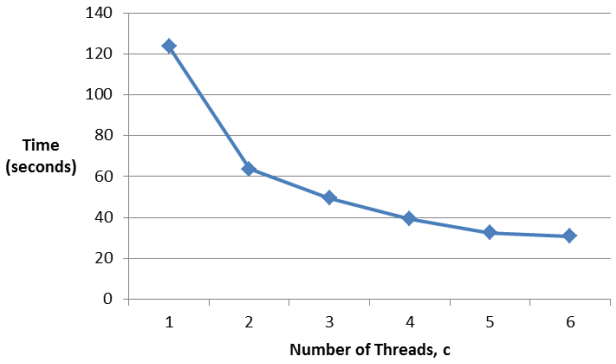


Fig. 2: Measured times for the Combinatorial Candidate Generation pipeline against different values of c. The model predicted $c = 5.37$ for the *Map* stage, which we round up to 6. Simulation runs confirm that $c = 6$ results in optimal performance.

A particularly interesting aspect of this simulation is the fact that the *Generate* stage automatically partitions the result vector according to the available *Device* memory. This makes it possible to run simulations for arbitrarily large datasets, since partitions are processed one at a time. A computed partition is transferred from *Device* to *Host*, which leads to the processing of the next partition. The maximum partition size is always kept lower than the available *Device* memory size.

Common wisdom would lead us to believe that partition size should be kept as large as possible in order to improve the *Device* to *Host* data transfer. Figure 3 shows the performance impact of different partition sizes on simulation performance. It can be observed that the performance degrades only for a very large number of partitions. Up to a certain small number of partitions, the performance does not vary significantly with the varying number of partitions.

The above mentioned result is very important for pipeline performance. If we use a single large partition, it can only be consumed by one *Map* server. In order to exploit $c > 1$ in the *Map* stage, it is essential that we generate multiple smaller partitions so that these can be consumed in parallel.

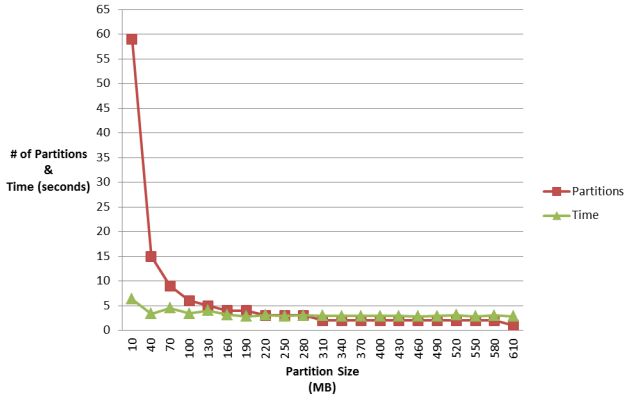


Fig. 3: Correlation between number of partitions and total simulation time. The $x-$axis depicts increasing partitions sizes. As partitions size increases, the number of partitions decreases. We observe that number of partitions only affects performance significantly for a high number of partitions. Afterwards, the performance stabilizes.

Note that this holds true only if the following stage supports $c > 1$. In the toy pipeline presented in Section 5.2, the *Input* stage would not benefit from this strategy, since the stage that follows is inherently serial.

6. Conclusion

Summary: We have presented an approach to mathematically model hybrid parallel pipelines using Queuing Theory. The model is complemented by an algorithm that can be used to identify the bottleneck stage, as well as the optimal number of threads to use for *Host* processing stages that can process items in parallel. We have provided empirical evidence to support our claims.

Discussion: Throughout the paper, we have assumed that a *Device Kernel* stage is executed on a single *Device* such as GPU. Here we propose that the model can be extended to describe multi-GPU systems as well. This would require *H2D*, *D2H*, and *Device Kernel* stages with $c =$ number of GPUs.

The model is also designed to assume a one-to-one mapping between software and hardware threads. This is due to the fact that the performance of an oversubscribed system is dependent on scheduling algorithms used by the pipeline framework and the operating system. This means that it is not possible to present results for such systems without loss of generality. Therefore, we decided not to include results of modeling oversubscribed systems.

In the future, we would like to investigate the possibility of utilizing further insights from Queuing Theory to improve the design and performance of hybrid parallel pipelines. Moreover, it would be interesting to extend the concept of

Device to other types of accelerators, such as the Intel Xeon Phi co-processor. At this point, we do not know whether the model presented in this paper can be extended to other *Devices*.

References

[1] R. Jain, "The art of computer system performance analysis: techniques for experimental design, measurement, simulation and modeling," *New York: John Willey*, 1991.

[2] A. Navarro, R. Asenjo, S. Tabik, and C. Cascaval, "Analytical modeling of pipeline parallelism," in *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '09, 2009, pp. 281–290.

[3] A. Navarro, R. Asenjo, S. Tabik, and C. Çaşcaval, "Load balancing using work-stealing for pipeline parallelism in emerging applications," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09, 2009, pp. 517–518.

[4] D. González, F. Almeida, L. Moreno, and C. Rodríguez, "Towards the automatic optimal mapping of pipeline algorithms," *Parallel Comput.*, vol. 29, no. 2, pp. 241–254, Feb. 2003.

[5] H. Wei, J. Yu, H. Yu, and G. R. Gao, "Minimizing communication in rate-optimal software pipelining for stream programs," in *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '10, 2010, pp. 210–217.

[6] A. Malik and D. Gregg, "Orchestrating stream graphs using model checking," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 3, pp. 19:1–19:25, Sept. 2008.

[7] H. Jia, Y. Zhang, G. Long, J. Xu, S. Yan, and Y. Li, "GPURoofline: A model for guiding performance optimizations on GPUs," in *Proceedings of the 18th International Conference on Parallel Processing*, ser. Euro-Par'12, 2012, pp. 920–932.

[8] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[9] C. Nugteren and H. Corporaal, "The boat hull model: Enabling performance prediction for parallel computing prior to code development," in *Proceedings of the 9th Conference on Computing Frontiers*, ser. CF '12, 2012, pp. 203–212.

[10] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram, "Grophecy: Gpu performance projection from cpu code skeletons," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011, pp. 14:1–14:11.

[11] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, ser. HPCA '11, 2011, pp. 382–393.

[12] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 152–163, June 2009.

[13] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-m. W. Hwu, "An adaptive performance modeling tool for GPU architectures," *SIGPLAN Not.*, vol. 45, no. 5, pp. 105–114, Jan. 2010.

[14] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, Eds., *Sourcebook of Parallel Computing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[15] in *Computer Architecture*, ser. Lecture Notes in Computer Science, A. Varbanescu, A. Molnos, and R. van Nieuwpoort, Eds., 2012, vol. 6161.

[16] I. Lee, T. Angelina, C. E. Leiserson, T. B. Schardl, J. Sukha, and Z. Zhang, "On-the-fly pipeline parallelism," in *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2013, pp. 140–151.

[17] M. A. Suleman, M. K. Qureshi, Khubaib, and Y. N. Patt, "Feedback-directed pipeline parallelism," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '10, 2010, pp. 147–156.

[18] M. Kamruzzaman, S. Swanson, and D. M. Tullsen, "Load-balanced pipeline parallelism," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13, 2013, pp. 14:1–14:12.

[19] E. Raman, G. Ottoni, A. Raman, M. J. Bridges, and D. I. August, "Parallel-stage decoupled software pipelining," in *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '08, 2008, pp. 114–123.

[20] R. Rangan, N. Vachharajani, M. Vachharajani, and D. I. August, "Decoupled software pipelining with the synchronization array," in *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '04, 2004, pp. 177–188.

[21] N. Vachharajani, R. Rangan, E. Raman, M. J. Bridges, G. Ottoni, and D. I. August, "Speculative decoupled software pipelining," in *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*. IEEE Computer Society, 2007, pp. 49–59.

[22] R. F. Van der Wijngaart, S. R. Sarukkai, and P. Mehra, "Analysis and optimization of software pipeline performance on MIMD parallel computers," *Journal of Parallel and Distributed Computing*, vol. 38, no. 1, pp. 37–50, 1996.

[23] M. I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," *SIGARCH Comput. Archit. News*, vol. 34, no. 5, pp. 151–162, Oct. 2006.

[24] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, "A catalog of stream processing optimizations," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 46:1–46:34, Mar. 2014.

[25] W. Thies, V. Chandrasekhar, and S. Amarasinghe, "A practical approach to exploiting coarse-grained pipeline parallelism in C programs," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 40, 2007, pp. 356–369.

[26] J. Gummaraju and M. Rosenblum, "Stream programming on general-purpose processors," in *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 38, 2005, pp. 343–354.

[27] Y. Choi, C.-H. Li, D. D. Silva, A. Bivens, and E. Schenfeld, "Adaptive task duplication using on-line bottleneck detection for streaming applications," in *Proceedings of the 9th Conference on Computing Frontiers*, ser. CF '12, 2012, pp. 163–172.

[28] A. Pop and A. Cohen, "A stream-computing extension to openmp," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, ser. HiPEAC '11, 2011, pp. 5–14.

[29] A. H. Hormati, M. Samadi, M. Woh, T. Mudge, and S. Mahlke, "Sponge: Portable stream programming on graphics engines," *SIGARCH Comput. Archit. News*, vol. 39, no. 1, pp. 381–392, Mar. 2011.

[30] F. Khalid, F. Feinbube, and A. Polze, "Hybrid CPU-GPU Pipeline Framework PDPTA'14."

[31] C. Wagner, "Nullspace approach to determine the elementary modes of chemical reaction systems," *The Journal of Physical Chemistry B*, vol. 108, no. 7, pp. 2425–2431, 2004.

[32] F. Khalid, Z. Nikoloski, P. Tröger, and A. Polze, "Heterogeneous combinatorial candidate generation," in *Euro-Par 2013 Parallel Processing*. Springer, 2013, pp. 751–762.

[33] D. Jevremovic, C. T. Trinh, F. Srienc, and D. Boley, "On algebraic properties of extreme pathways in metabolic networks," *Journal of Computational Biology*, vol. 17, no. 2, pp. 107–119, 2010.

An FPGA Architecture for Text Search Using a Wavelet-Tree-Based Succinct-Data-Structure

Hasitha Muthumala Waidyasooriya, Daisuke Ono, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {hasitha, ono1831, hariyama, kameyama}@ecei.tohoku.ac.jp

Abstract— *Succinct data structures are introduced to efficiently solve a given problem while representing the data using as little space as possible. The full potential of the succinct data structures have not been utilized in the software-based implementations. This paper discusses an FPGA-based hardware architecture for text search that uses succinct data structures. We proposes a hardware-oriented data structure and its decoding method. The proposed architecture can be used in text searches using up to 4.3GB large text files.*

Keywords: Succinct data structures, text-search, FPGA.

1. Introduction

Succinct data structures [1] are introduced to efficiently solve a given problem while representing the data using as little space as possible. Such data structures are used in many fields such as bio-informatics, text processing, etc. To solve the problem efficiently, the original data are usually pre-processed. If the original data contain n bits, "a little space" means that the storage space of the pre-processed data must be in the order of n (O(n)). To efficiently solve the problem, the processing time must be in the order of 1 (O(1)). That is, the processing time does not depend on the input data size.

Although the data storage size is in the order of n, the actual storage size is $k \times n$, where k usually takes a value from tens to thousands. As a result, the storage size of the succinct data structure is many times larger than the original data size. However, recent computers have a very large memory capacity and extremely large hard disk space. Therefore, implementing such data structures is possible and some of those implementations have given reasonably good results. However, they have many limitations so that the full potential of the succinct data structures have not been utilized. The main problem is the memory access bottleneck. Although the processing time is independent of the data size, the memory access is unpredictable and requires many clock cycles. Moreover, the data are usually in a compressed or encoded state, so that a decompression or decoding overhead is required. Therefore, it is often a serious challenge to efficiently utilize the succinct data structures for massively parallel implementations.

Designing a custom hardware is a good solution to such problems. A custom hardware contains a large number of compact processing elements (PE) that are specialized to solve only the given problem. The data paths between the PEs and the memory can be designed to efficiently use the full memory bandwidth. The decompression/decoding can be done in parallel in minimum number of clock cycles. we consider an FPGA-based accelerator for text search applications. An FPGA is a reconfigurable LSI that contains millions of programmable logic gates. Recently, speed and power consumption of the FPGAs are greatly improved, and it would be very practical to use the FPGA-based platform for real applications. However, the lack of huge DDR3 memories is a major problem in FPGA boards. Many high-end FPGA boards contain just 4 GB of memory capacity. Therefore, we have to compress the data as much as possible while still allowing the efficient access to the data.

To implement succinct data structures on hardware, we can not rely on the order of the computations. Even the order is small, the processing time or the storage space could be so large that the data structure may not be implemented on the hardware. In this paper, we consider the factors such as the memory bandwidth, word width of the memory, storage size etc to find a hardware-compatible succinct data structure, which could actually be implemented on the hardware. We also consider hardware-oriented data compression method to reduce the storage space further without increasing the processing time.

2. Succinct-Data-Structure

2.1 Text search using rank

In this paper, we limit the given problem to the text search. The operation $rank_q(T, x)$ returns the number of "element q"s from a text T up to the position x. The element q could be any symbol such as a number, a letter, a byte, etc, and T is an array that contain many elements. The implementation of the *rank* operation in a constant time is presented in [1], [2]. Using the *rank*, a quarry can be searched in a text with a processing time proportional to the size of the quarry and not proportional to the size of the text. That is, the search time does not increase with the size of the text.

```
Search(Q, i, k, l)
begin
    I = φ
    k = 0
    l = |X|
    for i = |Q| − 1 to i = −1 do
        if i == −1 then
            return [k, l]
        end
        k = C(Q[i]) + rank_{Q[i]}(B, k − 1)
        l = C(Q[i]) + rank_{Q[i]}(B, l) − 1
                    //B is the BWT string of X
        if k ≤ l then
            i = i − 1
        else
            return φ //return empty
        end
    end
end
```

Algorithm 1: Text search algorithm

The text search method is shown in algorithm 1. In this algorithm, the search quarry Q is searched in the text X. The number of elements in X and Q are given by $|X|$ and $|Q|$ respectively. The text X is pre-processed to construct the *rank* table and the array $C(.)$. We explain the pre-processing using an example in Fig.1. The text X is shown in Fig.1(a) where the end of the text is identified by "$". As shown in Fig.1(b), the text is shifted to the left until all the symbols are moved. The shifted (rotated) text is sorted in lexicographical order as shown in Fig.1(c). The suffix array (SA) in Fig.1(c) shows the sorted array of all the suffixes. This rotation and sorting is also called the Burrows-Wheeler transform (or BW transform) [3] and the string in the last column of Fig.1(c) is called the "BWT string" and denoted by B. Then we count the number symbols from the beginning to each index and put those values on a table. This "*rank*" table is shown in Fig.1(d). For example, $rank_E(B, 3) = 1$, since there are only one "E" appears from the index 0 to 3 in the *rank* table. The number of symbols that are lexicographically smaller than a is given by $C(a)$ where $a \in B$.

Fig.2 shows the searching of the quarry (Q) in text (X). The quarry Q and $C(.)$ array are shown in Figs.2(a) and 2(b) respectively. According to [4], if a quarry q is a substring of the text X and $k(aq) \leq l(aq)$, the quarry aq is also a substring of X where aq equals the quarry $\{a, q\}$. The terms k and l, given by Eqs.(1) and (2) respectively, are the lower and upper bounds of the suffix array interval of X.

$$k(aq) = C(a) + rank_a(B, k(q) - 1) \quad (1)$$

$$l(aq) = C(a) + rank_a(B, l(q)) - 1 \quad (2)$$

We can find the position of Q in X by repeatedly applying Eqs.(1) and (2) to every symbol in Q as shown in Fig.2(c).

Position	0	1	2	3	4	5	6
Text (X)	E	E	H	G	A	G	$

(a) Text data (X)

Position	0	1	2	3	4	5	6
0	E	E	H	G	A	G	$
1	E	H	G	A	G	$	E
2	H	G	A	G	$	E	E
3	G	A	G	$	E	E	H
4	A	G	$	E	E	H	G
5	G	$	E	E	H	G	A
6	$	E	E	H	G	A	G

(b) Moving (rotation) of the text

Suffix array		0	1	2	3	4	5	6
Index	Position in X							(BWT string B)
0	6	$	E	E	H	G	A	G
1	4	A	G	$	E	E	H	G
2	0	E	E	H	G	A	G	$
3	1	E	H	G	A	G	$	E
4	5	G	$	E	E	H	G	A
5	3	G	A	G	$	E	E	H
6	2	H	G	A	G	$	E	E

(c) Sorting of text data

a	A	E	G	H
C(a)	1	2	4	6

Index	A	E	G	H
0	0	0	1	0
1	0	0	2	0
2	0	0	2	0
3	0	1	2	0
4	1	1	2	0
5	1	1	2	1
6	1	2	2	1

(d) $C(.)$ and *rank* data

Fig. 1: Pre-processing the text

The suffix array interval (SA) is [6,6] so that we can find the actual position using the suffix array in Fig.1(c). In this case, $SA[6, 6] = 2$. The search is done in 3 steps proportional to the number of symbols in the quarry Q.

2.2 Data storage and processing time

In the initial work of succinct data structures [1], a method to store the *rank* data and compute the *rank* in a constant time is proposed. Given a binary sequence $B[0, n-1]$ of size n, a two-level directory structure is built. The first level contains large blocks of size $log_2 n \times log_2 n$. For each large block, the *rank* of the first entry is stored in a separate array. This requires $n/log_2 n$ storage. Each large block is divided in to small blocks of size $log_2 n/2$. Therefore, each large block contains $2log_2 n$ number of small blocks. Within a large block, another array is used to store the rank of the first entry of all small blocks. For all large blocks, this array requires $4nlog_2(log_2 n)/log_2 n$ bits. A look-up table is used to store the answer to every possible *rank* on a bit string of size $log_2 n/2$. It requires $2^{log_2 n/2} \times log_2/2 \times log_2(log_2 n/2)$ bits. All arrays and tables can be implemented using O(n) bits, and it supports *rank* queries in a constant time. Please refer [1] for more details. Since we use many arrays and tables, this method needs multiple (although a constant number of) memory reads to compute the *rank*. Moreover, this method is proposed for bit vectors. It is not efficient to use this

Position	0	1	2
Quarry (Q)	H	G	A

(a) Search quarry (Q)

a	A	E	G	H
$C(a)$	1	2	4	6

(b) C array

$k(\,) = 0, \quad l(\,) = 6$
$k(A) = C(A) + rank_A(B, 0-1) = 1 + 0 = 1$
$l(A) = C(A) + rank_A(B, 6) - 1 = 1 + 1 - 1 = 1$

$k(GA) = C(G) + rank_G(B, 1-1) = 4 + 1 = 5$
$l(GA) = C(G) + rank_G(B, 1) - 1 = 4 + 2 - 1 = 5$

$k(HGA) = C(H) + rank_H(B, 5-1) = 6 + 0 = 6$
$l(HGA) = C(H) + rank_H(B, 5) - 1 = 6 + 1 - 1 = 6$

$SA[6,6] = 2$

(c) Searching of the quarry (Q) in text (X)

Fig. 2: Searching the quarry Q in text X

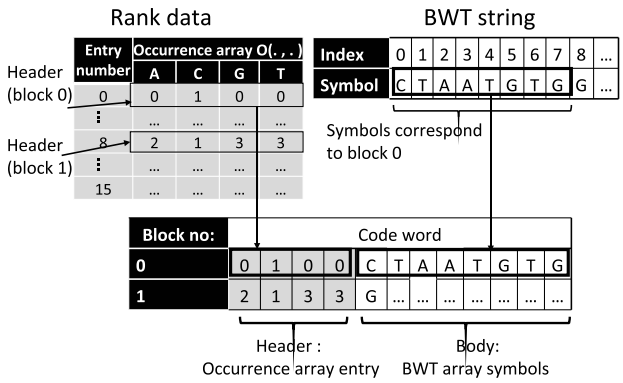


Fig. 3: *rank* data encoding of a human genome

method when the input is not a bit vector but contain multiple characters, such as general text.

A different data structure for the multi-character text search is proposed in bio-informatics applications such as short-read alignment [5]. In short-read alignment, a short DNA fragment is searched in a large genome, which is basically a text search. Genome data contains 4 symbols, "A,C,G,T" that are represented by 2 bits. Fig.3 shows a *rank* table of 16 entries. We divide the rank data table into two blocks where each block contains 8 entries. Then the first entry is chosen as the header. The rest of the entries are replaced by the BWT symbols. Since one BWT symbol has significantly smaller size compared to a *rank* entry, this method reduces the storage size. However, in the decoding, we have to count the number of symbols of each character in the body. To do this in constant time, we need a population count (popcount) hardware.

A human genome contains approximately 3 billion symbols. Therefore the rank table contain 3 billion entries where each entry has $log_2(3\text{ billion}) \times 4$ bits. That is 128 bits. Therefore, the storage space for the rank table requires 128×3 billion bits which is approximately 48GB. The above encoding method is used in [6] to successfully implement the short-read alignment using just 1.5 GB of data. In [6], the *rank* table is divided in to blocks where each block contain 64 entries. The first entry of each block is used for the header, which requires 128 bits. The rest of the entries in each block are replaced by the BWT symbols. Since 2 bits are required to represent the "A,C,G,T" symbols, the body contains only 128 bits (2×64). Therefore, one code word is 256 bits and we need 3 billion/64 of such code words. That is 1.5 GB. Moreover, 256 bits can be read in one memory read in FPGA. Note that, one memory read provides the access to a block of consecutive data. The popcount of 64 symbols can be done in a few steps in hardware and many such popcount architectures are already proposed [7].

To use this method in text search, let us consider a general case that has n symbols in the BWT string. We consider m different symbols in the alphabet. Therefore, one rank data entry requires $m \times log_2 n$ bits. Since there are n entries we need a total of $n \times m \times log_2 n$ bits. We divide the rank data into multiple blocks where each block contain p entries. In each block we store the first entry as the header. The rest of the entries in a block is replaced by the symbols in the BWT string. Therefore, the required total bits (T_{bit}) is given by Eq.3.

$$T_{bit} = (m \times log_2 n) \times \frac{n}{p} + n \times log_2 m \tag{3}$$

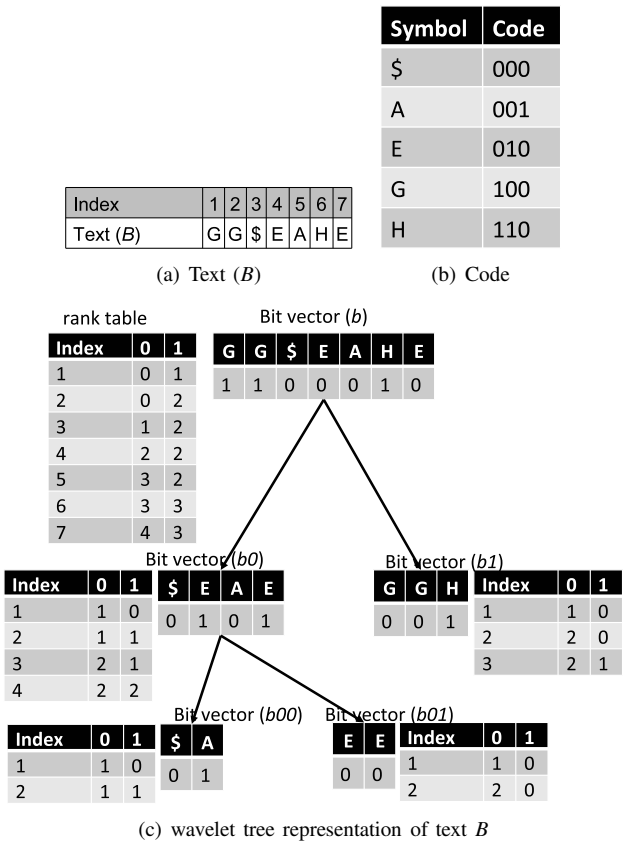
If this data structure is to be succinct, T_{bit} must be in the order of O(n). To satisfy this condition, the block size p must be greater than or equals to $log_2 n$. Moreover, the symbol count of a block must be done in a constant time irrespective of the size of n. That is, popcount(p) must be done in a constant time. Since there are popcount hardware that have constant computation time, constant processing time is achieved.

Since we use FPGA, we consider a memory size of 4GB. This condition is reasonable since many FPGA boards with high-end FPGAs contain this much of memory. Now let us calculate how much memory is required and how large is a block when we consider a 1GB input text file. We also consider each letter in the input file contain 8 bits (1 Byte) and there are 128 meaningful letters in the alphabet. Therefore, $n = 1GB/1B$ and $m = 128$. From Eq.3, when the total bits T_{bit} equals to 4GB, the block size $p = 1229$. As a result, one code word contains a header of 128×30 bits and a body of 1229×7 bits. That is 12443 bits. Therefore, if the word width of the memory is 512 bits (512 bits are accessed in one read), 25 memory reads are required to get one rank data value. After that, we have to perform the popcount function for 1229 symbols. As wee can see here, although we can store the data, accessing it and decoding it is very costly in terms of both time and area. Even a single memory access may take several cycles to complete,

25 memory reads per a *rank* data is not practical. Therefore, we need a better data structure.

2.3 Wavelet tree based data structure

As we saw above, the strategies relating to the binary sequences or small number of symbols cannot be applied directly to the data structures with many symbols. The wavelet tree proposed in [8] permits a way to compute the *rank* of an arbitrary alphabet of size m efficiently. Let us explain the construction of the wavelet tree using the example in Fig.4. For a given text B shown in Fig.4(a), a code is assigned to every symbol as shown in Fig.4(b). The construction of the wavelet tree start from the most significant bit (MSB) of the code. A bit vector b is created by using the MSB of each symbol in the text B as shown in Fig.4(c). That is, "0" is assigned to the symbols "$, A, E" and "1" is assigned to the symbols "G, H". Then we divide the bit vector in to two groups. One group contains the symbols that their corresponding bits in b are 0. The other group contains the symbols that their corresponding bits in b are 1. Then we assign bit vectors $b0$ and $b1$ for each group using the second most significant bit. This process continues until a unique bit (0 or 1) is assigned for every symbol in a group. After the construction of the wavelet tree, we create *rank* tables for each bit vectors.



(a) Text (B) (b) Code

(c) wavelet tree representation of text B

Fig. 4: Construction of a wavelet tree

Fig.5 shows how to compute *rank* using wavelet tree. In this example, $rank_E(B, 4)$ is considered. Note that, B is the text shown in Fig.4(a). The computation of *rank* is done from the top to the bottom of the wavelet tree. Since the MSB of the symbol "E" is zero, we compute $rank_0(b, 4)$. Then we come down to the second level of the wavelet tree and use the input vector $b0$, since "E" is included in the group that the MSB of "E" is zero. Then $rank_1(b0, 2)$ is calculated. Similarly, the calculation is done for all the levels in the wavelet tree as shown in Fig.5.

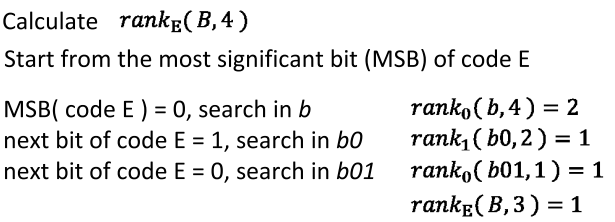


Fig. 5: Computation of $rank_E(B, 4)$ using wavelet tree

Although Fig.4(c) shows the *rank* table for the symbols "0" and "1", we just have to store the *rank* of only one symbol. The *rank* of the other symbol is derived by subtracting the *rank* of the known symbol from the index. For example, $rank_0(b, x) = x - rank_1(b, x)$ where b is the bit vector and x is the index. There are many *rank* tables in $log_2 m$ levels. However, in each level, the sum of all entries in all tables equals to the number of symbols in the reference text. Therefore, the total number of bits required ($T_{wavelet}$) is given by Eq.(4).

$$T_{wavelet} = \left\{ log_2 n \times \frac{n}{p} + n \right\} \times log_2 m \tag{4}$$

Using Eq.(4), we can obtain the block size for the example of 1GB input data. In this case, $p = 68$. Therefore, in one block we have a 30 bits large header and 68 bits large body. Therefore, a code word contains a total of 98 bits. This much of data can be accessed in one memory read. Note that, the wavelet tree has $log_2 m$ levels of bit vectors and we have to access a bit vector in each level. Therefore, in this example, a total of 7 memory reads are required to obtain one *rank*. This is a substantial reduction of the memory reads.

2.4 Proposed data structure for FPGAs

In this paper, we discuss a data structure that considers the hardware specification. We consider a memory model where each read access W bits from the memory. As shown in Fig.6, a code word consists of a header and a body. The size of the header is decided by the number of symbols n in the text. The size of the body is decided by the number of entries in a block of the *rank* table. Since the header requires $log_2 n$ bits, the body contains maximum of $W - log_2 n$ bits which should be equal to the number of entries in a block. Therefore, the block size $p = W - log_2 n$. The body size

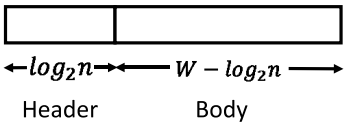


Fig. 6: A cord word of W bits long

could be further reduced by using byte-pair encoding (BPE). BPE [9] is a simple data compression method that the most common pair of consecutive bytes of data is replaced with a byte that is not been used already in the compressed data file. The same method could be applied for the bit array too and it is already used in text processing in [10]. In this case, we apply BPE for each cord word separately by using a common dictionary data. According to the experimental results using various text files. we found that 80% compression ratio could be achieved. The compression ratio is decided by the worst case. The required memory size T_{prop} is given by Eq.(5).

$$T_{prop} = \left\{ log_2 n \times \frac{0.8n}{W - log_2 n} + n \right\} \times log_2 m \qquad (5)$$

In practical cases, $W - log_2 n$ is much larger than $log_2 n$. Therefore, the storage is in the order of $O(n)$. The memory access is in the order of $O(log_2 m)$ which is independent of n. Therefore, we can say that this data structure is succinct.

3. FPGA architecture and evaluation

Fig.7 shows the overall architecture. It consists of a PE array and two DDR3 memories. The *rank* data of the text are stored in the DDR3 memory. Then the search queries are transfered to the DDR3. PEs process the search queries and find the search positions. Those data are written to a shared memory and later read by the host computer. The search queries can be sent in batches. After one batch is finished, another batch is transfered to the DDR3 memory. Therefore, we can process any number of search quarries while the quaries in a batch are processed in parallel by multiple PEs.

The structure of a PE is given in Fig.8. It consists of a 32-bit adder, a comparator and pipeline registers to perform the calculations explained in algorithm 1. The "ADD/SUB" unit in PE is used to calculate the suffix array interval given by Eqs.(1) and (2). The comparator and the control path do all the conditional branches in the "Search" procedure. New search queries are fed to the PEs after the old ones are searched. The output is read by the CPU. Unlike the CPU that has a complex floating-point ALU and very complicated control circuit, a PE is a very simple unit that specialized only to search a query. It is designed using minimum resources. Therefore, we can have a lot of PEs in the same FPGA to provide performance comparable to a computer cluster that has many CPUs.

The hardware module that decodes the *rank* data is shown in Fig.9. We extract only the required bits from the body of

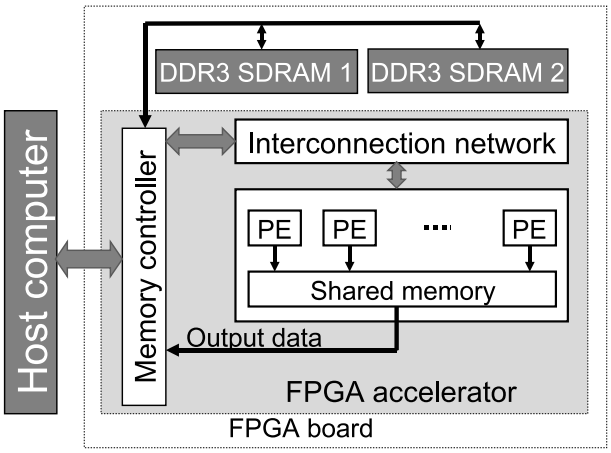


Fig. 7: Accelerator architecture

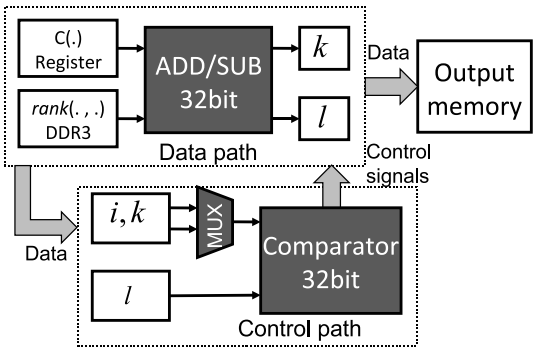


Fig. 8: Structure of a PE

a code word. For example, if we need only 16 bits starting from the LSB (least significant bit), we do the "bitwise AND" operation with a mask. In this case, the mask is 0xFFFF. Since the memory address corresponds to the *rank* entry number, the mask is obtained by decoding the memory address. After the required bits are determined, we count the number of 1's using a popcount module. Finally, the symbol count is added to the header. The decoder is pipelined, and an output is produced in every clock cycle after the pipeline is fully filled.

This decoding method has an added advantage of reducing

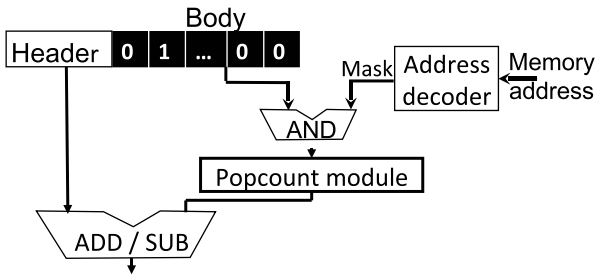


Fig. 9: Hardware decoder

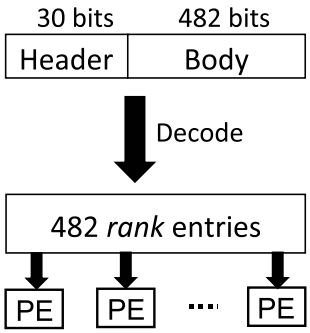


Fig. 10: Sharing of the decoded data

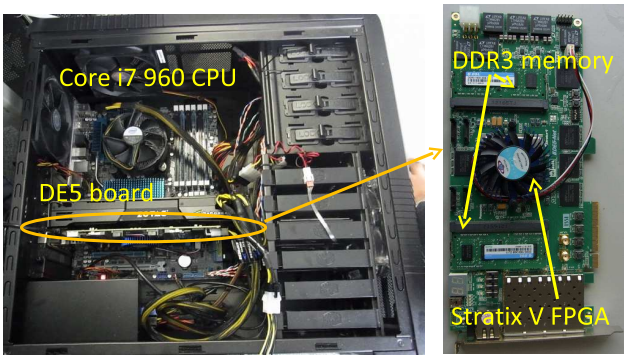


Fig. 11: Evaluation environment

the memory access. For example, let us consider the text file of 1GB large. The number of symbols in the file is given by n and $log_2 n = 30$, so that the header is 30 bits. If the word size of one memory read is 512 bits, the body contain 482 bits. Therefore, is it possible to obtain 482 *rank* data entries by decoding one code word as shown in Fig.10. Since we do parallel processing using multiple PEs, some of those *rank* data could be used in more than one PE. In such cases, the number of memory accesses are reduced.

For the evaluation, we used DE5 board [11] that contains "Altera 5SGXEA7N2F45C2 FPGA" and two 2GB DDR3-SDRAMs. The system shown in Fig.11 contains a core i7-960 CPU and a DE5 board connected through the PCI express port. The operating frequency of the accelerator is estimated to be 100MHz. We estimated that around 128 PEs can be implement on the FPGA.

Table 1 shows the size of the original text and encoded text. Usually, text data are in bytes so that the original text size is calculated by the actual file size. However, in the evaluation, we consider an alphabet of 128 characters. Therefore, one symbol requires only 7 bits. Compared to that, the FPGA implementation require a similar amount of bits with an increase of just 5.5%. In fact, the storage size is smaller than the original text file size. The reason for the small storage size is that we encode a large block of over 400 entries into a single code word. Therefore, the header size is very small. Since we use the wavelet tree representation, the header size is further reduced. However, using more bits in the body require a larger popcount function. That increases the hardware overhead. To reduce the hardware overhead, we have to reduce the number of bits in the body.

Table 1: Required data size

Original data size of the text (8bits per a symbol)	(7bits per a symbol)	Required storage size after encoding
1GB	0.88GB	0.92GB
2GB	1.75GB	1.84GB
4GB	3.50GB	3.69GB
4.3GB	3.76GB	3.97GB
5GB	4.38GB	4.61GB

4. Conclusion

This paper discusses an FPGA-based hardware architecture for text search that uses succinct data structures. We proposes a hardware-oriented data structure and its decoding method. The proposed architecture can be used in text searches up to 4.3GB large data. The storage space is just 5.5% larger than the original data size (7bits per symbol) and smaller than the input file size (1 byte per symbol).

Acknowledgment

This work is supported by MEXT KAKENHI Grant Numbers 24300013 and 15K15958.

References

[1] G. Jacobson, "Succinct static data structures. PhD thesis", Carnegie Mellon University, 1989.
[2] G. Jacobson, "Space-efficient static trees and graphs", 30th Annual Symposium on Foundations of Computer Science, pp.549-554, 1989.
[3] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm", Digital Equipment Corporation, Palo Alto, CA, Technical report 124, 1994.
[4] P. Ferragina and G. Manzini, "Opportunistic data structures with applications", Proc. of 41st Symp. on Foundations of Computer Science, pp.390-398, 2009.
[5] Heng Li and Richard Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform", Bioinfomatics, Vol.25, No.14, pp.1754-1760, 2009.
[6] H. M. Waidyasooriya, M. Hariyama and M. Kameyama, "Implementation of a custom hardware-accelerator for short-read mapping using Burrows-Wheeler alignment", Conf Proc IEEE Eng Med Biol Soc., pp.651-654, 2013.
[7] H. S. Warren, "Hacker's Delight (2^{nd} edition) - Chapter 5", 2012.
[8] R. Grossi, A. Gupta, J.S. Vitter, "High-order entropy-compressed text indexes", Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA'03), pp.641-650, 2003.
[9] Philip Gage, "A New Algorithm for Data Compression", C/C++ Users Journal, 12(2), pp23-28, 1994.
[10] H. M. Waidyasooriya, D. Ono, M. Hariyama and M. Kameyama, "Efficient Data Transfer Scheme Using Word-Pair-Encoding-Based Compression for Large-Scale Text-Data Processing", Conf Proc IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), pp.639-642, 2014.
[11] http://www.altera.com/education/univ/materials/boards/de5/unv-de5-board.html.

Parallel processing of Breadth First Search by Tightly Coupled Accelerators

Takahiro Kaneda
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
kaneda@am.ics.keio.ac.jp

Takuji Mitsuishi
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
mits@am.ics.keio.ac.jp

Yuki Katsuta
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
katsuta@am.ics.keio.ac.jp

Takuya Kuhara
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
kuhataku@am.ics.keio.ac.jp

Toshihiro Hanawa
The University of Tokyo
5-1-5 Kashiwanoha, Kashiwa,
277-8589, Japan
hanawa@cc.u-tokyo.ac.jp

Hideharu Amano
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
asap@am.ics.keio.ac.jp

Taisuke Boku
University of Tsukuba
1-1-1 Tennodai, Tsukuba,
305-8573, Japan
taisuke@cs.tsukuba.ac.jp

The Tightly Coupled Accelerators (TCA) architecture connects a number of graphics processing units(GPUs) directly through PCI express using dedicated switches called PEACH2 (PCI-Express Adaptive Communication Hub Ver.2). By making the best use of the low-latency communication supported by PEACH2, the breadth-first search (BFS) algorithm from Graph500 which requires frequent communications between GPUs, was implemented with multiple GPU systems. In using the BFS with the TCA, 1.58 times better performance was achieved than with a common implementation using MPI.

Keywords

GPU, Cluster, Tightly coupled accelerators architecture, PEACH2

1. INTRODUCTION

In recent years, due to the spread of general purpose computation using on graphics processing units(GPUs), heterogeneous clusters with multiple hosts each equipped with GPUs, have been the mainstream of high performance computing systems. Such systems are expected to be used for the recently emerging big data processing as well as for numerical computation. However, such heterogeneous clusters cause a large latency between GPUs communicating across nodes by indirect communication via the memory of host CPUs. In non-numerical computation procedures such as graph processing, small data communication is frequently required and the communication latency tends to bottleneck the performance improvement obtained by using multiple GPUs provided in heterogeneous clusters.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2015) Boston, MA, USA, June 1-2, 2015.

The Tightly Coupled Accelerators (TCA) architecture[1][2] provides direct data communication between accelerators connected to different nodes. PEACH2 (PCI Express Adaptive Communication Hub Ver.2) is a realizaton of TCA architecture implemented by field-programmable gate array(FPGA), and it uses the PCI Express(PCIe), commonly connects a host CPU and accelerators, as a network link. The hardwired logic on the FPGA of PEACH2 provides a direct memory access (DMA) controller(DMAC) that can handle PCIe transfers directly. Using PEACH2 enables a double ring network to be formed with the PCIe, which was originally designed only for a tree network with a single host CPU as a root. The memories of host CPU and attached GPUs which are connected by PEACH2 are mapped into a single address space of the PCIe, and data can be transferred by write access to the address. The current PEACH2 is implemented on Altera's Stratix IV FPGA, operates on a 250MHz clock, and the minimum latency between two GPUs is only 2.3 ms, much smaller than that using the MVAPICH2 with Infiniband.

HA-PACS/TCA is a testbed as a proof-of-concept system for TCA architecture in the University of Tsukuba's Center for Computational Science and started using it for scientific computation. However, the low latency communication it provides can be most efficiently used for non-numerical computing rather than large scale scientific computing programs, which require a higher bandwidth to cope with large block data transfers. Recently, big data computing functions such as graph analysis have come to require short, frequent messages, and these are difficult to handle with standard Infiniband connected heterogeneous clusters. In the work reported in this paper, we implemented the breadth-first search(BFS) algorithm from Graph500 on the HA-PACS/TCA and compared the performance obtaind with that obtained when using MPI over Infiniband.

Table 1: Specifications of PEACH2

FPGA family	Stratix IV GX
FPGA chip	EP4SGX530NF45C2
Process Technology hline Port	40nm PCIe Gen2 x8 * 4 port
Max bandwidth fo a port	4GB/sec
Max Frequency	250MHz
Internal bus width	128bit
On-board DRAM	DDR3 512MByte

2. COMMUNICATION USING PEACH2

2.1 PEACH2

Although PCIe has been commonly used in recent CPUs, it is designed for I/O networks with a tree structure and makes it difficult to connect multiple nodes directly. However, the PEACH2 switching hub enables PCIe to be used as a low-latency network. Figure 1 shows a block diagram of a PEACH2 chip implemented on an FPGA. There are four ports on it; port N is an endpoint for PCIe Gen2 x8 and is plugged into the PCIe connector on the host CPU board. A node is formed by connecting GPUs to the same back-plane board. Multiple nodes are connected as a ring network formed with two ports: Port E, an endpoint, and Port W, a root complex. The remaining Port S is a switchable x16 port which works as x8 port and is used to connect two ring networks, each of which can connect 8 nodes. The routing function embedded in the FPGA determines the destination port merely by checking the destination address of a PCIe packet to form a single address space for data transfers. The DMAC supports sophisticated block data transfers in the address space. Table 1 shows the details of PEACH2. It was implemented with Altera's Stratix-IV and operates on a 250MHz system clock. A 512MByte DDR3 SDRAM is provided on the board.

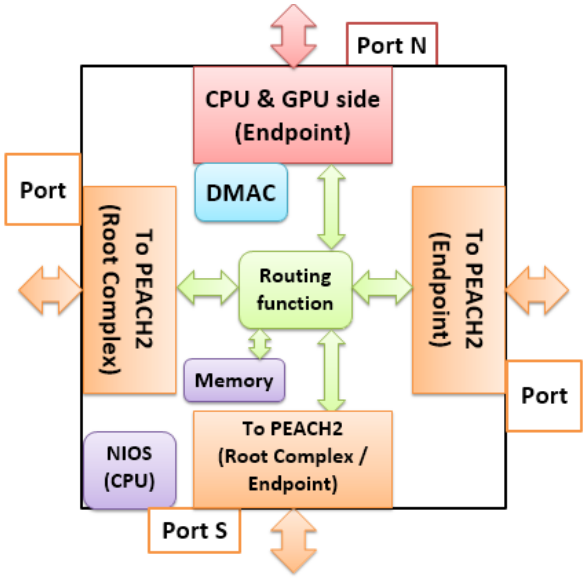


Figure 1: Block diagram of PEACH2

2.2 Communication with PEACH2

Figure 2 shows a multi-node system connected with PEACH2. Since 8 nodes form a ring by using Ports E and W, 16 nodes can be connected with Port S in total. If more nodes need to be connected, the next level network using Infiniband is required. Figure 4 shows an example of the shared address setting for 16 nodes.

The 512 GByte total address region is split, and a 32GByte address is assigned to each node. The routing function provided in PEACH2 has control registers for address mask as well as for lower bound and upper bound, and the destination port is statically determined by checking the address with the address mask. On PEACH2, memory accesses to remote nodes are restricted to memory write requests. Instead of memory read, which is difficult to implement efficiently, the proxy write mechanism can achieve the same effect by using driver support.

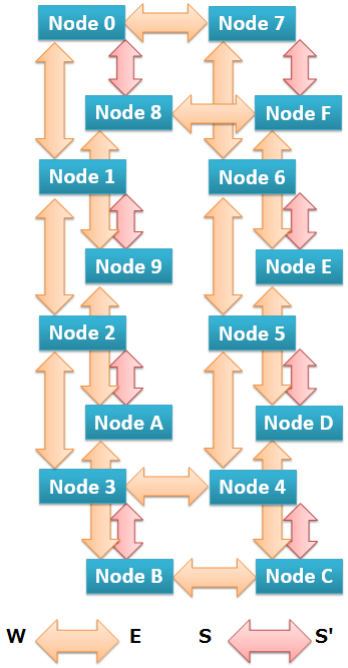


Figure 2: Multi-node system connected with PEACH2

The PEACH2 provides two types of communication: PIO and DMA. The former is useful for short message transfers, while the latter can only perform a store operation to remote nodes. In order to enable PIO communication, the PCIe region assigned to the PEACH2 is mapped through the device driver for the user space by an mmap interface. The PEACH2 DMAC supports enhanced DMA functions, including chaining using descriptors and block stride data transfer.

The TCA system specifications are shown in Table 2. The physical configuration of the two nodes in Figure 4, shows they are connected. An FPGA borad (that of PEACH2), two Ivy Bridge processors, and four of NVIDIA K20X GPUs are installed on each node. For the PEACH2 in PCIe Gen2 x8, GPUs are connected by a PCIe Gen2 x16 bus. Multiple nodes in the cluster are connected by PEACH2, and each

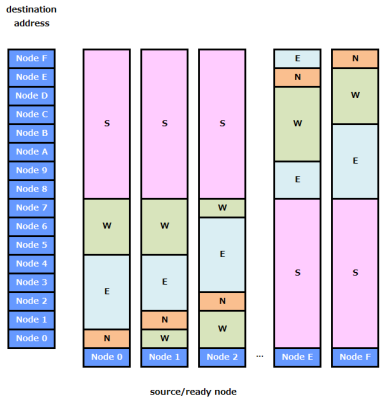


Figure 3: PCIe address spaces of PEACH2

node has a two-step configuration for connecting to a higher level network via the Infiniband standard. Figure 1 shows a block diagram of the entire system.

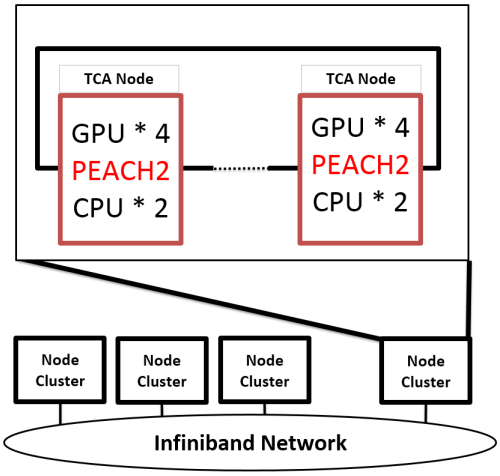


Figure 4: PCIe address spaces of PEACH2

2.3 Details of the target system

Table 3 shows the system we used in the implementation. It is an experimental cluster system as a prototype of HA-PACS/TCA in the University of Tsukuba's Center for Computational Sciences. It provides almost the same configuration as the HA-PACS/TCA available for use at the center, which in 2013 took third place in the Green500, the ranking of the most energy efficient supercomputers in the world. For performance comparison, it provides both PEACH2 and Infiniband for each node. The communication performance of PEACH2 was reported in [3]; the ping-pong latency in case of GPU-GPU is only 2.3 ms, as opposed to 6.5ms for MVAPICH2-GDR2.0 using GPUDirect for the RDMA option with Mellanox Infiniband. The maximum bandwidth between GPUs over nodes is about 2.3GBytes/sec.

3. BREADTH-FIRST SEARCH AND ITS PARALLEL PROCESSING

Table 2: Specifications of HA-PACS/TCA

CPU	Intel Xeon E5-2680 v2(Ivy Bridge-EP)
Num of Core	20 Core/Node (10 Core/Socket 2 Socket)
Clock	2.8 GHz
Peak spec	364 TFLOPS
PCI-express	Generation 3 80 Lane (40 Lane/CPU)
Memory	128 GB, DDR3 1866MHz, 4 channel/Socket, 119.4 GByte/s/Node
GPU	NVIDIA Tesla K20X
Num of GPU	4 GPU/Node
Memory	24 GByte/Node (6 GByte/GPU)
Node Cluster Connection	Infiniband QDR 2 rail (Mellanox ConnectX-3 dual head)
TCA board	Stratix IV 530 GX

Table 3: Evaluation environment

CPU	Intel(R) Xeon(R) CPU E5-2680
Clock	2.80GHz
Memory	128GB
GPU	NVIDIA Tesla K20m
Memory	5GB
PEACH2	Stratix IV EX4SGX530
OS	CentOS6.4
Host Compiler	GCC 4.4.7
CUDA	Toolkit 6.0
MPI	MVAPICH2-GDR 2.0
Library	CUB v.1.3.2

3.1 Level synchronized BFS

Breadth-first search (BSF) is an algorithm with which every vertex of a graph is visited in the breadth first order. Each vertex is labeled by the parent number or distance from the source vertex. Here, the label of each vertex is represented by the parent as in the Graph500 benchmark[4]. The target graph is represented by an adjacency matrix in a compressed sparse row (CSR) sparse matrix format. We used Level Synchronized BFS, which is a representative parallel BFS, and processed it as shown in the following pseudo-code.

A CQ holds vertices of the current depth level, while an NQ holds the vertices of the next depth level. The array *visited* holds whether a vertex has been visited or not. The search results are stored in the array *pred* as the parents identifiers. If there are no parents, -1 is held in *pred*.

3.2 Related work

A lot of research has been reported on the parallel execution of level-synchronized BFS with a single GPU or multiple GPUs. Harish et al. proposed algorithms for a single GPU[5], and we extended the method for multiple GPUs. Mastrostefano[6] extended it to multi-GPU systems and proposed a method for reducing the communication. Mitsuishi et al.[7] improved it for multi-GPU systems with poor communication capacity. Suzumura et al. implemented level synchronized BFS on a TSUBAME2.0 supercomputer at the Tokyo Institute of Technology[8]. For executing the BFS on a large scale supercomputer, they used the 2D Partitioning-Based BFS, which places processors, vertices and adjacency

Algorithm 1 Level-synchronized BFS

```
1: for all vertex v in parallel do
2:   pred[v] ← −1
3:   pred[r] ← 0
4:   Enqueue(CQ, r)
5:   while CQ! = Empty do
6:     NQ ← empty
7:     for all uinNQ in parallel do
8:       u ← Dequeue(CQ)
9:       for each v adjaccent to u in parallel do
10:        if pred[v] = −1 then
11:          pred[v] ← −1
12:          Enqueue(NQ, v)
13:        end if
14:      end for
15:    end for
16:    swap(CQ, NQ)
17:  end while
18: end for
```

matrices in a two-dimensional array to improve the performance.

4. DESIGN AND IMPLEMENTATION

In implementing the level-synchronized BFS, we adopted a standard approach for comparing TCA architecture and an MPI-connected GPU cluster.

4.1 The BFS in a multi-GPU system

Algorithm**??** for a single GPU can be extended to multi-GPU systems by replacing CQ and NQ with the arrays of bit-vector *in_queue* and *out_queue*, respectively.

1. Each GPU has an adjacency matrix in the CSR sparse matrix format. First, the root is stored in *in_queue*, and corresponding bsf_tree is marked as visited. Others are set to be unvisited by storing 0. The content of *out_queue* is also initialized to be 0.
2. Each GPU checks an assigned vertex u. If u is unvisited, go to the next step. Otherwise, check the next vertex.
3. Check all neighboring nodes of u. If the corresponding location of *in_queue* is 1, update bsf_tree, and write 1 into *visited* and *out_queue*.
4. Gather all data in *out_queue* of all GPUs and make an *in_queue*. If all values in *in_queue* are 0, it shows that the search is finished.
5. Go to step 2.

This algorithm includes communication between GPUs connected to different nodes for exchanging data in *out_queue*.

When BFS is executed in a multi-GPU system, GPUs need to communicate with other GPUs connected to the different nodes. The communication across the node becomes the performance bottoleneck. To reduce the communication overhead, the amount of transfer data is compressed by using the replicated-csr method. This data exchange is done by using the MPI function in common Infiniband clusters. In the case of TCA, we can use an application programming interface(API) by using the shared data space supported by PEACH2.

4.2 Communication data size reduction

Each *out_queue* is a bit-vector whose location correspond to an index of a vertex. That is, we need to transfer the *out_queue* whose size is equal to $(allvertices/thenumberofGPUs)$. Here, we compress the size of exchanged data with the method shown in Figure 5.

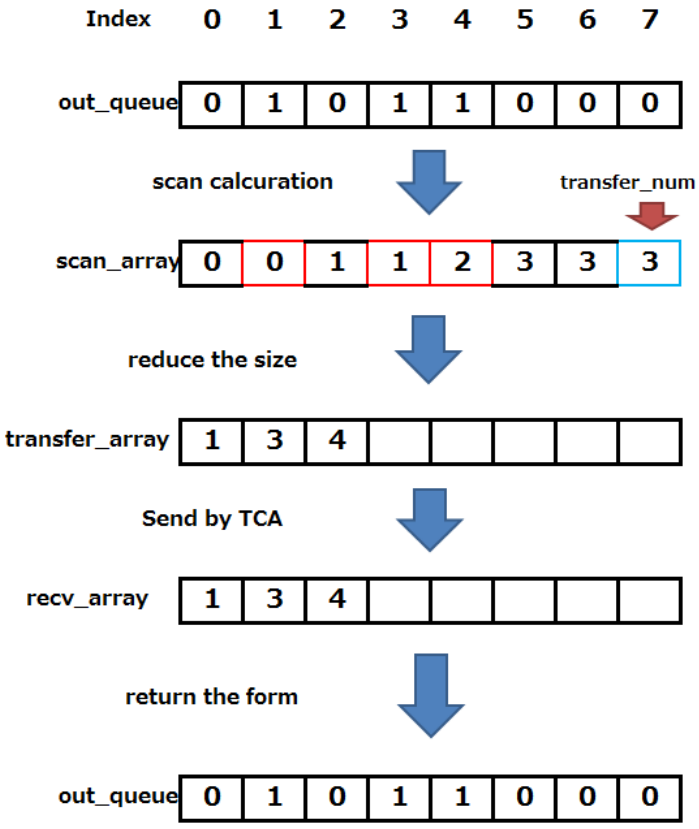


Figure 5: An example of bit-vector compression

First, a *scan_array* is made by performing a scan calculation that accumulates the number "1" in *out_queue*. If there is a position where the number in *scan_array* is changed it means that there is a "1" in the *out_queue*. Thus, we can make a $transfer\_array$ that only includes the position of "1", and it is transferred until "-1" is found. A GPU can restore the original "out_queue" from the receiving data. Note that this compression can be performed for the "out_queue" of all vertices in parallel. This method is especially advantageous when the target graph is sparse. The compression algorithm is shown in Algorithm2.

Algorithm 2 Compression algorithm

```
1: scan_array ← Perform scan calculation for out_queue
   of all vertices in parallel
2: transfer_num ← scan_array[last]
3: if out_queue[index] = 1 then
4:   transfer_array[scan_array[index]] ← index in par-
     allel
5: end if
6: Transfer transfer_array
7: Return the in_queue the flag form
```

Note that this compression is only effective for TCA communication that supports quick transfer for small data size. For MPI communication, MPI_Allgather() which only supports fixed length is advantageous for Infiniband GPU clusters.

4.3 The communication path in a GPU

In the target multi-GPU system, two GPUs form a node and two nodes are connected with TCA or MPI. First, two GPUs in the same node communicate with each other by using the CUDA API. Then, GPUs connected to the different nodes exchange the data by using the TCA API or MPI as shown in Figure 6. For a GPU, all data can be gathered with two inner-node data transfers and one inter-node data transfer.

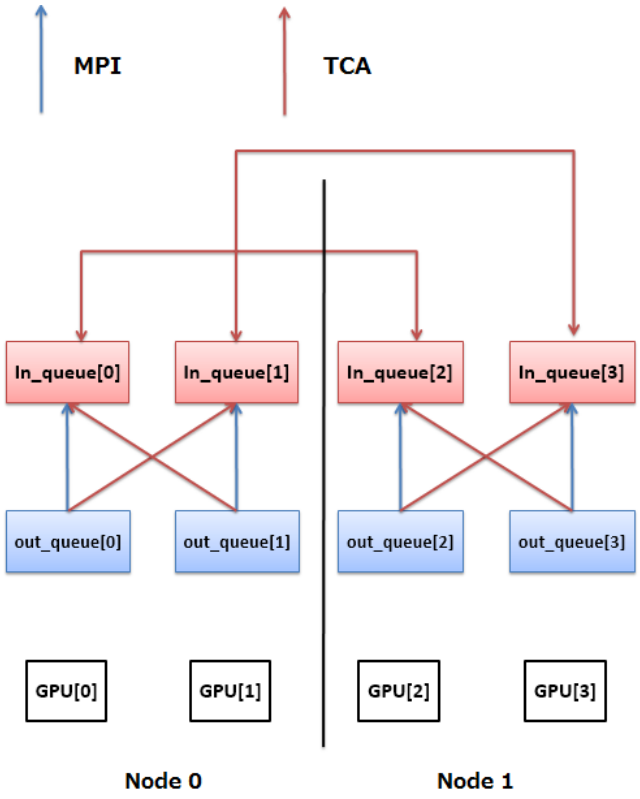


Figure 6: Communication path

5. EVALUATION

In this section, we show how we evaluated and compared the performance of the BFS with TCA and that with MPI. Graphs from the Graph500 benchmark are used as target graphs.

5.1 Evaluation environment

We used an experimental cluster with two nodes in the University of Tsukuba's Center for Computational Sciences. The node specifications are shown in Table3. In the system, two nodes are connected with TCA by using two PEACH2 boards attached to each node. Each host has Infiniband as an independent network, which enables to compare two networks having exactly the same node configuration.

5.2 Graph500

For the measurements we used the Graph500, which is a benchmark for measuring performance by evaluating the processing time of the graph search.

A graph is generated with parameters called *scale* and $edgefactor$. The *scale* represents the number of vertices of the graph with the formula: $the\ number\ of\ vertices\ = 2^{scale}$. The $edgefactor$ determines the number of edges: that is, $the\ number\ of\ edges\ = the\ number\ of\ vertices\ * edge\ factor$

Performance is represented by the number of edges traversed in a second. This measure is called TEPS (Traversed Edges Per Second); a larger TEPS means better performance.

5.3 Evaluation result

Figures7, 8, and 9 show the results obtained for the BFS while changing the *scale* and the $edge\ factor$. All figures show that the BFS performance becomes better as the scale becomes larger. It is obvious that a lot of threads can work in parallel for large *scale*. In addition, the BFS using TCA is more advantageous than that using MPI with large *scale*. When $scale = 16$ and $edgefactor = 64$, the performance of the BFS using TCA is 1.58 times faster than that with MPI. Table.4 show the reduction ratio values obtained. We achieved reduction by roughly 40$\tilde{6}$0%. Figure10 is the graph that shows execution time when the data are exchanged between nodes. The execution time and target communication time are shown for "tca" or "mpi", which includes cudaMemcpyDtoH, HtoD, cudaMemset and kernels for data reduction in TCA, and cudaMemcpyDtoH HtoD and cudaMemset for the same in MPI.

Figure10 makes it clear that the run time for the TCA version is shorter in many cases. When the scale is small, there is large variation in the datae, and the results may therefore change for each run. However, even these small advantages will have a positive effect on the execution performance.

In the BFS, searching accounts for 90% of the execution time, and less than 10% of the communication time. It is unreasonable to assume that the data reduction and low latency communication achieved with TCA are the only reasons for the improved performance that was obtained. Another reason that can be considered is the data exchanged by sandwiching a process such as reduction while performing a large number of consecutive data write operations. However, it is difficult to analyze this in detail. In the work we report here, we measured the communication time with MPI_Wtime() and the others by using "nvprof". However, the measurement data obtained is not enough sufficiently accurate for conducting a detailed analysis.

When $9 <= scale <= 10$, the BFS with MPI outperformed that with TCA. This might come from the fact that in this case the execution time needed for the reduction cancels positive the effects of the small latency.

6. CONCLUSION

We implemented the breadth-first search(BFS) algorithm on multiple graphics processing unit clusters using the Tightly Coupled Accelerators(TCA) architecture and optimized the communication for the TCA by making use of its lower

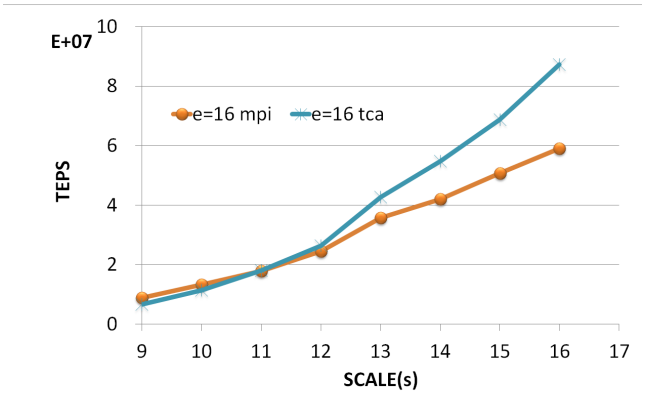


Figure 7: $edgefactor = 16$

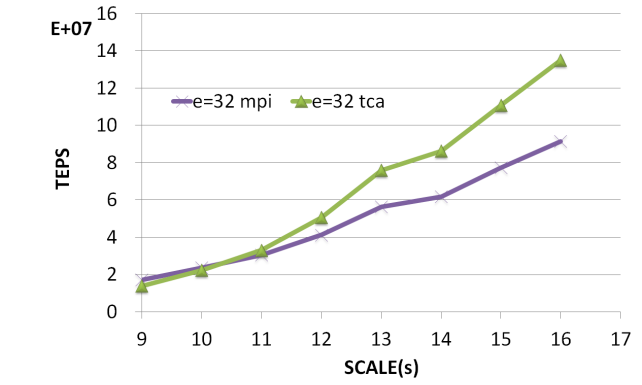


Figure 8: $edgefactor = 32$

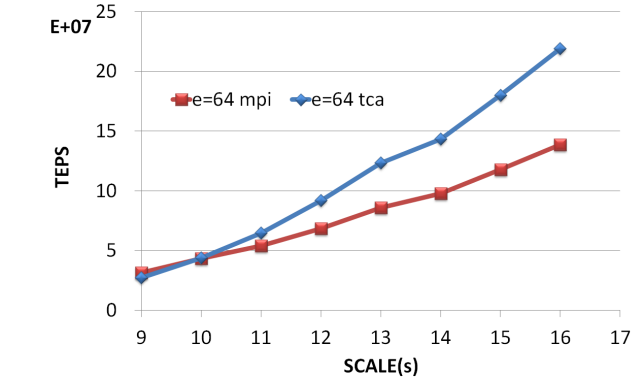


Figure 9: $edgefactor = 64$

Table 4: average ratio after data reduction(%)

		SCALE							
		9	10	11	12	13	14	15	16
edge factor	16	32.1	37.0	42.5	44.6	47.5	51.3	54.2	56.4
	32	25.8	32.4	35.4	40.7	43.9	45.0	46.4	47.6
	64	76.9	70.8	67.8	64.6	63.0	61.0	59.8	58.2

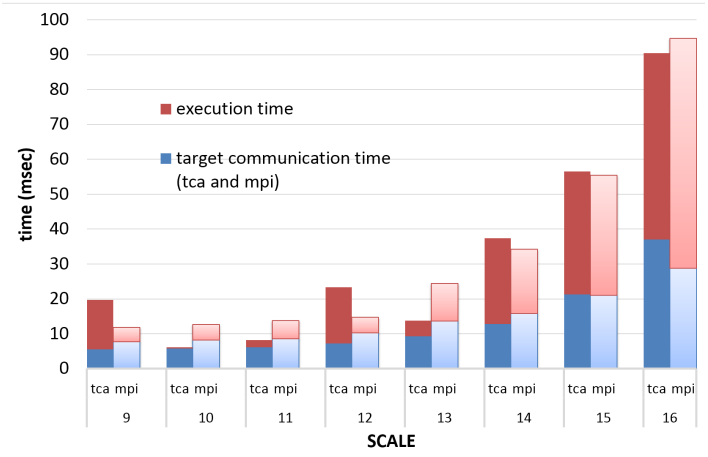


Figure 10: execution_time

latency than the MPI over Infiniband. For a graph from Graph500, the BFS with the TCA achieved 1.58 times performance than that with the MPI.

In future work, we will need to evaluate the performance obtained carrying out the following optimization procedures.

- Optimizing for larger scale,
- optimizing for a larger number of nodes,
- using additional techniques to reduce data size,
- using the new application programming interface (API) for PEACH2, and
- developing more accurate profiling tools.

These procedures will be necessary because currently the scale and the number of corresponding nodes are too small for the system that the TCA targets. In addition there are still a lot of data reduction methods that can be applied, and the newly developed API for PEACH2 is more suitable for higher performance implementation. Finally, more accurate profiling tools will need to be developed.

7. ACKNOWLEDGEMENT

The present study is supported in part by the JST/CREST program entitled "Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era" in the research area of "Development of System Software Technologies for post-Peta Scale High Performance Computing".

8. REFERENCES

[1] S. Otani, H.Kondo, T. Hanawa, S. Miura, and T. Boku. Peach: A multicore communication system on chip with pci express. In *IEEE Micro*, pp. 39–50, 2011.

[2] T. Hanawa, Y. Kodama, T. Boku, and M. Sato. Tightly coupled accelerators architecture for minimizing communciation latency among accelerators. In *IEEE 27th IPDPSW*, pp. 1030–1039, 2013.

[3] Yuetsu Kodama, Toshihiro Hanawa, Taisuke Boku, and Mitsuhisa Sato. PEACH2: An FPGA-based PCIe network device for Tightly Coupled Accelerators. In *Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2014)*, pp. 5–10, 6 2014.

[4] Graph 500. "`http://www.graph500.org/`".

[5] Pawan Harish and P.J. Narayanan. Accelerationg Large Graph Algorithms on the GPU Using CUDA. In *HiPC 2007*, pp. 197–208, 2007.

[6] Enrico Mastrostefano. *Large Graphs on multi-GPUs*. PhD thesis, Spienza University of Roma, 2013.

[7] Takuji Mitsuishi, Shimpei Nomura, Jun Suzuki, Yuki Hayashi, Masaki Kan, and Hideharu Amano. Accelerating breadth first search on gpu-box. In *International symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, HEART'14, July 2014.

[8] Toyotaro Suzumura, Koji Ueno, Hithoshi Sato, Katsuki Fujisawa, and Satoshi Matsuoka. Performance Evaluation of Graph500 on Large-Scale Distributed Environment. In *IISWC*, pp. 149–158, Nov 2011.

Butterflies Solve Bidiagonal Toeplitz Systems

Brian J. Murphy[1,2]
brian.murphy@lehman.cuny.edu

Aron Wolinetz[2]
aronw26@gmail.com

Joshua Rogers[1]
joshuajoskow@gmail.com

[1] Department of Mathematics and Computer Science
Lehman College, Bronx, NY 10468 USA
City University of New York

[2] Ph.D. Program in Computer Science
The Graduate Center, New York, NY 10016 USA
City University of New York

Abstract— *We reduce bidiagonal Toeplitz matrix inversion to f-circulant matrix inversion. In so doing we derive a fast Fourier transform based parallel solver for diagonally dominant as well as ill-conditioned bidiagonal Toeplitz systems. The number of parallel steps required by our algorithm depends on the degree of diagonal dominance or ill-conditioning and not system size. Our algorithm is designed with implementation on an array of parallel FFT processors in mind.*

Keywords: Bidiagonal Toeplitz, Diagonally Dominant, Ill-conditioned, FFT.

Introduction

Our subject is parallel solution of a bidiagonal Toeplitz system of linear equations $T\mathbf{x} = \mathbf{b}$ which expands to

$$\begin{pmatrix} 1 & & & \\ c & 1 & & \\ & \ddots & \ddots & \\ & & c & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}. \quad (1)$$

Here and hereafter $T = (t_{i,j})_{i,j=0}^{n-1}$, $t_{i,j} = 1$ for $i - j = 0$, $t_{i,j} = c \neq 0$ for $i - j = 1$, and $t_{i,j} = 0$ otherwise, $\mathbf{x} = (x_i)_{i=0}^{n-1}$, and $\mathbf{b} = (b_i)_{i=0}^{n-1}$. Note that the system is scaled so that the main diagonal is composed exclusively of ones, with no loss of generality. These systems are at the heart of problems as diverse as cubic spline and B-spline curve fitting [3], [11], preconditioning for iterative linear solvers [2], [13], computation of photon statistics in lasers [8], computational fluid dynamics [26], solution of multidimensional diffusion computations [9], solution of neuron models by domain decomposition [14], and more.

We propose a parallel algorithm based on the fast Fourier transform (FFT) to solve such systems. Cooley and Tukey [4] introduced the FFT in 1965 as a mechanism for fast computation of the discrete Fourier transform (DFT) on computers. General parallel architectures tend to suffer from high latency and restrictive bandwidth for communication between processing units and/or clusters of processing units and dedicate effort to fetching and decoding instructions. Specialized hardware to perform the FFT in parallel alleviates these impediments to efficient parallel computation. A simple set of computations at the heart of the FFT can be carried out by a circuit known as the Butterfly. A Butterfly circuit as depicted in Figure 1 accepts two inputs. One input is multiplied by what is referred to as a twiddle factor, which are nth roots of unity for an FFT of dimension n. Then in parallel this product is both added to and subtracted from the other input. The two resulting values are the output of the Butterfly circuit. FFT processors generally incorporate $\log_2 n$ stages of $n/2$ parallel Butterfly circuits to provide pipelined computation of an n point DFT.

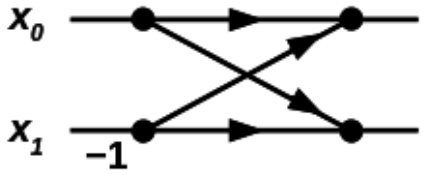


Fig. 1: Butterfly circuit with two inputs x_0 and x_1 and twiddle factor -1.

Field Programmable Gate Arrays (FPGAs) capable of implementing a 64 point parallel pipelined FFT and capable of producing 25.6 giga samples per second (GSPS) are readily available [7]. For our Bidiagonal Toeplitz solver the pipelining of the underlying FFTs available from FPGA's and other more efficient hardware is of great practical interest. This both because a number of real world applications involve solving large numbers of these systems [10], [26] and because our method partitions the computation into multiple smaller independent FFTs that can be pipelined if not performed entirely in parallel.

Our algorithm is fast, scalable, fine grained, and exhibits a simple implementation based on an FFT kernel. It depends on a reduction of the inversion of T to the inversions of a circulant matrix and a skew circulant matrix. Circulant and skew circulant matrices are defined by their first column, maintain their structure when inverted, and are diagonalized via FFT so that they are easily and efficiently inverted by applying an inverse FFT (IFFT) to the element-wise reciprocal of their first column's Fourier image [21]. An n input FFT requires $O(n \log_2 n)$ arithmetic operations (ops)

and $O(\log_2 n)$ parallel steps. Our algorithm improves on this bound for $n \times n$ systems by partitioning the problem into multiple smaller instances of the original problem, each of which can be processed independently of the others. The level of partitioning and therefore the number of parallel steps required by our algorithm is determined by the degree of diagonal dominance or ill-conditioning inherent in the system and not by n. The partitioning reduces both the number of ops and parallel steps required to produce a solution as well as the number of transistors applied. This algorithm is fully scalable in an embarrassingly parallel way, because multiple smaller and independent FFTs replace a single larger FFT and for a given degree of diagonal dominance or ill-conditioning the size of these FFT computations is constant as n grows.

0.1 Organization of our paper

We have organized the rest of our paper as follows: We provide some background in Section 1. We consolidate definitions and preliminary facts for later use in the next section. We present original work, which forms the mathematical basis for our algorithm in Section 3. We present and analyze our algorithm in Section 4. We report experimental results in Section 5.

1. Background

1.1 Computation with Toeplitz Matrices

Given vector $\mathbf{t} = (t_i)_{i=1-n}^{m-1}$, $T = (t_{i,j})_{i,j=0}^{m-1,n-1}$ where $t_{i,j} = t_{i-j}$ is a Toeplitz matrix. Such matrices are invariant along their diagonals. Note that it is the special case for T illustrated in (1) to which T will refer hereafter.

An intimate relationship holds between polynomial multiplication and many Toeplitz matrix computations. We recall the basis for this simple but fundamental connection.

Theorem 1: (Cf. [21, Section 2.4.3].) The matrix equation

$$\begin{pmatrix} u_0 & & O \\ \vdots & \ddots & \\ \vdots & \ddots & u_0 \\ u_m & \ddots & \vdots \\ & \ddots & \vdots \\ O & & u_m \end{pmatrix} \begin{pmatrix} v_0 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} p_0 \\ \vdots \\ \vdots \\ p_m \\ \vdots \\ p_{m+n} \end{pmatrix} \tag{2}$$

is equivalent to the polynomial equation

$$\left(\sum_{i=0}^{m} u_i x^i\right)\left(\sum_{i=0}^{n} v_i x^i\right) = \sum_{i=0}^{m+n} p_i x^i.$$

The theorem immediately reduces polynomial multiplication to Toeplitz matrix-by-vector multiplication and vice versa [21, Section 2.4].

It is well known that the product of two polynomials degree n and m, $n \geq m$, can be determined via a convolution of their coefficient vectors efficiently via FFT in $O(n \log_2 n)$ ops [5]. The FFT is utilized to perform a multi-point evaluation of both polynomials. Corresponding points on the polynomials are multiplied producing points on the product polynomial. Finally, these points are interpolated to produce the product polynomial via IFFT. The link demonstrated above in Theorem 1 tells us that the same fast procedure will accommodate Toeplitz matrix-by-vector multiplication [21]. By extension FFTs apply to Toeplitz matrix multiplication, inversion, and system solving [1], [19], [25].

1.2 Some Bidiagonal and Tridiagonal Solvers

One can invert an $n \times n$ bidiagonal Toeplitz matrix T, as depicted in (1), in $n+1$ arithmetic operations (ops) via simple application of the recurrence

$$m_0 = 1, \quad m_k = -cm_{k-1},$$

where $\mathbf{m} = (m_i)_{i=0}^{n-1} = ((-c)^i)_{i=0}^{n-1}$ is the defining first column of triangular Toeplitz matrix

$$T^{-1} = \begin{pmatrix} 1 & & & \\ -c & 1 & & \\ \vdots & \ddots & \ddots & \\ (-c)^{n-1} & \dots & -c & 1 \end{pmatrix}. \tag{3}$$

Further, one can solve an $n \times n$ bidiagonal Toeplitz system of equations $T\mathbf{x} = \mathbf{b}$ in $2n-1$ ops via the recurrence $x_0 = b_0, \quad x_i = b_i - cx_{i-1}$ for $i = 1, 2, \dots, n-1$. However, due to data dependencies between successive iterations parallelization is stymied.

Other algorithms incorporate additional ops to achieve parallelism. For instance recursive doubling [24] requires $O(n \log_2 n)$ ops and while cyclic reduction [10] and the Spike Algorithm [22] both require $O(n)$ ops their constants of proportionality far exceed 2. $O(\log_2 n)$ parallel steps is the asymptotic record bound for direct solution of bidiagonal systems in general and is shared by the aforementioned algorithms. In the case of a diagonally dominant system, these and other algorithms can trade accuracy for some speedup by employing an early termination. The speedup, however appears insignificant [12]. Additionally, these and other algorithms either fail to provide fine grained parallelism or their implementation on generalized parallel hardware necessitates agglomeration of data elements due to tremendous communication demands thereby requiring the surrender of parallelism.

1.3 Connecting Bidiagonal and Tridiagonal Toeplitz System Solving

A tridiagonal Toeplitz matrix

$$M = \begin{pmatrix} d & 1 & & \\ c & d & \ddots & \\ & \ddots & \ddots & 1 \\ & & c & d \end{pmatrix}$$

is easily and efficiently split into the product of two bidiagonal Toeplitz matrices plus a perturbation term via the quadratic formula. Our focus is on two different ways to represent M, i.e.

$$\begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ r_1 & & \ddots & & \\ & \ddots & & \ddots & \\ & & r_1 & 1 \end{pmatrix} \begin{pmatrix} r_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & r_2 \end{pmatrix} + \begin{pmatrix} d-r_2 & & & \\ & & & \\ & & & \\ & & & \end{pmatrix},$$

and

$$\begin{pmatrix} r_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & r_2 \end{pmatrix} \begin{pmatrix} 1 & & & & \\ r_1 & \ddots & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & r_1 & 1 \end{pmatrix} + \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & d-r_2 \end{pmatrix},$$

where $r_1 = \frac{d \pm \sqrt{d^2-4c}}{2}$, $r_2 = d - r_1$ and $|r_1| < 1 < |r_2|$. Conforming values for r_1 and r_2 can always be found [20]. Hereafter, we write the matrix equations $M = LU + P_1$ and $M = UL + P_2$ to represent the above depictions of M.

For a symmetric M several algorithms were devised. Rojo [23] solved $(LU + P_1)\mathbf{x} = \mathbf{b}$ by solving $LU\mathbf{y} = \mathbf{b}$ via backward and then forward substitution followed by application of the Sherman Morrison formula to correct for the perturbation represented by P_1. Yan and Chung [27] based their approximation algorithm on the same splitting, but corrected a smaller segment of the solution to $LU\mathbf{y} = \mathbf{b}$. McNally [16] and McNally et al. [18] built on these works to produce parallel algorithms for approximating the solution. McNally [17] and Nemani [20] both extended these ideas to non-symmetric systems. For each of these algorithms the pattern established by Rojo [23] whereby a correction step is applied after solving a system based on the perturbed matrix $M - P_1 = LU$ continued. For diagonally dominant systems McNally [15] did away with the correction step by solving portions of both $LU\mathbf{y} = \mathbf{b}$ and $UL\mathbf{z} = \mathbf{b}$ recognizing that the leading elements of $\mathbf{z}$ and the trailing elements of $\mathbf{y}$ are very good approximations for the corresponding elements of $\mathbf{x} = M^{-1}\mathbf{b}$. This eliminated communication between partitions of the decomposition. Each of these methods based on the Toeplitz $LU + P$ splitting applies a course-grained decomposition.

1.4 Executing a Parallel Bidiagonal Solver

The runtime clocked speed of a parallel algorithm is intimately tied to the hardware on which it is executed. It is a function of many parameters beyond the number of ops and/or steps the algorithm performs. Number of processors, memory hierarchy, synchronization, communication, latency, bandwidth, and other factors can play a major role in its determination. The core computations at the heart of most parallel bidiagonal and tridiagonal solvers do not generalize to a vast array of other computations. Therefore specialized hardware dedicated to these computations is not readily available. Most parallel bidiagonal and tridiagonal solvers then are just about guaranteed to be implemented in software for processing on a generalized hardware platform such as networked central processing units, vector processors, and graphics processing units. Such devices incur overhead costs for performance of their fetch execute cycle, memory hierarchy latencies and bandwidth limitations, as well as other inefficiencies built-in to allow for the flexibility of executing an algorithm inscribed in software. Of course where needed network communication can be a huge drag on performance when its latency cannot be hidden behind computation and when bandwidth becomes a bottleneck.

Parallel bidiagonal and tridiagonal solvers can encounter additional implementation issues that limit their processing speed, particularly for large system size and where the finest of granularity is desired. Take CR and Spike for instance. Implementation of CR generally requires agglomeration of data points for processing large systems. This due to the need for communication at every stage between neighboring nodes, where those nodes considered to be neighbors are increasingly distant at each stage of the parallel computation. To make such communication reasonably efficient one would need to overcome ostensibly inherent obstacles in memory system design to meet the seemingly contradictory requirement to produce a fast symmetric shared memory serving all processors. On the other hand the Spike Algorithm successfully decouples computation into independent partitions, but does not allow for the finest of granularity. Additionally, the decomposition stage of Spike can involve multiple iterations thereby adding parallel steps. Work efficiency can be an important factor in choosing a parallel algorithm. This is certainly a major consideration when the number of data points is far in excess of the number of available processors. In such a case the CR and Spike algorithms can be good candidates for implementation.

On the other hand, achieving a payoff by matching processor count to the number of data points is a difficult proposition. Maximizing parallelism can often be counterproductive. Rather than boosting performance, communication can easily overwhelm computation and result in performance degradation [6].

Our algorithm when matched to FFT processors avoids all of these issues that plague the efficient implementation of other algorithms for solving these systems.

2. Some Preliminaries

2.1 Definitions

- $\Omega = (\Omega_i)_{i=0}^{n-1}$ where $\Omega_i = e^{(-2\pi\sqrt{-1}/2n)i}$.
  Each Ω_i is a $2n^{th}$ root of unity.
- I is the $n \times n$ identity matrix, n defined by context.
- $\mathbf{e}_i$ is the i^{th} column of I.
- $||M||_2$ is the 2-norm of a matrix M.
- ϵ is the machine epsilon, i.e. the smallest floating point value such that $1 \pm \epsilon \neq 1$.
- $\mathbf{u} = (u_i)_{i=0}^{n-1}$ and $\mathbf{v} = (v_i)_{i=0}^{n-1}$ are n-vectors.

- $\mathbf{w} = (w_i)_{i=0}^{k-1}$ is a k-vector, $k \leq n$.
- $\mathbf{u} .* \mathbf{v} = (u_i v_i)_{i=0}^{n-1}$.
- $s ./ \mathbf{v} = (s/v_i)_{i=0}^{n-1}$.
- $\mathbf{u} ./ \mathbf{v} = (u_i/v_i)_{i=0}^{n-1}$.
- $\text{top}_k(\mathbf{v}) = (v_i)_{i=0}^{k-1}$: leading k elements of $\mathbf{v}$, $k \leq n$.
- $\text{bot}_k(\mathbf{v}) = (v_i)_{i=n-k}^{n-1}$: trailing k elements of $\mathbf{v}$, $k \leq n$.
- $\text{pad}_n(\mathbf{w}) = \mathbf{v} = (v_i)_{i=0}^{n-1}$, where $v_i = w_i$ for $0 \leq i < k$ and $v_i = 0$ for $k \leq i < n$.
- $F_n(\mathbf{w})$: n-vector DFT of n-vector $\text{pad}_n(\mathbf{w})$.
- $W_n(\mathbf{u})$: n-vector Inverse DFT (IDFT) of n-vector $\mathbf{u}$.
- $m(n)$: number of steps required to compute $F_n(\mathbf{w})$ on n processors.
- $Z_f = (z_{i,j})$: $n \times n$ matrix such that $z_{i,i-1} = 1$, for $i = 1, \ldots, n-1$, $z_{0,n-1} = f$, and $z_{i,j} = 0$ for all other pairs (i, j).
- $Z_f(\mathbf{v}) = \sum_{i=0}^{n-1} v_i Z_f^i$: $n \times n$ f-circulant matrix defined by its first column $\mathbf{v}$, and scalar f, so that matrices $Z_0(\mathbf{v})$, $Z_0^{\mathsf{T}}(\mathbf{v})$, $Z_1(\mathbf{v})$, and $Z_{-1}(\mathbf{v})$ are lower triangular Toeplitz, upper triangular Toeplitz, circulant, and skew-circulant respectively.

2.2 Facts

The following are known (see [21]) and presented without proof:

$$Z_f^{-1}(\mathbf{v}) = Z_f(\mathbf{u}), \text{ where each pair } \mathbf{v}, \mathbf{u} \text{ is unique.} \quad (4)$$

$$Z_0(\mathbf{v}) = [Z_f(\mathbf{v}) + Z_{-f}(\mathbf{v})]/2 \quad (5)$$

$$Z_0(\mathbf{v})\mathbf{u} = \text{top}_n(W_{2n}(F_{2n}(\mathbf{v}) .* F_{2n}(\mathbf{u}))) \quad (6)$$

$$Z_1^{-1}(\mathbf{v})\mathbf{e}_0 = W_n(1./F_n(\mathbf{v})) \quad (7)$$

$$Z_{-1}^{-1}(\mathbf{v})\mathbf{e}_0 = W_n(1./F_n(\Omega .* \mathbf{v}))./\Omega \quad (8)$$

$$Z_1(\mathbf{v})\mathbf{u} = W_n(F_n(\mathbf{v}) .* F_n(\mathbf{u})), \quad (9)$$

$$Z_{-1}(\mathbf{v})\mathbf{u} = W_n(F_n(\Omega .* \mathbf{v}) .* F_n(\Omega .* \mathbf{u}))./\Omega \quad (10)$$

We will express our algorithm in terms of lower bidiagonal Toeplitz matrices, however, because $(Z_0^{-1}(\mathbf{v}))^{\mathsf{T}} = (Z_0^{\mathsf{T}}(\mathbf{v}))^{-1}$ they apply equally to the upper bidiagonal case. Hereafter, where it is notationally convenient and without loss of generality because $\text{top}_m(Z_0^{-1}(\mathbf{v})\mathbf{e}_0) = Z_0^{-1}(\text{top}_m(\mathbf{v}))\mathbf{e}_0$, $n \geq m$, we assume $n = 2^i$ for a positive integer i.

3. The Basis for our Algorithm

3.1 In General

The following lemma indicates that given lower bidiagonal Toeplitz matrix $T = Z_0^{-1}(\mathbf{v})$ and f-circulant matrix $C = Z_f^{-1}(\mathbf{v})$ defined by identical first columns, $\mathbf{v}$, a scalar factor is all that differentiates the first columns of T^{-1} and C^{-1}.

Lemma 1: $Z_0^{-1}(\mathbf{v})\mathbf{e}_0 = [1 - f(-c)^n]Z_f^{-1}(\mathbf{v})\mathbf{e}_0$ where $\mathbf{v} = \mathbf{e}_0 + c\mathbf{e}_1$ and $f(-c)^n \neq 1$.

Proof: $Z_f(\mathbf{v})$ and $Z_0(\mathbf{v})$ differ only in the last element of their respective first rows. Inspection confirms $Z_0^{-1}(\mathbf{v})\mathbf{e}_0 = ((-c)^i)_{i=0}^{n-1}$. Consider then the product $Z_f(\mathbf{v})Z_0^{-1}(\mathbf{v})\mathbf{e}_0$. Its first element, the dot product of $Z_f^{\mathsf{T}}(\mathbf{v})\mathbf{e}_0$ and $Z_0^{-1}(\mathbf{v})\mathbf{e}_0$, is $(1)(1) + fc[(-c)^{n-1}] = 1 - f(-c)^n$. Because all other rows of $Z_f(\mathbf{v})$ are identical to the corresponding rows of $Z_0(\mathbf{v})$ the remaining $n-1$ elements of $Z_f(\mathbf{v})Z_0^{-1}(\mathbf{v})\mathbf{e}_0$ are all zero. We have then that $Z_f(\mathbf{v})Z_0^{-1}(\mathbf{v})\mathbf{e}_0 = [1 - f(-c)^n]\mathbf{e}_0$. Clearly then $Z_0^{-1}(\mathbf{v})\mathbf{e}_0 = [1 - f(-c)^n]Z_f^{-1}(\mathbf{v})\mathbf{e}_0$. ■

The next lemma reduces inversion of a lower bidiagonal Toeplitz matrix to the inversions of a circulant and a skew circulant matrix.

Lemma 2: $Z_0^{-1}(\mathbf{v}) = \{[1 - (-c)^n]Z_1^{-1}(\mathbf{v}) + [1 + (-c)^n]Z_{-1}^{-1}(\mathbf{v})\}/2$, where $\mathbf{v} = \mathbf{e}_0 + c\mathbf{e}_1$ and $|c^n| \neq 1$.

Proof: Due to (4), $Z_0^{-1}(\mathbf{v}) = Z_0(\mathbf{u})$ and $[1 - f(-c)^n]Z_f^{-1}(\mathbf{v}) = Z_f(\mathbf{w})$. According to Lemma 1, n-vector $\mathbf{u} = \mathbf{w}$. (5) tells us $Z_0(\mathbf{u}) = [Z_1(\mathbf{u}) + Z_{-1}(\mathbf{u})]/2$. ■

3.2 In the Diagonally Dominant Case

Where $|c| < 1$ we have diagonal dominance. Obviously, for numerical computation, if $|c^n| < \epsilon$ then $1 \pm (-c)^n = 1$. Under such circumstances Lemma 2 tells us that

$$Z_0^{-1}(\mathbf{v})\mathbf{b} \approx [Z_1^{-1}(\mathbf{v}) + Z_{-1}^{-1}(\mathbf{v})]\mathbf{b}/2. \quad (11)$$

Clearly, from (5), (6), (7), (8), (9), (10), and (11), in this case $Z_0^{-1}(\mathbf{v})\mathbf{b} = \text{top}_n(W_{2n}(F_{2n}(\mathbf{b})./F_{2n}(\mathbf{v})))$, and so $Z_0^{-1}(\mathbf{v})\mathbf{b}$ is computed as a subvector of a vector convolution.

3.3 In the Ill-Conditioned Case

Where $|c| > 1$ we may have an ill-conditioned system. Here we demonstrate that (11) applies when $|c^n| > 1/\epsilon$. Obviously, $b_0 = x_0$ and $b_i = x_i + cx_{i-1}$ otherwise. Let $S = Z_0^{\mathsf{T}}(\mathbf{u})$, where $u_0 = 0$ and $u_i = (-1)^{i-1}c^{-i}$ otherwise. After some telescoping, we see that $\hat{\mathbf{x}} = S\mathbf{b}$ takes the form

$$\begin{pmatrix} x_0 + c^{1-n}x_{n-1} \\ x_1 + c^{2-n}x_{n-1} \\ \vdots \\ x_{n-2} + c^{-1}x_{n-1} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & c^{-1} & -c^{-2} & \cdots & c^{1-n} \\ & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \\ & & & \ddots & -c^{-2} \\ & & & & c^{-1} \\ & & & & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ cx_0 + x_1 \\ cx_1 + x_2 \\ \vdots \\ cx_{n-2} + x_{n-1} \end{pmatrix},$$

where $\hat{\mathbf{x}} = (\hat{x}_i)_{i=0}^{n-1}$ such that $\hat{x}_{n-1} = 0$ and $\hat{x}_i = x_i + c^{1-n+i}x_{n-1}$, for $i = 0, 1, \ldots, n-2$. Given that the goal is to solve $T\mathbf{x} = \mathbf{b}$, computing the product $\hat{\mathbf{x}} = S\mathbf{b}$ results in an error vector

$$\mathbf{x} - \hat{\mathbf{x}} = \begin{pmatrix} -c^{n-1}x_{n-1} \\ -c^{n-2}x_{n-1} \\ \vdots \\ -c^{-1}x_{n-1} \\ x_{n-1} \end{pmatrix}. \quad (12)$$

It is clear from (12) that for large n many of the leading elements of $\mathbf{x}$ will be very well approximated by the corresponding elements of $\hat{\mathbf{x}}$. If $|c^k x_{n-1}| < \epsilon$ then $x_i = \hat{x}_i$ for $0 \leq i < k$ up to machine precision. Only the x_i for

$k \leq i < n$ are poorly approximated by $\hat{x}$. Obviously, k depends on c and to a lesser extent x_{n-1}, but not on n. Clearly, larger values of c, which correspond to systems "suffering" from higher condition numbers, will result in a lower absolute error in the approximations of each x_i.

Note that S is ill-conditioned. However, discarding its first column and last row, which are zero vectors, results in a submatrix that is well conditioned. The computation that determines $\hat{\mathbf{x}}$ is effectively limited to multiplication by this submatrix. Therefore multiplying $\mathbf{b}$ by S is numerically stable, unlike multiplying $\mathbf{b}$ by T^{-1}.

Consider the diagonally dominant triangular Toeplitz matrix $I - S$. Clearly it's inverse, $(I - S)^{-1}$, is a diagonally dominant upper bidiagonal Toeplitz matrix with $1/c$ on the first super diagonal. Therefore, we can solve $(I-S)^{-1}\mathbf{y} = \mathbf{b}$ to find

$$\hat{\mathbf{x}} = \mathbf{b} - \mathbf{y} = \mathbf{b} - (I - S)\mathbf{b} = \mathbf{b} + S\mathbf{b} - \mathbf{b}.$$

In effect we derive very good approximations for many elements of the solution vector to an ill-conditioned bidiagonal Toeplitz system by solving a diagonally dominant bidiagonal Toeplitz system followed by a vector subtraction.

Now consider Lemma 1 when $f = 1$ and $c^n > 1/\epsilon$. Under such circumstances $1 + c^n = c^n$ and $1 - c^n = -c^n$ at machine precision so that

$$Z_0^{-1}(\mathbf{v})\mathbf{e}_0 \approx -c^n Z_1^{-1}(\mathbf{v})\mathbf{e}_0 \approx c^n Z_{-1}^{-1}(\mathbf{v})\mathbf{e}_0$$

and of course

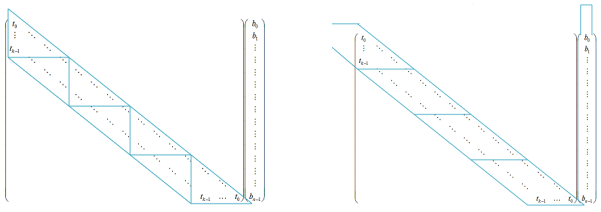
$$c^{-n} Z_0^{-1}(\mathbf{v})\mathbf{e}_0 \approx -Z_1^{-1}(\mathbf{v})\mathbf{e}_0 \approx Z_{-1}^{-1}(\mathbf{v})\mathbf{e}_0. \tag{13}$$

(13) and (3) tell us that

$$S \approx [Z_1^{-1}(\mathbf{v}) + Z_{-1}^{-1}(\mathbf{v})]/2. \tag{14}$$

(14) tells us that (11) applies not only to the diagonally dominant case, but to the ill-conditioned case as well.

3.4 Further Numerical Chicanery



(a) $k \times k$ blocks (b) Augmented with $k \times 2k$ blocks

Fig. 2: Partitioning the Computation

All entries on the ith sub-diagonal of T^{-1} are $(-c)^i$ as indicated in (3). Therefore, beyond a prescribed bandwidth k dependent on $|c|$, T^{-1} will decay to numerical insignificance when $|c| < 1$. In this case we deliver computational savings, in essence, by treating T^{-1} as a banded triangular Toeplitz matrix of bandwidth k or higher. For notational convenience and with no loss of generality then, hereafter, we assume $k = 2^i$ for a positive integer i and will refer to T^{-1} as if it were in fact k-banded.

We can partition T^{-1} into $k \times k$ blocks producing n/k identical lower triangular Toeplitz blocks along the block main diagonal and $n/k-1$ identical upper triangular Toeplitz blocks on the block first subdiagonal as depicted in Fig.2a. These $2n/k-1$ blocks encompass all "non-zero" elements of "k-banded" T^{-1}. However, we improve on this partitioning as illustrated in Fig.2b. We augment T^{-1} by concatenating k additional columns on its left to form an $n \times (n + k)$ matrix B that extends both the "k-banded" Toeplitz pattern of T^{-1} and the $k \times k$ block partitioning. We combine the pairs of non-zero blocks that share the same row of block matrix B to form n/k identical $k \times 2k$ Toeplitz blocks. We concatenate a zero vector of dimension k to the top of n-vector $\mathbf{b}$ to form $(n + k)$-vector $\hat{\mathbf{b}}$. We then solve the original system $T\mathbf{x} = \mathbf{b}$ by multiplying each of the n/k identical non-zero $k \times 2k$ blocks of B by the corresponding subvectors of $\hat{\mathbf{b}}$. These products are computed as wrapped convolutions of the first row of each of the n/k non-zero blocks of B and its corresponding subvector of $\hat{\mathbf{b}}$. Wrapped convolutions are computed efficiently via FFT/IFFT. It is the central n elements of these convolutions that we seek, and which survive the wrapping intact. Of course the previous subsection makes it clear that this partitioning scheme applies to ill-conditioned systems as well.

4. Our Algorithm

Based directly on the work presented in Section 3 we derive the following algorithm:

Algorithm 1: Diagonally Dominant Lower Bidiagonal Toeplitz Solver

INPUT: Scalar c where $\mathbf{t} = \mathbf{e_0} + c\mathbf{e_1}$ and $T = Z_0(\mathbf{t})$, and n-vector $\mathbf{b}$.

COMPUTE: $\mathbf{x} = (\mathbf{x}_i)_{i=0}^{n/k-1}$,
computing each $\mathbf{x}_i$ in parallel, where

$\mathbf{x}_i = \mathrm{bot}_k(W_{2k}(F_{2k}(\mathbf{b}_i)./F_{2k}(\mathbf{t})))$, where

$\mathbf{b}_i = (b_j)_{j=k(i-1)}^{k(i+1)-1}$, $b_j = 0$ for $j < 0$,

where $c^k < \epsilon$ for an integer k,
where k is the effective numerical bandwidth of T^{-1}

OUTPUT: $\mathbf{x} \approx T^{-1}\mathbf{b}$.

Algorithm 1 indicates that each of the $\mathbf{x}_i$ can be determined in parallel. Further these computations can be carried out in isolation with no need for communication across partitions. $F_{2k}(\mathbf{t})$ can be computed once and shared globally or computed redundantly more locally. $F_{2k}(\mathbf{t}) = F_{2k}(\mathbf{e}_0 + c\mathbf{e}_1) = F_{2k}(\mathbf{e}_0) + cF_{2k}(\mathbf{e}_1)$. Clearly then, we can pre-compute $F_{2k}(\mathbf{e}_0)$ and $F_{2k}(\mathbf{e}_1)$, so that computation of $F_{2k}(\mathbf{t})$ requires only $O(1)$ steps. $F_{2k}(\mathbf{t})$ and $F_{2k}(\mathbf{b}_i)$ can be computed simultaneously. Computing $F_{2k}(\mathbf{b}_i)$ via FFT requires $O(\log_2 k)$ steps as does application of the IFFT. The element-wise division is performed in $O(1)$ steps. Therefore, finding each $\mathbf{x}_i$ entails $2m(2k)+2$ steps. Note that this step count does not depend on n, but instead on c, as c determines k. Clearly, choosing k such that $c^k < \epsilon$ will provide optimum accuracy for this algorithm on a given machine.

5. Experimental Results

Experiments support this contention. Table 1 presents minimum values for k determined empirically.

$\lvert c\rvert \leq$	.34	.59	.76	.87	.93	.96	.98
k	32	64	128	256	512	1024	2048

Table 1: Experiments provide the above guidance for choosing the size of partitions, k, depending on the degree of diagonal dominance.

For proof of concept and to verify numerical stability, our algorithm has been tested in MATLAB and CUDA implementations, but not on the hardware for which it is best suited in terms of accelerating throughput.

c	Condition#	Recurance	G.E.	FFT
1.0E-08	1.0E+00	7.0E-19	7.0E-19	1.7E-16
1.0E-06	1.0E+00	8.3E-19	8.3E-19	1.8E-16
1.0E-03	1.0E+00	1.6E-18	1.6E-18	1.7E-16
1.0E-01	1.2E+00	3.0E-17	2.8E-17	1.7E-16
5.0E-01	3.0E+00	4.5E-17	4.5E-17	1.7E-16
9.0E-01	1.9E+01	8.0E-17	7.8E-17	1.8E-16
9.9E-01	2.0E+02	2.1E-16	2.0E-16	1.7E-16*
1.01E+00	1.6E+04	9.3E-15	9.0E-15	3.4E-02*
1.1E+00	2.8E+15	5.8E+03	5.8E+03	3.2E-02
2.0E+00	1.6E+16	9.5E-01	9.5E-01	3.2E-02
1.0E+01	Inf	NaN	NaN	3.2E-02
1.0E+03	Inf	NaN	NaN	3.4E-02
1.0E+06	Inf	NaN	NaN	3.9E-02
1.0E+08	Inf	NaN	NaN	3.7E-02

Table 2: Relative Errors $||b - T\hat{x}||_2/||b||_2$ by Method for $n = 2048$. * - indicates $n = 2048$ was not large enough for a meaningful result and so here $n = 8192$. For ill-conditioned matrices the table shows deceiving results, as a small number of poorly approximated elements greatly influence the norm. If these trailing elements are taken out of consideration the numbers for the ill-conditioned systems mirror those for the diagonally dominant.

We present accuracy results in Table 2. For each entry in Table 2, multiple runs of 1000 iterations were performed and the mean relative error determined. Relative error was calculated as $||b - T\hat{x}||_2/||b||_2$. Due to machine precision limits many iterations of the recurrence $x_0 = b_0, \quad x_i = b_i - cx_{i-1}$ and MATLAB's Gaussian elimination are found to have a relative error of zero. This contributes to an artificial lowering of some of the relative error means associated with these algorithms below machine detectable levels. In some cases the inaccuracy is around two orders of magnitude. Table 2 shows us that the new solver provides a high level of accuracy for the diagonally dominant case that comes very close to the machine's double precision accuracy. When the system is ill-conditioned our algorithm provides excellent results for a good many entries of the solution vector. The relative error calculation does not differentiate between a localized error and an error distributed throughout the solution vector and so it does not provide a full picture of the results in these cases. Empirical results though bear out the work in Section 3. In the ill-conditioned case, our algorithm produces a cluster of inaccurate values at the tail end of each partition of the solution vector. A simple tweak to our algorithm produces a solution vector with only one such cluster at the end of the solution vector. At the expense of additional computation, and potentially one additional parallel step, due to a potential doubling of the partition size, we overlap the matrix blocks so that the corrupted tail end of one partition is computed as the leading end of its next neighbor.

Our CUDA implementation was executed on an NVIDIA 580 GTX Fermi device. For real systems partitioned to match k to the 32 thread processors(TPs) of a symmetric multiprocessor (SMP), our algorithm is competitive with other algorithms implemented on Fermi devices. After distributing such partitions to all 16 TPs throughput is approximately 0.3 GSPS. As expected, as k grows other more work efficient algorithms grow in superiority there on. The use of Titan units from NVIDIA with 128 TPs per SMP and the register shuffling feature may tilt the field in favor of an FFT based algorithm where $k \leq 256$, but clearly an architecture designed to process FFTs will prevail here.

6. Concluding Remarks

While we lack the expertise to bring together the hardware components best suited for these algorithms, it is clear that the main component is readily available. For instance Dillon Engineering [7] produces a FPGA capable of producing 25.6 GSPS when configured as a pipelined 64 point FFT processor. The supporting circuitry around the FFT processor need only perform a handful of predetermined floating point operations in an embarrassingly parallel fashion. Our FFT based algorithm would allow an array of small FFT processors and their supporting circuitry embedded on FPGAs to tackle large linear systems by processing system blocks in complete isolation.

Clearly our algorithm could be implemented on a 64 point device to solve any bidiagonal Toeplitz system where $|c| \leq 1/4$. For a real system where $|c| \leq 1/2$ a few extra steps can be added due to symmetry in the Fourier image and the same 64 point device can be applied. At one end of the spectrum, a single such FPGA could handle all partitions on its own at a pipelined theoretical 25.6 GSPS for real systems. At the other end of the spectrum two FPGAs could be assigned to each partition. In addition to the embarrassingly parallel O(1) step computations, one FPGA would apply the FFT to the input as it arrives for processing and the other the IFFT that produces the final output. The FPGAs in separate partitions work independently in an embarrassingly parallel manner. Samples per second theoretically approach 25.6 GSPS times the number of partitions. Providing each processor with its input at a rate to keep the pipeline saturated is perhaps the biggest challenge. But even the memory can be partitioned into banks to serve the processors in parallel. Fortunately, if a hardware package including the FFT processor(s) can handle a single partition, scaling up could be almost as simple as dropping in additional hardware.

In all, our algorithm when implemented on FFT processors with minimal supporting circuitry has many advantages over other algorithms. Not only does it exhibit the finest of granularity, but is designed for implementation on readily available hardware that can actually take full advantage of that granularity thereby maximizing parallelism even for the largest of linear systems. For a given k its op count is $O(n)$ and it requires $O(1)$ parallel steps. That's $O(n \log_2 k)$ and $O(\log_2 k)$ respectively when k is not held constant, but $\log_2 k$ will not be large except when c is exceptionally close to 1 as Table 1 and extrapolation thereof indicates. CR requires $O(\log_2 n)$ steps but requires a great deal of agglomeration. SPIKE requires $O(n/p)$ steps on p processors, but is very course grained. Combine the low computational requirements of our algorithm with hardware designed to carry out these specific computations and it becomes clear that the algorithm will compete successfully.

The next step must be to find collaborators and get our algorithm implemented on FPGAs. In this context, in addition to FFTs, number theoretic transforms should be investigated as kernels for our algorithm.

A MATLAB implementation of our algorithm is found at: $http:\backslash\backslash comet.lehman.cuny.edu/bmurphy/bidiagonal$

References

[1] R. R. Bitmead and B. D. O. Anderson. Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations. *Linear Algebra and Its Applications*, 34:103Ű–116, 1980.

[2] Raymond H. Chan. Toeplitz preconditioners for Toeplitz systems with nonnegative generating functions. *IMA Journal of Numerical Analysis*, 11(3):333–345, 1991.

[3] K.-L. Chung and L.-J. Shen. Vectorized algorithm for b-spline curve fitting on Cray X-MP EA/16se. In *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, Supercomputing'92, pages 166–169, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[4] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math, Comp.*, 19:297Ű–301, 1965.

[5] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.

[6] James Demmel. *Avoiding Communication in Numerical Linear Algebra*.

[7] Dillon Engineering. http://www.dilloneng.com/fft_ip/parallel-fft.

[8] J. Guitart and S. Ruiz-Moreno. Strict calculation of the light statistics at the output of a traveling wave optical amplifier. *Electronic Letters*, 29:1589–1590, 1993.

[9] D. Heller. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM Journal on Numerical Analysis*, (13):484–496, 1976.

[10] R. W. Hockney. A fast direct solution of PoissonŠs equation using Fourier analysis. *Journal of the ACM*, 12(1):95–113, Jan. 1965.

[11] Josepllu is Larriba-Pey, Juan J. Navarro, and Angel Jorba. Vectorized algorithms for natural cubic spline and b-spline curve fitting. In *Proceedings of PDP'96*, pages 385–392, 1996.

[12] Josepllu is Larriba-Pey, Juan J. Navarro, Angel Jorba, and Oriol Roig. Review of General and Toeplitz Vector Bidiagonal Solvers. *Parallel Computing*, 22(8):1096–1126, 1996.

[13] Fu-Rong Lin and Wai-Ki Ching. Inverse Toeplitz preconditioners for Hermitian Toeplitz systems. *Numerical Linear Algebra with Applications*, 12(2–3):221–229, March/April 2005.

[14] M. Mascagni. A parallelizing algorithm for computing solutions to arbitrarily branched cable neurons. *Journal of Neuroscience Methods*, (36):105–114, 1991.

[15] Jeffrey M. McNally, L.E. Garey, and R.E. Shaw. A communication-less parallel algorithm for tridiagonal Toeplitz systems. *Journal of Computational and Applied Mathematics*, (212):260–271, 2008.

[16] Jeffrey Mark McNally. Fast parallel algorithms for tri-diagonal symmetric toeplitz systems. Master's thesis, University of New Brunswick (Canada), St. John, 1999.

[17] Jeffrey Mark McNally. *A scalable communicationless parallel algorithm for tri-diagonal Toeplitz systems*. PhD thesis, University of New Brunswick (Canada), St. John, 2003. AAINQ87633.

[18] Jeffrey Mark McNally, L.E. Garey, and R.E. Shaw. A split-correct parallel algorithm for solving tri-diagonal symmetric Toeplitz systems. *Internaional Journal of Computing and Mathematics*, 75:303–313, 2000.

[19] M Morf. Doubling Algorithms for Toeplitz and Related Equations. pages 954Ű–959, 1980.

[20] S. S. Nemani. *Perturbation methods for circulant-banded systems and their parallel implementation*. PhD thesis, University of New Brunswick (Canada), St. John, 2001.

[21] Victor Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Birkhäuser, Boston, MA, USA, 2001.

[22] Eric Polizzi and Ahmed H. Sameh. A parallel hybrid banded system solver: the spike algorithm. *Parallel Comput.*, 32(2):177–194, February 2006.

[23] O. Rojo. A new method for solving symmetric circulant tri-diagonal system of linear equations. *Comput. Math. Appl.*, (20):61–67, 1990.

[24] Harold S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *J. ACM*, 20(1):27–38, January 1973.

[25] Volker Strassen. Gaussian Elimination is not Optimal. *Numer. Math*, 13:354Ű–356, 1969.

[26] Xian-He Sun and Stuti Moitra. A fast parallel tridiagonal algorithm for a class of CFD applications. Technical Report 3585, NASA, 1996.

[27] W. M. Yan and K. L. Chung. A fast algorithm for solving special tri-diagonal systems. *Computing*, 52:203–211, 1994.

BSPonP2P: Towards Running Bulk-Synchronous Parallel Applications on P2P Desktop Grids

Rodrigo da Rosa Righi, Gustavo Rostirolla, Vinicius Facco Rodrigues, Alexandre Veith, Cristiano André da Costa
Applied Computing Graduate Program, Unisinos, Av. Unisinos, 950, São Leopoldo, Rio Grande do Sul, Brazil

Abstract— *Today, BSP (Bulk-Synchronous Parallel) represents one of the most used models for writing tightly-coupled parallel programs. A BSP application is divided in one or more supersteps, each one ending with a synchronization barrier. As resource substrates, commonly clusters and eventually computational grids are used to run BSP applications. In this context, we investigate the use of collaborative computing and idle resources to execute this kind of demand, so we are proposing a model named BSPonP2P to answer the following question: How can we develop an efficient and viable model to run BSP applications in P2P Desktop Grids? We answer it by providing both process rescheduling and checkpointing, enabling BSPonP2P to address dynamism at application and infrastructure levels and resource heterogeneity. The results concern a prototype that ran over a subset of the Grid5000 infrastructure, showing encouraging results on using collaboration and volatile resources for obtaining High Performance Computing effortlessly.*

Keywords: Bulk-Synchronous Parallel, P2P, Process Rescheduling, Checkpointing, Performance

1. Introduction

The widespread use of parallel machines crucially depends on the availability of a model of computation simple enough to provide a convenient basis for software development. Concerning this, the Bulk Synchronous Parallel (BSP) model of computation has been proposed by L. G. Valiant as an unified framework for the design and programming of general purpose parallel computing systems [17]. BSP applications are composed by a set of processes that execute supersteps, each one divided into three phases: (i) local computations on each process; (ii) global communication actions and; (iii) a synchronization barrier. The barrier phase should wait for the slowest process before starting the next superstep, so an efficient process-processors mapping is crucial for getting an acceptable performance [8]. This topic is yet more relevant when considering heterogeneous (different processing and network bandwidth capacities) and dynamic (fluctuations in network bandwidth and processors' load) environments [12].

BSP represents a common used model for writing successful parallel programs that exhibit phase-based computational behaviors, being extensively used to organize MPI (Message Passing Interface) codes [6]. As deployment machines, this programming model has been used on clusters and computational grids [7]. Particularly, this kind of grid is known by normally presenting a centralized or hierarchical architecture, high-speed networks linked to the Internet and nodes that slowly change their participation behavior along the time [5], [7]. In addition, for using one of the aforementioned parallel machines, the user must either buy the computational and network infrastructures or present a previous contract/agreement with the institution that host them. Concerning this landscape, we started the study of low cost and collaborative environments to take profit of end nodes around the Internet effortlessly. This effort culminated in an architecture proposed by Zhao, Liu and Li named P2P Desktop Grids [19] (PDG). Although joining the power of idle resources, a high level of dynamism with the sudden leaving of users (and consequently, reducing also the available CPU cycles of the system) and the use of worldwide-scale and Internet-based connections are challenges when associating this architecture with the purpose of HPC (High Performance Computing). In this way, our work presents the following problem statement: *How can we explore collaborative computing in PDG to run BSP applications efficiently?*

Aiming at answering the posed question, we are proposing *BSPonP2P* - a model that encompasses an infrastructure, overlay network, scheduling algorithms and runtime management to run BSP programs in PDG. Unlike computational grids, the target architecture is not managed by skilled professionals. BSPonP2P addresses collaborative computing at middleware level, where programmers do not need to change their applications in order to execute them in a P2P setting. Taking profit of the formal organization of BSP programs and mixing structured and nonstructured P2P deployments, the proposed model addresses performance but do not putting away the need of a fault tolerance layer that is crucial on PDG. This article describes BSPonP2P and its runtime strategies to answer the problem statement. Moreover, we also present its evaluation with a BSP scientific application in a real Grid infrastructure. In the best of our knowledge, BSPonP2P is the first proposal to cover BSP programs and P2P settings, being the pioneer on covering round-based parallel applications with spatial decoupling in an easier and costless way.

The remainder of this article will first introduce the related work in Section 2. Section 3 describes BSPonP2P

in details, demonstrating its rationales and contributions. Evaluation methodology and the discussion of the results are presented in Sections 4 and 5, respectively. Finally, Section 6 emphasizes the scientific contribution of the work and notes several challenges that we can address in the future.

2. Related Work

This section briefly presents initiatives to run applications on collaborative environments. Focusing to support Bag-of-Tasks (BoT) applications, the authors in [1] present a generic content-based publish/subscribe system called DPS. Moreover, Leite et al. [10] propose a load balancing architecture using a P2P-like structure for desktop grids.

Besides BoT, the master-slave approach is addressed in [4], [14], [15], [18]. Balasubramaniam et al. [2] and Byung et al. [16] presented aproaches targeting Desktop Grids, and Godfrey et al. [4], Shudo [15], Senís et al. [14] and Wu and Tian [18] present aproaches targeting PDG.

Concerning BSP parallel applications, both Mizan [8] and Camargo et al. [3] are representative for heterogeneous and dynamic environments. Mizan is a dynamic load balancing that captures data from computation and communication metrics. Camargo enable the use of not only idle processor cycles, but also unused disk space of shared machines, and a checkpointing-based mechanism.

Table 1 presents a summary of the aforementioned systems and algorithms. As shown, the initiatives approach different models for collaborative environments and assorted scheduling strategies. We can note that few works are focusing on metrics different of computation, as well as on failure control. In this regard, we observe a research opportunity to work with tightly-coupled applications, such as BSP, on collaborative environments, offering pertinent strategies to cover BSP features on highly dynamic and heterogeneous substrates.

3. BSPonP2P: Proposal to Run BSP Programs in P2P Desktop Grids

Considering both the Internet advances and the ease on acquiring computing resources (in this case, personal computers, tablets and smartphones), we observe the increase adoption of collaborative computing to run any kind of applications at a low financial cost. The simple idea is to profit idle CPU cycles, since the aforementioned resources are used mostly to access social networks, Internet queries and programs that consume a low computational substrate [9].

In this regard, we are proposing the BSPonP2P model to design how BSP applications would run in PDG efficiently. To accomplish this, BSPonP2P proposes a network overlay architecture, as well as strategies to turn viable the matching involving collaborative infrastructure and the BSP programming model. To accomplish this viability, we are exploring both process checkpointing and rescheduling. Checkpointing brings reliability and performance saving to the model: when someone leaves the system in a superstep then a checkpoint is used to restart the application in the last saved point. Rescheduling, in its turn, aims at covering dynamism, since both nodes and networks can become overloaded at application runtime; so, process can be on-the-fly migrated to novel locations to improve application performance. Particularly, rescheduling is highly pertinent in BSP programs, since they are composed by supersteps limited by a synchronization barrier in which the slowest process always bounds the parallel performance.

3.1 Network Overlay Architecture

BSPonP2P architecture was developed taking in mind both structured and unstructured P2P networks, as shown in Figure 1. Firstly, we are working with a structured ring-based network following the so-called Chord P2P protocol [11]. Chord uses a DHT and a Finger table to provide message exchange and routing in an efficient, scalable and secure way. This kind of network is used to connect nodes defined as Managers. We are using a timer denoted tto (time-to-organize) to reorganize the Managers in the Chord ring, aiming at optimizing communication latency among them. Each Manager is responsible for a specific cluster, where the cluster here means a parallel machine, a local network, a mobile device, or a single computer. Structured P2P networks aim at providing performance for large scale deployments [19], emphasizing our design decision for using them for Managers Interactions because of we plan to compose a worldwide scale architecture. A cluster, in turn, is organized in an unstructured manner. This decision was taken because this kind of organization offers better flexibility and dynamism with heterogeneous and unstable resources. The nodes inside a cluster are named End Nodes, or only Nodes, and they are responsible to execute the BSP applications actually.

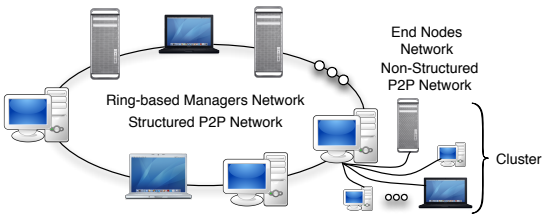


Fig. 1: Computational Overlay Network with two communication levels: (i) among the Managers; (ii) between a Manager and an End Node.

Each resource can act as a Manager or End Node. We created a Computational Overlay Network (CON) to manage message routing, scheduling, as well as the entrance and the leaving of a resource in the infrastructure. Each cluster i has a maximum of n_i End Nodes. This value is configurable and the administrator can change it according to network size. Thus, the first resource will act as a Manager and the others

Table 1: Comparison among initiatives to run parallel applications on collaborative environments.

Initiatives	Target system	Model application	Migration	Data Replication	Load balancing	Monitoring
Anceaume et al. [1]	PDG	Bags of Task	-	-	Computation	Computation
Balasubramaniam et al. [2]	Desktop Grids	Master/Slave	-	-	Computation	Computation
Camargo et al. [3]	PDG	BSP	yes	yes	Computation	Computation
Godfrey et al. [4]	PDG	Master/Slave	yes	yes	Computation	Computation
Khayyat et al. [8]	Desktop Grids	BSP	yes	no	Computation	Computation
Leite et al. [10]	PDG	Bags of Task	yes	yes	Work Stealing	Computation
Sentís et al. [14]	PDG	Master/Slave	yes	no	Computation	Computation and Memory
Shudo et al. [15]	PDG	Master/Slave	yes	no	Computation	Computation
Byung et al. [16]	Desktop Grids	Master/Slave	yes	no	Computation	Computation
Wu et al. [18]	PDG	Master/Slave	yes	yes	Computation	Computation

up to n_i will serve as End Nodes to the composed cluster i. After reaching n_i, another Manager is selected and then, other cluster is created. Upon entering the network, an End Node must report how much of their computing resource will be available to BSPonP2P. The resource's owner sets this parameter. By default, 100% of CPU is available when the user is not using the equipment or a percentage is employed, otherwise.

Aiming at getting End Node data periodically, a Manager sends query requests at intervals of t_i seconds. Each Manager defines t_i for cluster i at launching time. The queries are sended in a random Walk strategy. By definition, if the Manager does not receive a response from an End Node two consecutive times, then the Manager disconnects it from the CON. This procedure is executed to ensure that the Node is connected and able to execute a process from a future demand. Each cluster with less than n End Nodes is a candidate to receive the next incoming resource. Among them, the cluster with the lowest identification is actually chosen to host this new resource.

CON automatically reorganizes the network when a node, either a Manager or End Node, suffers a crash or intentionally leaves the collaborative infrastructure. In the case of a Manager, the oldest End Node in the cluster (comparing the stay time in the CON, and not other metrics like computational power since a Manager is responsible mainly for routing) is promoted to be the Manager. This new Manager is then updated with the data about cluster itself, supersteps in execution (if any) and the applications running in the resources under its responsibility. If an End Node crashes and it was executing supersteps of one or more applications, the Manager has partial data about the execution and can select other peer in accordance with the scheduling function to relaunch the application from the last saved checkpoint.

3.2 Scheduling and Runtime Strategies

This subsection describes how are deploying a BSP application in a P2P setting, showing also the runtime strategies to address performance and fault tolerance. We are targeting phases-based applications such as BSP, however the application model applied in BSPonP2P diverges from the traditional BSP [17], since this last one was firstly defined for homogeneous clusters.

The communication within the CON is divided into two levels, depending on the node's role: the first level comprises communication among the Managers, while the second level represents an interaction between a Manager and an End Node. After this introduction, the application launching occurs as follows. An End Node has a BSP demand and submits it to its Manager (the Manager just acts as organizer, it doesn't process the demand), informing the binary code and the number of process to run the application. Using the first-level communication, the mentioned Manager chooses the target cluster for each process.

The second-level communication is important to notify a Manager to choose an End Node under its responsibility to run a process. After selecting one End Node per process, an Execution Network is composed as depicted in Figure 2. It comprises a direct connection of each End Node (computing resource) to the Manager that started the BSP demand. This Manager will coordinate process communication and will pass the final result to the requester. The idea here is to save hops while performing communication actions among the BSP processes.

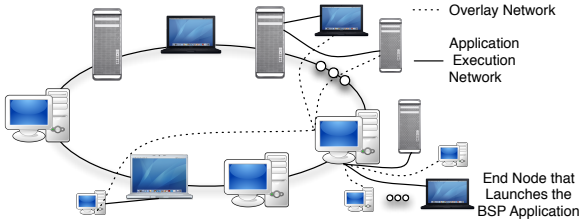


Fig. 2: Creation of an Execution Network, in which the Manager that receives the BSP demand acts as a gateway that directly communicates with all End Nodes involved in the BSP computation.

The evaluation of the first level will decide which cluster will execute a particular process. For that, we are using a decision function denoted PM (Potential of Migration) proposed by Righi et. al. [12] in the MigBSP proposal. PM is computed through Equation 1, which receives as inputs i and j, a process and a cluster, respectively. In this context, $Comp$, $Comm$ and Mem denote computation, communication and memory metrics, respectively. The larger the PM value, the most profitable is the target cluster j in receiving a process i. Each process is tested against each cluster and the largest PM value indicates the cluster for the process. Different from MigBSP, BSPonP2P uses PDG,

which implies in a modification of the computation metric in accordance with Equation 2. $T(i)$ and $Set(j)$ are inherited from MigBSP [12], and denote the computational time of process i in the last superstep and the relative performance of the cluster j, respectively. BSPonP2P adds $\overline{X}_{Resource}$ and $\overline{X}_{User}$ with the following objectives:

- **User**: here, $\overline{X}_{User(j)}$ is used to evaluate the users that either are running or have already ran applications in the resources of the cluster j in evaluation. If the users in the aforementioned context present a behavior of low usage of CPU power, this metric is close to 1. On the other hand, the value is close to 0 if the historical data does demonstrate a higher utilization of CPU cycles.
- **Resource**: The metric $\overline{X}_{Resource(j)}$ denotes an arithmetic average of the resources utilization in the cluster j. The idea is to work with a historical data in this metric, where close to 1 represents a lower usage of CPU or close to 0, otherwise. Thus, User metric analyses the pattern of access by the users, while the Resource observes the utilization degree of the resources.

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j) \quad (1)$$

$$Comp(i,j) = (\frac{\overline{X}_{Resource(j)} + \overline{X}_{User(j)}}{2}).T(i).Set(j) \quad (2)$$

Table 2 presents an example of how we are computing both User and Resouce metrics. The first column of this table refers to the captured observations on each cluster. In this scenario there are three users, named X, Y and Z, and two clusters, $C1$ and $C2$. $C1$ is composed by machines the A and B, while $C2$ includes machines C and D. Assuming the distribution among the clusters as presented in Table 2, the computation of the metric Users in cluster $C2$ will result 25% ($\frac{10\%+20\%+30\%+40\%}{4}$) while the Resource usage in cluster $C2$ will result in 15% ($\frac{10+20}{2}$). Note that the user Z does not have influnce in the User metric for cluster C2, because it is not performing any task in cluster C2. In the same way, Resource for C1 is 53.33% ($\frac{30\%+40\%+90\%}{3}$), while User for this cluster is 38% ($\frac{10\%+20\%+30\%+40\%+90\%}{5}$).

Table 2: Example of infrastructure usage, including 2 clusters and 3 users

Observation	User	Machine	Cluster	% of Machine use (CPU)
1	X	C	C2	10
1	Y	D	C2	20
2	X	A	C1	30
2	Y	A	C1	40
2	Z	B	C1	90

The evaluation with the second communication level is used to define which End Node in a cluster will run a specific process. The definition of the executor node is made based on the availability of the equipment. At this point, a simple assessment is made, where samples of at least three ratings and a maximum of ten reviews of availability (amount of computational resource available) are used. The samples are based on past records received by the Manager.

As runtime strategies, BSPonP2P offers process rescheduling and checkpointing. Both take place after ending a particular superstep, this point refers to a consistent global state of the distributed system. The idea is to offer a runtime management that aims at reducing the load imbalance among the processes, so decreasing the execution time of each superstep. Rescheduling tests are done not at each superstep, but the superstep index is defined in accordance with the MigBSP parameter called α. The system is launched with a predetermined value of α, which represents the interval of supersteps to evaluate process rescheduling. At each α supersteps rescheduling may occur if there is another most suitable cluster to execute a process according to PM function.

Aiming at dealing with dynamic environments, BSPonP2P profits from the phases-based organization of a BSP application to take a distributed snapshot of the application. This is done by saving a local checkpoint in each process, representing a basic BSPonP2P mechanism for addressing fault tolerance. The idea consists of not restarting the application from scratch in the presence of a node failure or outgoing user. Only data of the last superstep is saved, since all processes advance in a round-based fashion. This feature allows a time reduction if happens any crash in the system, for example, if anyone that is participating in a superstep leave the network (a Manager or End Node), the model have been projected to restart from the last point saved. The execution only lose a few supersteps and it doesn't need to restart from the beginning of the demand.

3.3 Observing Different Scenarios and Goals

The BSPonP2P's differential approach is highlighted by the adoption of process migration and checkpointing. Figure 3 illustrates different scenarios after running a BSP application using BSPonP2P. Scenario i represents the simple execution of a BSP application, disabling any service or scheduling functionality. The End Node-process mapping is fixed, being defined by the user.

Scenario ii adds the scheduling calculus in the first and second levels of the CON. Scenario i always outperforms scenario ii, since this last one adds dynamic scheduling overhead. Situations c, d, e and f represent the possibilities found in scenario iii. This scenario enables process checkpointing and rescheduling. Situations c and d present not suitable migrations, or yet, migrations were profitable but the number of remaining supersteps are not so large to get back the time in migration investment.

Both situations e and f represent the success in running BSPonP2P. Although situation e has a larger time when compared to situation a, it was computed using the check-

pointing strategy. Therefore, we can gain by not needing to restart the application from the first superstep in the case of a node crash. Process migration was responsible for a better resource usage and performance in the last situation.

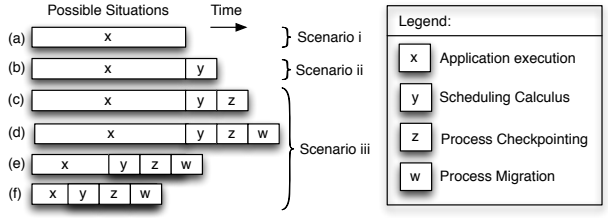


Fig. 3: Different execution scenarios of BSPonP2P. Both situations e and f of scenario iii are considered the main BSPonP2P's goal.

4. Evaluation Methodology

The evaluation was performed using SimGrid[1], a deterministic scientific instrument to study the behavior of scheduling algorithms in heterogeneous platforms, due to its high adoption in the scientific community. We applied simulation in three different scenarios using the Simgrid's MSG module: (i) Simple application execution; (ii) Application execution along with BSPonP2P scheduler without migration or checkpoint; (iii) Application execution with BSPonP2P scheduler with migration and checkpoint. The scenarios are graphically presented in Figure 3. The objective of the mentioned scenarios is to show the overload imposed by BPSonP2P (comparing scenarios i and ii), and the gain or loss of time when migration is enabled (comparing scenarios i and iii).

We also performed a recovery validation: in this case, we evaluated the time to resume the execution with checkpoint and compared with the time without this service. Without the checkpoint when a fault occurs, the system restarts from the beginning of the execution, instead of resuming from the last saved barrier as explained before. The comparison will be based on the exit of a host running a process in BSPonP2P.

We implemented a BSP application for computational fluid dynamics based on the principle of the Lattice Boltzmann Method (LBM) [13]. Each superstep is modeled by dividing the data in blocks, where each process performs a local computation using the block and, after that, sends updated data to its neighbor at the right. In order to perform the tests we allocated the first 15 nodes from each of the following Grid5000 clusters[2]: chimint and chicon located in Lille, paradent from Rennes, grephene from Nancy, gdx from Orsay, capricorne from Lyon, adonis from Grenoble, borderplage from Bordeaux, pastel from Toulouse and suno from Sophia, giving a total amount of 150 nodes.

[1]http://simgrid.gforge.inria.fr

[2]Details about computing resources and network connections can be found at http://www.grid5000.fr

Tests conducted in each scenario suffered the variation of three parameters: (i) α, which defines the interval of supersteps to perform the migration process starting with 4, 8 and 16 (same values used by [12]); (ii) Amount of supersteps whose values tested were 10, 50, 100, 500, 1000 and 2000; (iii) Amount of processes, assuming the values 11, 26, 51 and 89, randomly chosen to represent the environment that is found in PDG.

In order to represent the user interaction with the nodes and the variation of cluster availability we also created weight vector which represents the amount of computation available for the process to be executed. This values varied between 30 and 99 percent during all the executions changing at each superstep.

5. Discussing the Results

This section presents the results obtained when executing the LBM in the PDG varying the α value, number of supersteps and processes against each scenario. First we are going to evaluate BSPonP2P in the aforementioned scenarios without any machine leaving the CON. After, we evaluate the variation in the average time of supersteps and finally the checkpoint activation impact in the execution time when a machine leaves the CON.

5.1 Analysis of the Model Parameters

Analyzing the changes in the execution time according to the variation of the parameters presented above we can observe that the load imposed by BSPonP2P checkpointing and migration varies between an improvement of 6% with 26 processes, 2000 superstep and α equal to 16 and an overload of 17% with 89 processes, 100 supersteps and α equal to 4. Yet, in 76% of the cases the overload imposed was smaller than 5% in the total execution time. Also with α equal to 4 scenario iii is always better than scenario ii and with α equal to 8 scenario iii is better than scenario ii in all the cases with more than 11 processes.

The variation in execution time with different α values is better observed in Figure 4 where a comparison of the execution time between scenarios i and iii is presented. With the number of supersteps above 500 there is a decrease in the execution time, varying between -2.9% and -4.5% when α is equal to 4, -1.6% and -4.5% when α is equal to 8, and -3.8% to -5.5% when α is 16.

When evaluating the average time between each superstep, presented in Figure 5, we can observe a variation between 23.4 seconds with α 4 and 20.1 seconds with α 16. In some cases the migration do not present an improvement in the execution time, nevertheless with all α values there is an improvement in the average time between supersteps if compared to it's initial value (i.e. after the first migration).

The increase found in migration 3 and 4 in Figure 5 with α equal to 8 is given by the increased use of resources. The cluster elected to run the migration 4 had PM equal 2.89 and

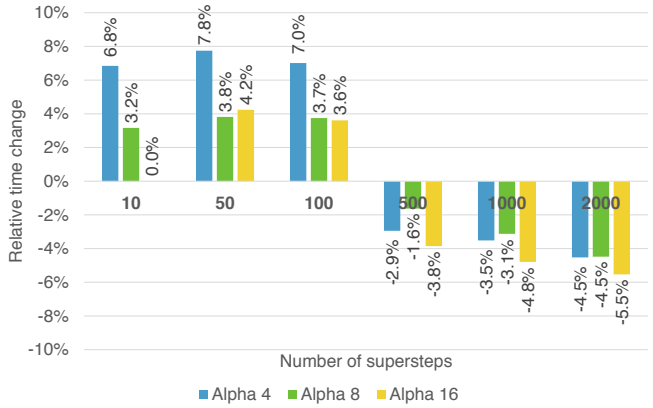


Fig. 4: Relative time variation of scenario iii when compared to scenario i varying the number of supersteps with 26 processes.

the average use of the cluster was 15% (18% of use among users and 12% utilization of the equipment). In the step that the migration 4 occurred the cluster PM which was running the previous superstep was 2.72. The reduction comparing the PMs was generated from the historical consumption of 18% found in the cluster that was running step 3, in this context the process was migrated to higher PM. Thus, the small increases found in Figure 5 are generated by differences in consumption of the user equipment and the network participants that occurred after the migration.

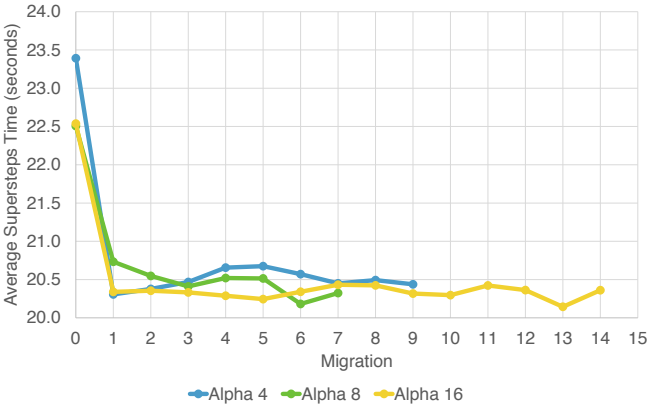


Fig. 5: Average time of supersteps in each migration with 26 processes varying the α value.

This dynamism in the resource utilization generated by the participants of the network leads to a tendency of unbalanced tasks. In Figure 6 we can observe that despite of alpha and amount of processes variation the migrations occurred along the entire execution.

This variation is also confirmed observing Figure 7 that shows the final tasks allocation with 51 processes and all α values. Despite of better computational resources of cluster Graphene (144 CPUs Xeon X3440, 16 GB memory and Infiniband-20G) when compared to Chicon (52 CPUs

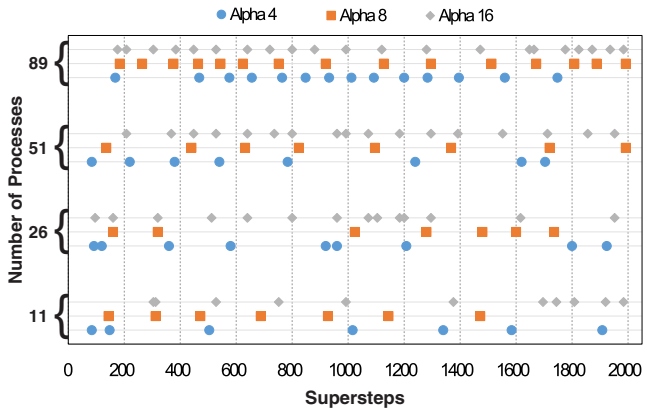


Fig. 6: Migrations distribution along the application execution varying the α value and number of processes.

Opteron 285, 4 GB memory and Myri-10G) for instance no migration pattern to this cluster can be detected. This behavior is highly related to Equation 2 which represents the computation resources usage by the user and equipments and is also used in migration calculus.

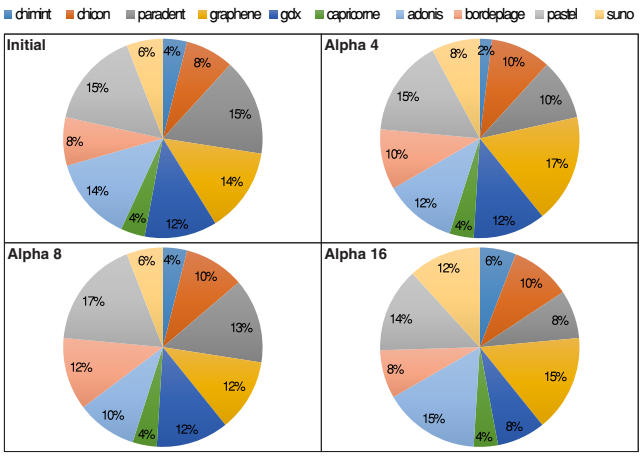


Fig. 7: Distribution of 51 processes among the clusters. The first graph indicates the initial distribution and the others the final distribution according to the α values.

5.2 Impact of the Checkpoint Activation

As mentioned earlier, we evaluated the overhead caused by BSPonP2P calculus, and the migrations occurred during an execution without any machine leaving the CON. In this experiment we analyze the recovery after an unexpected exit of a machine from the CON, since a user can leave the P2P network anytime. When there is no checkpoint the application must restart the execution from beginning, *i. e.*, scenario i, although with BSPonP2P whenever a migration occurs a checkpoint is saved and the application can restart from this point. In this context we evaluate scenario iii with 89 processes running, 2000 supersteps and α equal to 16, simulating an exit in different supersteps as can be seen in Figure 8.

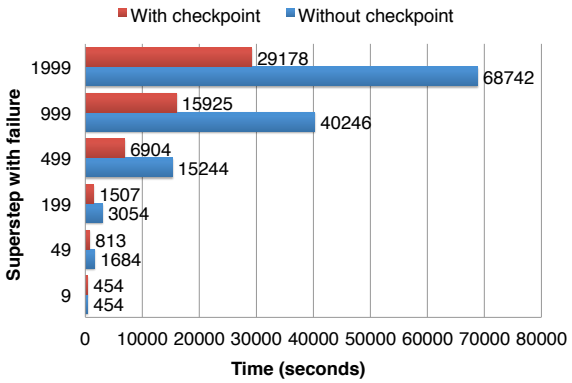


Fig. 8: Performance with and without checkpointing according to the supersteps with failure

Observing the results we can see that an exit in superstep 9 for instance did not cause any gain because there was no migration and consequently no checkpoint. On the other hand, when there is a bigger amount of superstep and a closer checkpoint, like the one occurred in the last case with an error in the superstep 1999 and the last checkpoint in the superstep 1016, an economy of more than 57% in time could be obtained.

6. Conclusion

This article presented BSPonP2P as an alternative to run BSP applications in PDG infrastructures. To the best of our knowledge, the proposed model is the first that joins the aforementioned programming model and the collaborative execution environment. Process rescheduling and checkpointing management is the BSPonP2P's scientific contribution. Thanks to both strategies, we demonstrated that the word "efficiency" referred in the problem statement means here performance and fault tolerance.

Besides presenting situations in which BSPonP2P outperforms the simple execution of a BSP application, most of the results using Grid5000 clusters showed an average overhead of 1.09% when using process rescheduling and checkpointing. We classify this rate as positive to BSPonP2P, because of an application must not be restarted from the scratch when any fault occurs (either when a node crashes or when an user sudden leaves the collaborative infrastructure).

Finally, we would like to emphasize that BSPonP2P is not restricted to BSP applications, but it can be used to manage any round-based computations in collaborative environments. Future research should evaluate BSPonP2P with process replication in order to launch copies of a process at specific superstep to run concurrently, so helping at both performance and fault tolerance areas.

Acknowledgment

This paper was partially founded by Santander and the following Brazilian agencies: CNPq, CAPES, FAPERGS.

References

[1] E. Anceaume, M. Gradinariu, A. Datta, G. Simon, and A. Virgillito. A semantic overlay for self- peer-to-peer publish/subscribe. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE Int. Conf. on*, pages 22–22, 2006.

[2] M. Balasubramaniam, N. Sukhija, F. Ciorba, I. Banicescu, and S. Srivastava. Towards the scalability of dynamic loop scheduling techniques via discrete event simulation. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th Int.*, pages 1343–1351, 2012.

[3] R. Camargo, F. Castor, and F. Kon. Reliable management of checkpointing and application data in opportunistic grids. *Journal of the Brazilian Comp Society*, 16(3):177–190, 2010.

[4] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *INFOCOM 2004. Twenty-third AnnualJoint Conf. of the IEEE Comp and Communications Societies*, volume 4, pages 2253–2262 vol.4, March 2004.

[5] F. P. Hargreaves, D. Merkle, and P. Schneider-Kamp. Group communication patterns for high performance computing in scala. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Functional High-performance Computing*, FHPC '14, pages 75–85, New York, NY, USA, 2014. ACM.

[6] B. Hendrickson. Computational science: Emerging opportunities and challenges. *Journal of Physics: Conf. Series*, 180(1):012013, 2009.

[7] K. Khan, K. Qureshi, and M. Abd-El-Barr. An efficient grid scheduling strategy for data parallel applications. *The Journal of Supercomputing*, 68(3):1487–1502, 2014.

[8] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proceedings of the 8th ACM European Conf. on Comp Systems*, EuroSys '13, pages 169–182, New York, NY, USA, 2013. ACM.

[9] E. Kijsipongse and S. U-ruekolan. Scaling hpc clusters with volunteer computing for data intensive applications. In *Comp Science and Software Engineering (JCSSE), 2013 10th Int. Joint Conf. on*, pages 138–142, May 2013.

[10] A. F. Leite, H. C. Mendes, L. Weigang, A. C. M. A. Melo, and A. Boukerche. An architecture for p2p bag-of-tasks execution with multiple task allocation policies in desktop grids. *Cluster Computing*, 15(4):351–361, 2012.

[11] L. Lin, K. Koyanagi, T. Tsuchiya, T. Miyosawa, and H. Hirose. Improving routing load balance on chord. In *Advanced Communication Technology (ICACT), 2014 16th Int. Conf. on*, pages 733–738, Feb 2014.

[12] R. d. R. Righi, L. Graebin, and C. A. da Costa. On the replacement of objects from round-based applications over heterogeneous environments. *Software: Practice and Experience*, pages n/a–n/a, 2014.

[13] C. Schepke and N. Maillard. Performance improvement of the parallel lattice boltzmann method through blocked data distributions. In *Comp Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th Int. Symposium on*, pages 71–78, Oct 2007.

[14] J. Sentís, F. Solsona, D. Castellà, and J. Rius. Discop2p: an efficient p2p computing overlay. *The Journal of Supercomputing*, 68(2):557–573, 2014.

[15] K. Shudo, Y. Tanaka, and S. Sekiguchi. P3: P2p-based middleware enabling transfer and aggregation of computational resources. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE Int. Symposium on*, volume 1, pages 259–266 Vol. 1, 2005.

[16] B. H. Son, S. woo Lee, and H.-Y. Youn. Prediction-based dynamic load balancing using agent migration for multi-agent system. In *High Performance Computing and Communications (HPCC), 2010 12th IEEE Int. Conf. on*, pages 485–490, 2010.

[17] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, Aug. 1990.

[18] D. Wu, Y. Tian, and K.-W. Ng. On the effectiveness of migration-based load balancing strategies in dht systems. In *Comp Communications and Networks, 2006. ICCCN 2006. Proceedings.15th Int. Conf. on*, pages 405–410, 2006.

[19] H. Zhao, X. Liu, and X. Li. A taxonomy of peer-to-peer desktop grid paradigms. *Cluster Computing*, 14(2):129–144, 2011.

Coarse Grained Parallel Algorithm for Hamiltonian Circuit in Convex Bipartite Graphs

Marco A. Stefanes[1], **Diego P. Rubert**[1], **and José Soares**[2]
[1]Faculdade de Computação, Federal University of Mato Grosso do Sul, Campo Grande-MS, Brazil
[2]Department of Computer Science, University of S. Paulo, S. Paulo-SP, Brazil

Abstract—*A bipartite graph $G = (V, W, E)$ is convex if there exists an ordering of the vertices of W such that, for each $v \in V$, the neighbors of v are consecutive in W. In this work, we address the Hamiltonian Circuit Problem, a well-known problem in Combinatorial Optimization. We present a novel sequential linear-time algorithm for determining a Hamiltonian circuit in convex bipartite graphs which can be easily parallelized. We also describe a coarse grained parallel algorithm for that problem which runs in time $O((|V|/p)\lg(|V|/p)\lg p)$, for p processors, using $O(\lg p)$ communication rounds. We also show how to efficiently implement our solution into PRAM and coarse grained parallel models. Our algorithm provides parallel scalability on commodity clusters. We have made experiments in a cluster composed of 64 processors, obtaining increasing speedups in our implementation. As far as we know, that is the first coarse grained parallel algorithm for the problem.*

Keywords: Scalable parallel algorithms, Convex bipartite graphs, Linear-time Hamiltonian circuit, Coarse grained parallel computing

1. Introduction

PRAM – parallel random access machine – is the main theoretical model of parallel computing. However, for many problems PRAM algorithms do not attain expected results in practice. One of the reasons is that fine grained PRAM model does not consider accordingly communication and computation costs. Features of commercially available parallel computers have shown we should take into account that communication costs are different of processing costs during the design of algorithms. Recently, several problems are implemented in more practical models which have been tested in cluster systems [1], [2], [3], [4], in cloud environments [5] and Hybrid environments [6], where it is possible to explore parallelism in multiple levels. An important feature of such parallel algorithms is its scalability, that is, the ability to handle efficiently a growing amount of work. We refer to such models as coarse grained parallel models. They are very appropriate to current parallel machines, since the computation speed is faster than communication speed in such machines.

Bipartite convex graphs were introduced by Glover [7], initially motivated by some industrial applications. Nowadays algorithms have been developed for this class of graphs in many applications, such as scheduling problems [8], DNA analysis [9] and constraint programming [10]. Several other applications of convex bipartite graphs were described in [11], [12], [13]. Sequential and parallel algorithms ([11], [8], [14], [15]) have been developed for finding a maximum matching in those graphs. A maximum matching is also used in [11], [16], [17] for determining a *maximum independent set* in convex bipartite graphs. In this paper, we use a maximum matching algorithm for finding a Hamiltonian circuit.

A *Hamiltonian circuit* in a graph $G = (V, E)$ is a simple circuit $(v_1, \ldots, v_n, v_1),\ n = |V|$, containing each vertex of V exactly once. If a graph contains a Hamiltonian circuit, then it is called a *Hamiltonian graph.* The problem of deciding whether a graph is Hamiltonian or not is a well-known topic in graph theory. The problem is NP-complete [18] even for bipartite graphs [19] or chordal bipartite graphs [20].

Let $G = (V, W, E)$ be a bipartite graph, where (V, W) is the partition of the vertices and E the set of edges. We say that G is *convex* if there is an ordering "$\leq$" of the vertices of W such that, for each vertex $v \in V$, the neighbors of v are consecutive in W. Considering the ordering of W, G can be represented by a set of $|V|$ pairs $\{(begin(v), end(v))|v \in V\}$, where $begin(v) \in W$ ($end(v) \in W$) is the smallest (largest) vertex of W adjacent to v. This is called a *compact representation* of G. Throughout this work convex bipartite graphs are given by their compact representation.

A goal of this work is to determine a Hamiltonian circuit in convex bipartite graphs. We present an $O(|V|)$ sequential algorithm for the problem and its version in PRAM and coarse grained parallel models. As far as we know, that is the first coarse grained parallel algorithm for the problem. Müller describes in [20] an $O(|V|^2)$ algorithm for the same problem. In a more general class of graphs, the *circular* convex bipartite graphs, Liang and Blum [12] present a sequential algorithm $O(|V|)$ for finding Hamiltonian circuits. However, our sequential algorithm has the feature of being easily parallelizable, as showed in this work. By experiments made in a cluster composed of 64 processors, we show the algorithm is scalable, since its performance improves proportionally to the growing in the number of processors. We obtained increasing speedups ranging from 0.71 to 3.92 in such machines.

The remainder of this work is structured as follows. First

of all, we specify the models of parallel computing in Section 2. In Section 3 we describe an $O(|V|)$ sequential algorithm for deciding whether a convex bipartite graph is Hamiltonian. The complexity and correction of the algorithm is presented in section 4. In Section 5 we detail how to implement the algorithm in the PRAM and BSP/CGM parallel models. Then, in Section 6 we report some experimental results. Finally, in Section 7, we present some conclusions.

2. Parallel Models

PRAM [21] is the best known parallel model, which is a straightforward generalization of the Von Neumann model. It is an idealized theoretical model of a shared memory MIMD machine. Although PRAM model is important from theoretical viewpoint, speedups obtained in practical implementations do not correspond to that expected in available parallel computers. There is a more practical model: BSP/CGM which is a coarse grained parallel model. In this model the communication between processors is done via an arbitrary network. Under the BSP [22] (*Bulk Synchronous Parallel*), computations are organized as a sequence of *supersteps* separated by synchronization barriers. The parameters of this model are: p, the number of processors; L, the minimum time of a superstep; and g, the quotient between speed of local computation and network bandwidth. The model CGM [23] (*Coarse Grained Multicomputer*) is a version of the BSP consisting of p processors with $O(n/p)$ local memory each, where $n/p \geq p^{\epsilon}$, for some $\epsilon > 0$. A BSP/CGM algorithm consists of local computation alternated with global communication. In a communication round, each processor sends and receives $O(n/p)$ data items.

3. Sequential Algorithm

The Algorithm Hamiltonian Circuit and its parallel versions are closely related to greedy maximum matchings. A *matching* M in a graph G is a subset of the edges of G such that no two edges in M are incident to a same vertex of G. A matching is *maximum* if its cardinality (number of edges) is as large as possible. A vertex v is *matched* by M if there exists an edge in M incident to v. A vertex v is *free* with respect to M if v is not matched by M. If there is no free vertex with respect to a matching M, it is called a *perfect matching*. The following concept, which uses the ordering "$\leq$" of W, has been shown to be very useful for determining a maximum matching in bipartite convex graphs.

Definition 3.1: A matching M in a bipartite convex graph $G = (V, W, E)$ is *greedy* if it has the following two properties:

1) if $(v, w) \in M$ and $v \in V$, then, for each $w' \in W$, with $begin(v) \leq w' < w$, there exists $v' \in V$ such that $(v', w') \in M$ and $end(v') \leq end(v)$;
2) if $w \in W$ is connected to a free vertex $v' \in V$, then there exists $v \in V$ such that $(v, w) \in M$ and $end(v) \leq end(v')$.

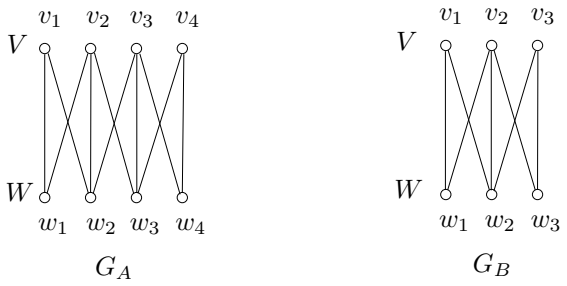


Fig. 1: Two Hamiltonian convex bipartite graphs illustrating the relationship between Hamiltonian circuit and perfect matching.

An important property of greedy matchings was discovered by Glover [7]: *every greedy matching is also a maximum matching*.

In Hamiltonian convex bipartite graphs we will show how to find a Hamiltonian circuit in convex bipartite graphs consisting, essentially, of the disjoint union of two perfect matchings. Although every Hamiltonian circuit in a bipartite graph is a union of two disjoint perfect matchings, it is not true that the union of two disjoint perfect matchings is always a Hamiltonian circuit. This fact is illustrated by the graph G_A in the Figure 1. The union of the two disjoint perfect matchings $\{(v_1, w_1), (v_2, w_2),\ (v_3, w_3), (v_4, w_4)\}$ and $\{(v_1, w_2), (v_2, w_1),\ (v_3, w_4), (v_4, w_3)\}$ is the union of two disjoint circuits. Depending on the choice of the first perfect matching, the remaining graph may not even contain another perfect matching. This fact is illustrated by the graph G_B in the Figure 1. The matching $\{(v_1, w_1), (v_2, w_2), (v_3, w_3)\}$ leaves the graph without another perfect matching.

As before we assume that a graph $G = (V, W, E)$ is given by its compact representation. We assume that the vertices of W are labeled is such a way that the sequence $w_1, w_2, \ldots, w_m$ are the vertices of W listed according to the ordering "$\leq$". We also define, for each $w \in W$, $next(w)$ as the vertex after w according to that ordering. Note that $next(w_m)$ is *undefined*. Let M be a matching in G. If $v \in V$ and $(v, w) \in M$, we denote by $\mathsf{M}(v)$ the vertex w. For $X \subseteq V$, we denote by $N(X) \subseteq W$ the set of vertices in W adjacent to vertices in X. Again, let n be $|V|$.

Algorithm Hamiltonian Circuit
Input: A graph $G = (V, W, E)$ in its compact representation.
Output: A Hamiltonian circuit if G is Hamiltonian.

(1) If $|V| \neq |W|$, then return "G is not Hamiltonian."
(2) Let v_1 be any vertex in V adjacent to w_1. $G_1 := G - v_1 - w_n$.
(3) Find a greedy matching M in G_1.
(4) If $|M| < n - 1$, then return "G is not Hamiltonian."
(5) $G_2 := G - w_1$.
(6) For $v_i \in V \backslash \{v_1\}$ do $begin(v_i) := next(\mathsf{M}(v_i))$.

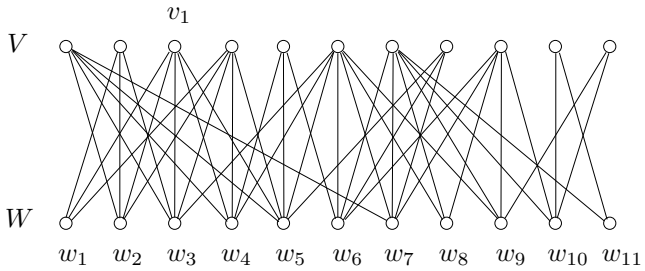


Fig. 2: Graph G with $|V| = 11$ showing the choice of v_1 by algorithm.

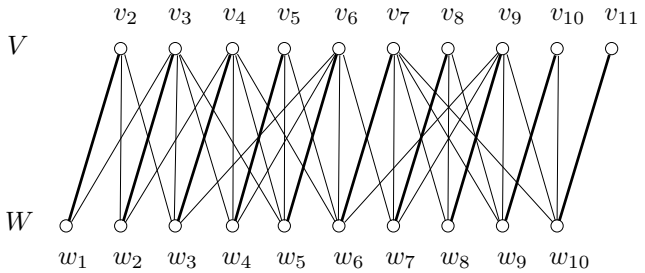


Fig. 3: Graph G_1. Bold lines are edges of M. V is labeled according to M.

If, for some v_i, $next(\mathsf{M}(v_i))$ is undefined, then return "G is not Hamiltonian."

(7) Find a greedy matching M' in G_2.

(8) Let $v_k \in V$ be a free vertex with respect to M'. If $|M'| < n-1$ or $w_n \notin N(\{v_k\})$, then return "G is not Hamiltonian". Else, return $M \cup M' \cup (v_1, w_1) \cup (v_k, w_n)$.

Figures 2, 3, 4 and 5 describe an example of a running of the algorithm beginning by the choice of the vertex v_1 in the Step 2, the greedy matching M found in G_1 in the Step 3, the greedy matching M' found in G_2 in the Step 7 and the resulting Hamiltonian circuit.

The Hamiltonian circuit found by the algorithm consists of either the circuit

$$(w_1, v_1, \mathsf{M}'(v_1), \mathsf{M}(\mathsf{M}'(v_1)), \mathsf{M}'(\mathsf{M}(\mathsf{M}'(v_1))), \ldots, v_k, w_n,$$

$$\mathsf{M}'(w_n), \mathsf{M}(\mathsf{M}'(w_n)), \mathsf{M}'(\mathsf{M}(\mathsf{M}'(w_n))), \ldots, \mathsf{M}(w_1), w_1)$$

or the circuit

$$(w_1, v_1, \mathsf{M}'(v_1), \mathsf{M}(\mathsf{M}'(v_1)), \mathsf{M}'(\mathsf{M}(\mathsf{M}'(v_1))), \ldots, w_n, v_k,$$

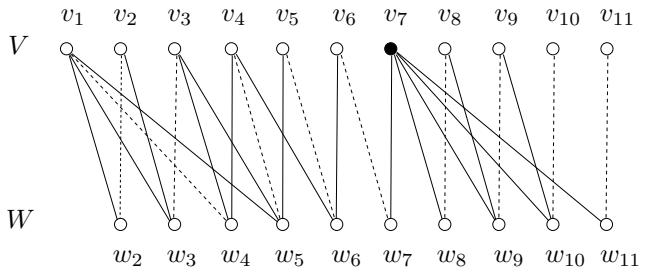


Fig. 4: Graph G_2. Dashed lines are edges of M'.

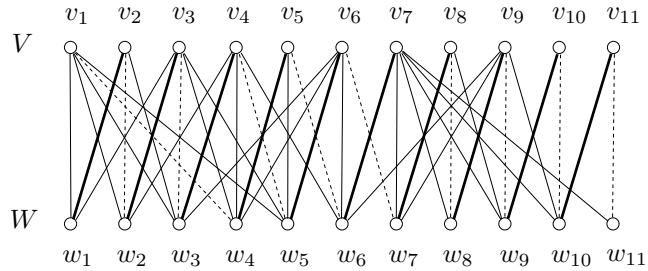


Fig. 5: Graph G. The Hamiltonian circuit consists of bold lines (M), dashed lines (M'), plus edges (v_1, w_1) and (v_7, w_{11}), where v_7 and w_{11} are free vertices with respect to M' and M, respectively.

$$\mathsf{M}(v_k), \mathsf{M}'(\mathsf{M}(v_k)), \mathsf{M}(\mathsf{M}'(\mathsf{M}(v_k))), \ldots, \mathsf{M}(w_1), w_1).$$

4. Correctness and Complexity

In this section, we demonstrate the correctness and time complexity of the Algorithm Hamiltonian Circuit. The following lemmas are useful to show the correctness of the algorithm.

Lemma 4.1: If $G = (V, W, E)$ is a Hamiltonian bipartite graph, then, for any proper non-empty subset S of V, we have $|S| < |N(S)|$.

Proof. Let C be a Hamiltonian circuit in G. Since S is a non-empty proper subset of V, considering only the edges in C incident to vertices in S, we find a neighborhood of S whose cardinality is larger than $|S|$. □

Lemma 4.2: If G is Hamiltonian, then G_1 contains a perfect matching.

Proof. Let C be a Hamiltonian circuit in G. Let P be the path from v_1 to w_n in C, and P' be the path from w_n to v_1 in C. Notice that, since the graph is bipartite, both P and P' have odd number of edges. The paths $P - v_1 - w_n$ and $P' - v_1 - w_n$ are vertex disjoint and their edges contain a perfect matching in $G_1 = G - v_1 - w_n$. □

For the sake of analysis, we consider next that vertices in V are labeled in such a way that for each i, $1 < i \leq n$, $\mathsf{M}(v_i) = w_{i-1}$.

Lemma 4.3: If G is Hamiltonian, then $\mathsf{M}(v_j) < end(v_j)$ for each j, $1 < j \leq n$.

Proof. By contradiction, suppose that G is Hamiltonian and there exists $1 < j \leq n$ such that $\mathsf{M}(v_j) = w_{j-1} \geq end(v_j)$. Since v_j is matched to w_{j-1}, there can only be the case that $\mathsf{M}(v_j) = end(v_j)$. If $N(\{v_1, \ldots, v_{j-1}\}) \subseteq \{w_1, \ldots, w_{j-1}\}$, we have, by Lemma 4.1, G is not Hamiltonian, a contradiction. So, we may assume that there exists $r < j$ such that $end(v_r) > end(v_j)$. We choose r as large as possible, in such a way that, for each i, $r < i \leq j$, we have that $end(v_i) \leq end(v_j)$. From the above observation, and remembering that M is greedy, it holds that

$begin(v_i) > \mathsf{M}(v_r)$ for each $r < i \leq j$. Hence, it follows that $N(\{v_{r+1}, \ldots, v_j\}) = \{\mathsf{M}(v_{r+1}), \ldots, \mathsf{M}(v_j)\}$, and, by Lemma 4.1 G, is not Hamiltonian. A contradiction. □

Lemma 4.4: If G is Hamiltonian, then each vertex in $W - \{w_1\}$ is matched by M'.

Proof. Since $M = \{(v_2, w_1), (v_3, w_2), \ldots, (v_n, w_{n-1})\}$ is a matching, using the Lemma 4.3 and the convexity of G, $M'' = \{(v_2, w_2), (v_3, w_3), \ldots, (v_n, w_n)\}$ is also a matching. Notice that M'' is a matching in G_2 that matches each vertex in $W - \{w_1\}$. Since M' is a maximum matching, M' will also match each vertex in $W - \{w_1\}$. □

If the graph G is Hamiltonian, then, after Step 7 of the algorithm, the graph G contains two matchings M and M', both of them with size $n-1$. Since in G_2 we have $|V| = n$ and $|W| = n - 1$, there exists exactly a free vertex in V with respect to M'.

Lemma 4.5: Let v_k be the free vertex of V in M'. If G is Hamiltonian, then v_k is adjacent to w_n.

Proof. By contradiction, suppose that v_k is not adjacent to w_n. Let q be such that $end(v_k) = \mathsf{M}(v_q) = w_{q-1}$. Since by Lemma 4.3 $end(v_k) > \mathsf{M}(v_k)$, we have that $k < q$.

First we show that, for each i, $1 \leq i < q$, $\mathsf{M}'(v_i) \leq \mathsf{M}(v_q)$. By construction of G_2, the only vertices in G_2 that can be adjacent to vertices in $W^* := \{w_2, w_3, \ldots, w_{q-1}\}$ are the vertices in $V^* := \{v_1, v_2, \ldots, v_{q-1}\}$. As v_k is one of those vertices and, by Lemma 4.4, all the vertices in $W - \{w_1\}$ are matched by M', M' induces a bijection between W^* and $V^* - v_k$.

By Lemma 4.1, we have that $|N(V^*)| > |\{w_1, w_2, \ldots, w_{q-1}\}|$. Hence, there exists an $r < q$ such that $end(v_r) > w_{q-1}$. Choose r with that property in such a way that j is maximum, where $w_j := \mathsf{M}'(v_r)$. As $end(v_k) < end(v_r)$, v_k is free in M' and M' is greedy, we have that w_j is not adjacent to v_k in G_2. Since $w_j = \mathsf{M}'(v_r) \leq \mathsf{M}(v_q)$ and the neighbors of v_k in G_2 are $w_k, w_{k+1}, \ldots, w_{q-1}$, we obtain that $w_j < w_k$.

We now show that $j = r$. Since $w_j = \mathsf{M}'(v_r)$, by construction of G_2, we have that $j \geq r$. If $j > r$, since v_j is adjacent to w_j in G_2 and M' is greedy, we can conclude that $end(v_j) \geq end(v_r)$. This would contradict the choice of r, because $\mathsf{M}'(v_j)$ would be larger than $\mathsf{M}'(v_r)$. So, $j = r$.

Notice that there exist $r - 1$ vertices smaller than v_r matched by M'. So, one of them is matched to some vertex not in the set $\{w_2, w_3, \ldots, w_{r-1}\}$. Let v_s be such vertex. It follows that $\mathsf{M}'(v_s) > \mathsf{M}'(v_r) = w_r$. Since v_s is not matched by M' to w_r, we have that $end(v_s) \geq end(v_r)$. Again, this contradicts the choice of r, showing that there exists no such r. A contradiction, proving the lemma. □

Using the lemmas above, we are ready to prove the following theorem. So, we conclude the algorithm correctly compute the Hamiltonian Circuit.

Theorem 4.6: The Algorithm Hamiltonian Circuit solves the problem of the Hamiltonian circuit in convex bipartite graphs.

Proof. We will show that the edge set $E_C := M \cup M' \cup (v_1, w_1) \cup (v_k, w_n)$ are exactly the edges of a Hamiltonian circuit in G.

Firstly observe that, by construction, (v_1, w_1) is an edge in G. The existence of the matchings M and M' and the edge (v_k, w_n) are guaranteed by Lemmas 4.2, 4.4, and 4.5.

By inspection, we can verify that E_C induces a subgraph of G where each vertex has even degree. Hence, E_C induces a collection of disjoint circuits in G. We need to show that E_C induces exactly one circuit, and, therefore, a Hamiltonian circuit.

Let C be one of the circuits induced by E_C. We shall show that the edge (v_k, w_n) is used by C. Since the choice of C is arbitrary and the circuits are disjoint, E_C has to induce exactly 1 circuit.

Since C contains at least 4 edges, the circuit C has to use at least one edge of M. So, we may consider that $C = (w_{j_1}, v_{i_1}, w_{j_2}, v_{i_2}, \ldots, w_{j_1})$, where $\mathsf{M}(w_{j_1}) = v_{i_1}$. We will proof that the sequence $j_1, i_1, j_2, i_2, \ldots$ is increasing, up to the edge (v_k, w_n) is used.

Recall that the vertices of V were labeled in such a way that for each i, $\mathsf{M}(v_i) = w_{i-1}$. So, it holds that $i_1 = j_1 + 1$, showing that $j_1 < i_1$. Next, we show that $i_1 \leq j_2$. If $(v_{i_1}, w_{j_2}) = (v_k, w_n)$, C uses the edge (v_k, w_n), as wished. So, we may assume that $(v_{i_1}, w_{j_2}) \neq (v_k, w_n)$. Since $i_1 > 1$, it cannot be the case that $(v_{i_1}, w_{j_2}) = (v_1, w_1)$. Hence, it is true that $\mathsf{M}'(v_{i_1}) = w_{j_2}$. As M' is a matching in G_2, where for each i, $begin_{G_2}(v_i) = next(\mathsf{M}(v_i))$, it holds that $\mathsf{M}'(v_{i_1}) = w_{j_2} \geq begin_{G_2}(v_{i_1}) = next(\mathsf{M}(v_{i_1})) = next(w_{i_1-1}) = w_{i_1}$. It follows that $i_1 \leq j_2$.

Applying the same idea of the above paragraph, we can show that $j_2 < i_2 \leq j_3$, unless the edge (v_k, w_n) is used. Since the circuit finishes in w_{j_1}, the sequence of indices cannot be always increasing and the edge (v_k, w_n) has to be used by C. □

The time complexity of the algorithm is dominated by the Steps 3 and 7, which consists in finding greedy matchings in a convex bipartite graph. Lipski and Preparata [11] presented an algorithm for finding a greedy matching in convex bipartite graph which can run in time $O(|V|)$, provided that a special version of the *union-find* algorithm is used [24]. Summarizing, we have the following theorem.

Theorem 4.7: The Algorithm Hamiltonian Circuit solves the problem of the Hamiltonian circuit in convex bipartite graphs in linear-time. □

5. Parallel Algorithms

The sequential algorithm from the previous section consists essentially of finding two greedy matchings in a convex bipartite graph $G = (V, W, E)$. Our parallel version consists in distributing the input graph uniformly among the processors. So, in steps 3 and 7 of the Algorithm Hamiltonian Circuit, we call a parallel greedy matching procedure. The remain of the algorithm stays the same as before. In order to do the steps 3 and 7 in the PRAM model, Dekel e Sahni [8] developed a EREW PRAM algorithm for the maximum matching problem that runs in time $O(\lg^2 n)$ using $O(n)$ processors, where $n = |V|$. In coarse grained parallel computing we can solve the maximum matching problem using the algorithm described by Soares and Stefanes [25] or the algorithm presented by Chan *et al.* [15] which use p processors, $O(\lg p)$ communication rounds, and spend $O((n \lg p)/p)$ time of local computations. The maximum matching found by those algorithms are greedy. Therefore, we have the following theorems.

Theorem 5.1: Given a convex bipartite graph $G = (V, W, E)$, the Hamiltonian circuit problem in G can be solved in the EREW PRAM model in time $O(\lg^2 |V|)$, using $|V|$ processors. □

Another solution for the Hamiltonian Circuit described in PRAM model and spending $O(\lg |V|)$ time, using $|V|^2$ processors is given by Bertossi and Bonuccelli [26].

Theorem 5.2: Given a convex bipartite graph $G = (V, W, E)$, the Hamiltonian circuit problem in G can be solved in the BSP/CGM model using p processors, $O(\lg p)$ communication rounds and $O((|V| \lg p)/p)$ time of local computations. □

6. Experimental Results

Our experiments were carried out in a high performance cluster at UFMS (Federal University of Mato Grosso do Sul) using 32 nodes. Each node is a 2.5Ghz quad core Xeon with 2GB of memory. Communication was carried out using a fat tree network implemented over an optical Myrinet switch with 10Gbits/s links.

6.1 Testing Graphs

For each convex bipartite graph $G = (V, W, E)$ we have used in tests, we generated its compact representation as follow. After choose the size of set W, for each vertex $v \in V$, we choose randomly the middle of the interval $[begin(v), end(v)]$ and with a previously defined density we determine the interval length derived from the Poisson distribution.

In order to test the implementation, we create graphs with intervals using density of 8%, i.e, we generated intervals with average length of 8% of the size of W. That value was chosen aiming to test some special details of the algorithm. We mean, in the Algorithm Hamiltonian Circuit, the interval density of 8% generates, in general, a Non-Hamiltonian graph and, with high probability, with only one matching the algorithm can decide whether the input graph is Hamiltonian or not. In another tests, we generated graphs using interval density of 45%, which enforce the algorithm to invoke matching procedure twice.

6.2 Results

The results related in this section were obtained from average time of ten executions of each instance. The size of the graphs was chosen aiming to equilibrate processing and communication costs, since we observed the algorithm speedup improves as the graph size grows.

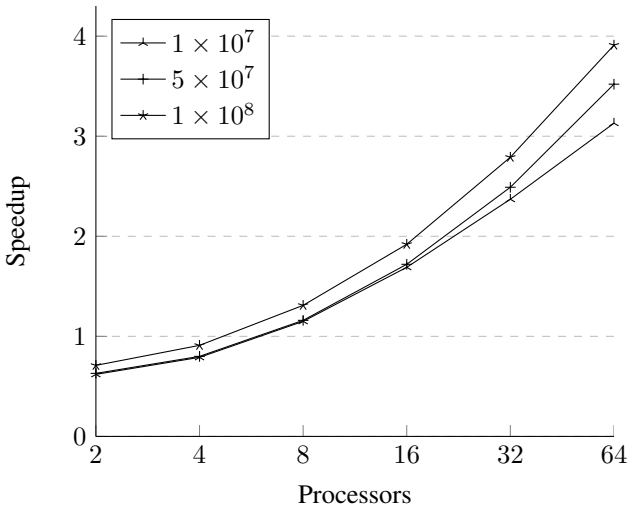


Fig. 6: Parallel algorithm speedups × number of processors for Hamiltonian Circuit, labels represent size of V.

Figure 6 shows the speedups of the algorithm in the cluster for graphs of sizes 1×10^7, 5×10^7 and 1×10^8. Speedups obtained by the algorithm ranged from 0.71 to 3.92 using 64 processors. We can see the algorithm is scalable when number of processors increases, although the speedup was less than 1 for 2 and 4 processors due to the cost of parallelization.

When sequential algorithm has time complexity $O(n)$ or $O(n \lg n)$, parallel version of this algorithm tends to have communication cost higher compared to local processing time, mainly when the communication rounds are increasing with the number of processors. The Hamiltonian Circuit Problem has that feature as we can see in Table 1, where we show the processing time compared to communication time.

In the Figures 7 and 8, we see the running time of the Hamiltonian Circuit algorithm for two different kinds of instances, respectively, generated graphs with density of 8% and 45% of the neighborhood for each $v \in V$.

Table 1: Processing time and communication time, in seconds, for Hamiltonian Circuit Algorithm where $|V| = 1 \times 10^7$, 5×10^7 and 1×10^8

	1×10^7		5×10^7		1×10^8	
	Proc	Com	Proc	Com	Proc	Com
2	31.7	8.1	170.3	41.9	309.1	80.6
4	21.0	10.3	113.0	54.0	204.1	101.4
8	12.9	8.6	70.0	45.8	126.0	85.9
16	7.6	7.1	41.3	36.7	74.3	69.6
32	4.4	6.5	23.8	30.1	42.8	56.5
64	2.5	5.4	13.7	24.4	24.7	46.0

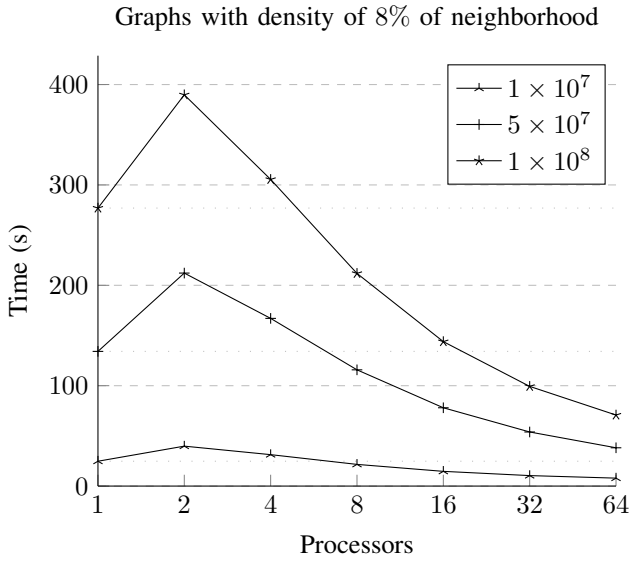


Fig. 7: Total running time in seconds × number of processors for Hamiltonian Circuit using interval density of 8%, labels represent $|V|$.

As shown in results, the running time of Hamiltonian Circuit Algorithm is basically the maximum matching running time. In the first case, displayed in Figure 7, the input graph is not Hamiltonian and the algorithm performs only one matching in order to make a decision, and in the second case, shown in Figure 8, the input graphs are Hamiltonian and the algorithm calls the maximum matching algorithm twice before making a decision.

7. Concluding Remarks

In this paper, we have investigated parallel solutions to the Hamiltonian Circuit Problem in convex bipartite graphs. We described a linear-time algorithm for finding a Hamiltonian circuit in $G = (V, E)$ that can be implemented under BSP/CGM model using p processors, $O(\lg p)$ communication rounds and $O((|V| \lg p)/p)$ time. We show through practical experiments our algorithm has good scalability. We also presented an algorithm where the Hamiltonian circuit

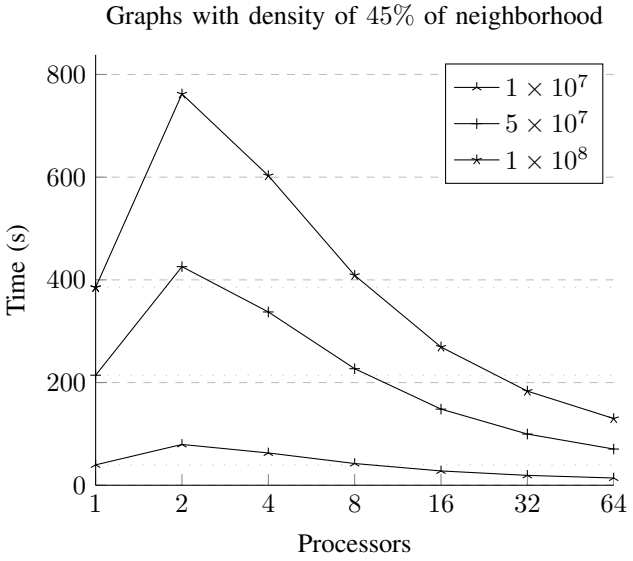


Fig. 8: Total running time in seconds × number of processors for Hamiltonian Circuit using interval density of 45%, labels represent size of V.

problem can be solved in the EREW PRAM model in time $O(\lg^2 |V|)$, using $|V|$ processors. A frequently considered hierarchy of graph classes is: permutation bipartite graphs $\subset$ doubly convex bipartite graphs $\subset$ convex bipartite graphs $\subset$ chordal bipartite graphs [27]. It would be interesting to develop parallel algorithms to the same problem for the class of chordal bipartite graphs. As future work, we could suggest the analysis of parallel Maximum Edges Bicliques Problem in this class of graphs. Besides, the study of the problem addressed here in more general classes of graphs, for instance, interval graphs, seem us appropriated.

References

[1] D. R. Higa and M. A. Stefanes, "A coarse-grained parallel algorithm for the matrix chain order problem," in *Proceedings of the 2012 Symposium on High Performance Computing*. Society for Computer Simulation International, 2012, pp. 1:1–1:8.

[2] C. E. R. Alves, E. N. Cáceres, and S. W. Song, "Finding all maximal contiguous subsequences of a sequence of numbers in O(1) communication rounds," *IEEE Transactions on Parallel and Distributed Systems, IEEE Computer Society*, vol. 24, no. 3, pp. 724–733, 2013.

[3] A. Ferreira, I. GuÃl'rin Lassous, K. Marcus, and A. Rau-Chaplin, "Parallel computation on interval graphs: algorithms and experiments," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 885–910, 2002.

[4] J. F. A. Vasconcellos, C. Nishibe, N. F. Almeida, and E. N. Cáceres, "Efficient parallel implementations of multiple sequence alignment using BSP/CGM model," in *Proc. of the 2014 International Workshop on Programming Models and Applications for Multicores and Manycores, in conjunction with PPoPP '14 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2014, pp. 103–110.

[5] A. Kraemer, J. C. d. Oliveira, F. A. G. d. Santos, A. C. Maciel, A. Goldman, and D. Cordeiro, "Dynamic creation of BSP/CGM clusters on cloud computing platforms," in *Emerging Intelligent Data and Web Technologies (EIDWT), Fourth International Conference on*. IEEE, 2013, pp. 767–772.

[6] L. G. Valiant, "A bridging model for multi-core computing," *Journal of Computer and System Sciences*, vol. 77, p. 154âĂŞ166, 2011.

[7] F. Glover, "Maximum matching in a convex bipartite graph," *Naval Research Logistic Quarterly*, vol. 14, pp. 313–316, 1967.

[8] E. Dekel and S. Sahni, "A parallel matching for convex bipartite graphs and applications to scheduling," *Journal of Parallel and Distributed Computing*, vol. 1, pp. 185–205, 1984.

[9] D. Nussbaum, S. Pu, J.-R. Sack, T. Uno, and H. Zarrabi-Zadeh, "Finding maximum edge bicliques in convex bipartite graphs," *Algorithmica*, vol. 64, no. 2, pp. 311–325, 2012.

[10] G. S. Brodal, L. Georgiadis, K. A. Hansen, and I. Katriel, "Dynamic matchings in convex bipartite graphs." in *MFCS*, ser. Lecture Notes in Computer Science, L. Kucera and A. Kucera, Eds., vol. 4708. Springer, 2007, pp. 406–417.

[11] W. Lipski and F. P. Preparata, "Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems," *Acta informatica*, vol. 15, pp. 329–346, 1981.

[12] Y. D. Liang and N. Blum, "Circular convex bipartite graphs: maximum matching and Hamiltonian circuits," *Information Processing Letters*, vol. 56, pp. 215–219, 1995.

[13] G. Steiner and J. S. Yeomans, "A linear time algorithm for maximum matchings in convex, bipartite graphs," *Computers and Mathematics with Applications*, vol. 31, no. 12, pp. 91–96, 1996.

[14] G. Gallo, "An $O(n \log n)$ algorithm for the convex bipartite matching problem," *Operations Research Letters*, vol. 3, pp. 31–34, 1984.

[15] A. Chan, F. Dehne, P. Bose, and M. Latzel, "Coarse grained parallel algorithms for graph matching." *Parallel Computing*, vol. 34, no. 1, pp. 47–62, 2008.

[16] A. Czumaj, K. Diks, and T. M. Przytycka, "Parallel maximum independent set in convex bipartite graphs," *Information Processing Letters*, vol. 59, pp. 289–294, 1996.

[17] J. Soares and M. A. Stefanes, "Algorithms for maximum independent set in convex bipartite graphs," *Algorithmica*, vol. 53, no. 1, pp. 35–49, 2009.

[18] M. Garey and D. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[19] M. Krishnamoorthy, "An NP-hard problem in bipartite graphs," in *SIGACT News*, vol. 7, 1975, p. 26.

[20] H. Müller, "Hamiltonian circuits in chordal bipartite graphs," *Discrete Mathematics*, vol. 156, pp. 291–298, 1996.

[21] J. Jájá, *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, 1992.

[22] L. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, pp. 103–111, 1990.

[23] F. Dehne, A. Fabri, and A. Rau-Chaplin, "Scalable parallel geometric algorithms for coarse grained multicomputers," in *9th Annual ACM Symposium on Computational Geometry*, 1993, pp. 289–307.

[24] H. N. Gabow and R. E. Tarjan, "A linear-time algorithm for a special case of disjoint set union," *Journal of Computer and System Sciences*, vol. 30, pp. 209–221, 1985.

[25] J. Soares and M. Stefanes, "BSP/CGM algorithm for maximum matching in convex bipartite graphs," in *15th Symposium on Computer Architecture and High Performance Computing - SBAC-PAD*, 2003, pp. 167–174.

[26] A. A. Bertossi and M. A. Bonuccelli, "Some parallel algorithms on interval graphs," *Discrete Applied Mathematics*, vol. 2, pp. 101–111, 1987.

[27] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

Associative Operations from MASC to GPU

Mingxian Jin
Department of Mathematics and Computer Science, Fayetteville State University
1200 Murchison Road, Fayetteville, NC 28301, USA

Abstract - *The Multiple Associative Computing (MASC) model is an enhanced SIMD (Single Instruction-stream Multiple Data-stream) model in the associative style for general parallel computation that has been studied last two decades. There have been a number of algorithms developed for this model. Recent research shows this model is extremely efficient when used for real-time scheduling in air traffic control systems. Associative operations are indeed key properties of this model. In particular, they all take constant time. In this paper, we present an implementation outline of these MASC associative operations on the popular architecture of GPU (Graphic Processing Units). This research aims to provide a bridge or general guidance to convert MASC algorithms to their GPU implementation. As the MASC architecture in today's technologies has not been built yet, this provides a possible way to implement MASC algorithms on an alternative platform so to verify their correctness and efficiencies especially for massive data input.*

Keywords: *MASC, Associative computing, GPU, SIMD*

1 Introduction

The Multiple Associative Computing (MASC) model is an enhanced SIMD (Single Instruction-stream Multiple Data-stream) model in the associative style for general parallel computation. It extends the concept of SIMD with associative properties and possesses features of easy programming and highly scalable. During last two decades, intensive research has been conducted regarding this model, mainly at Kent State University. It has been shown that this model is as powerful as some other well-known parallel computation models like PRAM (parallel random access machine) and restricted RM (reconfigurable meshes) [9]. There have been a number of algorithms developed on this model. These algorithms are across different application fields such as computer geometry, graphics, string matching, etc. Examples are in [1, 2, 4, 8]. Most recently, it has been shown that this model is extremely efficient when used for real-time scheduling in air traffic control system compared to its multiprocessor counterpart [12].

While more MASC algorithms are being developed, how to implement them becomes an interesting problem. Past effort has been made to build the MASC architecture, or ASC processor, using modern technologies of FPGS (field-programmable gate array) [10]. This research is still in the stage of experiment with up to 52 processing units. It is difficult to be used for any massively parallel processing, which is normally expected in a MASC algorithm. A standard associative language, called ASC, has also been developed for MASC across some platforms including Goodyear/Loral/Martin-Marietta's ASPRO and Thinking Machine's CM-2 [7, 8]. However, these platforms are not accessible in today's computer lab settings. Although the ASC programming language has been emulated on both PCs and workstations running UNIX to compile and execute simple ASC programs, the running environment is restricted by the emulating software and the non-associative style hardware construction. It is impossible to truly evaluate the performance of a MASC algorithm with massive data input.

We look for an alternative platform that is able to emulate the MASC model so to implement its algorithms in massively parallel with minimum efficiency loss. An ideal platform must have a close architecture and also be easily accessible. There is no doubt that GPU is an excellent choice.

A general- purpose computer with graphic processing units (GPU) is an emerging architecture that has attracted a lot of attention in last few years. The original purpose of using GPU is to accelerate intensive graphic data processing. Later, with introduction of NVDIA CUDA (compute unified device architecture), a high-level programming interface, GPU is evolved to be a powerful computing platform to support general purpose parallel computation. It has been used in numerous application fields for massively data parallel processing [6, 11]. GPU is a typical SIMD architecture and is especially good for fine-grained large amount data-intensive parallel computation. Its features provide possibility of implementing MASC algorithms with easy accessibility and high scalability.

In the MASC model, associative operations are indeed its key properties. To implement a MASC algorithm on a different architecture, we need to find a way to execute each of these operations in the corresponding running environment. This is our contribution in this paper. The remaining paper is organized as follows. Section 2 gives a description of the MASC model and the related research. Section 3 provides a

brief overview of GPU architecture with CUDA framework. Section 4 presents implementation steps for each MASC associative operation on GPU. Section 5 remarks general comparison between MASC and GPU to conclude the paper and also discusses future work.

2 The MASC Model

The MASC (for Multiple Associative Computing) model was developed at Kent State University based on earlier STARAN architecture at Goodyear Aerospace. It has been studied since the early 1970's. The MASC model consists of an array of processing elements (PEs) and one or more instruction streams (ISs), each of which issues commands to a disjoint set of PEs partitioned dynamically. In a MASC machine, the number of IS is normally expected to be small in comparison to the number of PEs. In this paper, we assume the MASC with one IS unless explicitly specified.

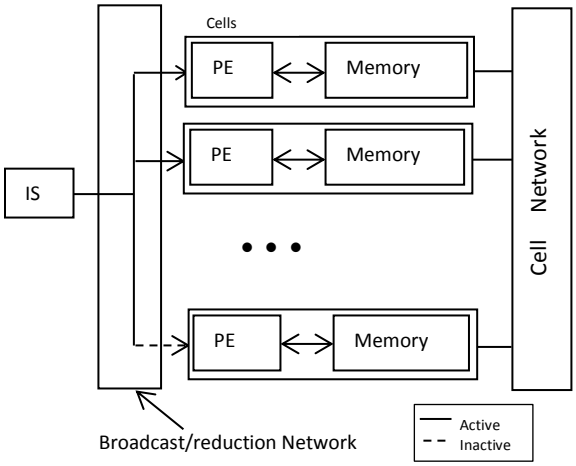


Fig. 1 The MASC model with one instruction stream

Detailed features of the MASC model can be found in [8]. A brief description follows. Each PE (or cell) has a local memory and is capable of performing the usual functions of a sequential processor other than issuing instructions. An IS is logically a processor which has a bus connecting it to each cell and can send an instruction to all cells. Each cell listens to only one IS and can switch to another IS based on local data tests when multiple ISs present. Cells can be active, inactive, or idle. An active cell executes the program steps from its IS while an inactive cell only listens. An IS can instruct an inactive cell to become active again (Fig 1).

If the word length is assumed to be a constant, then the MASC model supports the following associative operations in constant time. These have all been justified in [3].

- Global reduction of OR and AND of binary values each being held by an PE
- Global maximum and minimum of integer or real values each being held by an PE
- Associative search which finds all cells whose data value matching the search pattern. All data in the local memories of the cells is located by content rather than by address. These matching cells are called *responders* and those not are called *non-responders*.
- Pick-one which is used by the IS to select (or "pick one") arbitrary responder from the set of its active cells
- Broadcast which is used by the IS to instruct the selected cell to place a data item on the bus and all other cells listening to the IS receive this value in one step.

The MASC model may also include a cell network used for communications among PEs, an IS broadcast/reduction network (or the resolver network as another name) used for communication between the IS and cells, and a possible IS network in the case of multiple ISs that is used for IS communications.

A wide range of types of algorithms and several large programs have been developed for the MASC model and many of these have appeared in the literature. Examples are in [1, 2, 4, 8]. Moreover, simulations between MASC and other well-known parallel computation models such as PRAM and restricted RM have been well studied and published in the literatures as well. (See [9] for example). Most recent research has shown that this model is extremely efficient when used for real-time scheduling in air traffic control system compared to its multiprocessor counterpart [12].

As mentioned earlier, a standard associative language, called ASC, has been developed for MASC with one IS across certain platforms including Goodyear/Loral/Martin-Marietta's ASPRO, the WaveTracer, and Thinking Machine's CM-2, and provides true portability for parallel algorithms [7]. In addition, an ASC simulator has been implemented on both PCs and workstations running UNIX. It provides an efficient and easy way to test simple programs for algorithms designed for the MASC model. However, they cannot be used to truly evaluate performance of an MASC algorithm due to the restrictions of the emulating software and the non-associative style hardware construction.

Since the MASC model is developed based on early STARAN architecture that existed over 40 years ago, much effort has been made to build a new architecture based on today's technologies of FPGA so to support the MASC model and implement its algorithms. In [10], a scalable ASC processor with one IS using the FPGA technology is experimented with up to 52 PEs. It is still under study so difficult to be utilized for any MASC algorithm execution. It becomes a rising interest for us to find an alternative platform to implement MASC algorithms so to verify their correctness and efficiencies based on massive amount data input size.

GPU is an ideal choice which possesses features of easy accessibility and high scalability.

3 The GPU with CUDA framework

A general- purpose computer with graphic processing units (GPU) is an emerging architecture that has been attracted a lot of attention in the past decade. GPU was originally used to accelerate graphic computation and later evolved to perform general computation with introduction of high-programming languages and a specific framework CUDA (Computer Unified Device Architecture) as the programming interface. It becomes a very popular computing platform and has been used in numerous application fields [5, 6, 11].

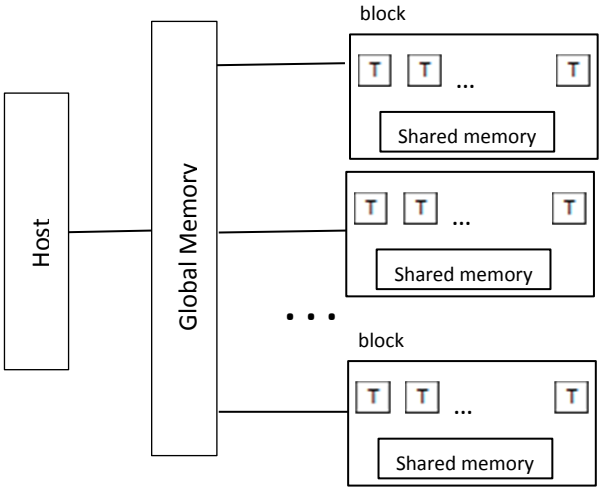


Fig. 2 Architecture of GPU with CUDA

A modern GPU consists of a host computer with an array of streaming multiprocessors forming groups of building blocks as the device. The host is normally considered to be a traditional central processing unit (CPU). In each streaming multiprocessor, there are a fixed number of streaming processors that execute the same instruction stream but running on different data sets. Each stream processor runs its own thread, as shown in Fig 2.

A CUDA-capable GPU supports several types of memory. Global memory and constant memory (not shown in the figure) can be read and written by the host such that data can be transferred between the host and the device. Each block has its shared memory that can be accessed by all threads. This provides an efficient way for threads within the block to communicate by sharing their input data and intermediate results during program execution. Each individual thread has registers as its locally accessed memory (not shown in the figure).

Commonly a GPU uses the host as the instruction steam to instruct multiple graphic processing units to perform data-intensive computation. This architecture is similar to the MASC model in that the host can be a multi-core CPU which can be corresponded to multiple ISs in MASC and many-core GPUs corresponding to MASC cells. Specifically, GPU executes in a SIMT (Single Instruction-stream and Multiple Threads) manner in which a thread is comparable to a MASC cell.

4 Associative Operations on GPU

Now implementation steps for MASC associative operations on the GPU platform are presented in this section. We map the data structures from MASC to GPU first. Then each associative operation is discussed one by one in regard of its implementation on GPU.

4.1 Mapping of the data structures

On MASC, data is stored by content instead of by address. In particular, data is organized in a tabular format with each PE holding a group of associative data. In an associative search, the search pattern is compared against the table of stored data in a bit-serial fashion. This allows search can be done in massively parallel and much faster. For example, a graph can be stored as a table structured as its adjacent matrix with each PE holding one row of data representing a vertex. In addition, other data pertaining to the vertex can also be stored on the same PE for parallel processing. Alternatively, one PE can hold data of one edge which includes weight, two end vertices, and/or other data depending on the application needs. Another example is that, in an air traffic control system, each PE holds all the data pertaining to an aircraft such as (x, y) positions, altitude, velocity, and so on.

GPU is in the traditional way to store data. A data item is identified by its memory address. In particular, for the device, a data item is stored in shared memory having its own memory address space or in the local memory of a thread that is addressed by its block index and thread index.

To simplify our discussion, we assume both ASC and GPU have sufficient numbers of PEs and device threads. In order to map associative data items from an MASC PE to a GPU thread, we can make a direct function map:

PE[k] → blockIdx (i).threadIdx(j)
Calculated by i = floor(k / blockSize) and
j = k MOD blockSize
where $0 \leqslant k \leqslant n$ for the MASC with n PEs and
$0 \leqslant i \leqslant$ blockSize -1 and
$0 \leqslant j \leqslant$ (number of threads in a block) - 1 on the GPU

All data that resides in local memory of a PE is mapped to the local memory of the corresponding thread. This provides a straightforward data mapping except identification number conversion.

4.2 Global reduction of OR and AND

Although CUDA provides `atomicOR` and `atomicAND` functions, these functions are used to perform logic OR and AND operations on the two words of data at a specified address in global memory or in the shared memory. They cannot be used for the global reduction of OR and AND on a group of data items as on MASC.

To perform a global reduction of OR on a group of data items on GPU, the n data items are assumed to be originally resided in the local memory of each thread. A global reduction is computed in the shared memory of the device with a block and then in the global memory among blocks. The final result is read by the host from the global memory.

We follow the idea in [5] for a global reduction of partial sum. Every two data items from two adjacent threads are reduced to one by moving the data items to the shared memory. All even-numbered threads perform this reduction in parallel which reduces the number of data items from *n* to *n*/2. Next round, every two data items from two adjacent even-numbered threads are reduced and the partial results are in the four multiple numbered threads. Iteratively, all data items will be reduced to one result as the global result. In order to avoid thread divergence [1], the data movements can be slightly changed by aligning the first half block and the second half block. All aligned threads are reduced in pair in parallel and partial results are stored in the first half threads corresponded shared memory. Iterations are going through in the same manner until the last result is obtained. The reduction is done in place, which means the data item in the shared memory is replaced by the partial result of logic OR. The CUDA-compatible code is shown as follows.

```
1. _shared_ boolean partialOR[ ];
2. unsigned int t = threadIdx.i;
3. for (unsigned int hop = blockSize.i >> 1;
        hop > 0; hop >> 1)
4. {
5.   _syncthreads( );
6.   if (t < hop)
7.       partialOR[t] = partialOR ||
                        partialOR[t + hop];
8. }
```

[1] Device divergence is a problem that occurs when some threads have to run a different instruction from other threads. It is usually caused by an *if-then-else* statement and degrades parallel processing performance.

After each block gets its last result, it is further sent to the global memory. The final result across all blocks can be obtained using a few more logic OR operations and then read by the host from the global memory.

Since this implementation uses software steps to emulate a hard-wired execution. It takes longer time by a factor of $O(\log n)$ for n threads, compared to the true MASC implementation.

It is obvious that a global reduction of AND can be done in the similar way.

4.3 Global reduction of maximum and minimum

On MASC, a global reduction maximum executes global OR in the bit-serial fashion. If the word length is ω, a global maximum takes $O(\omega)$. Since a word length is generally considered to be constant in all modern architectures, this is considered to be a constant operation as well. See [3] for detailed discussion.

On GPU, we will not use the bit-serial fashion as on MASC. This is because a global OR operation on GPU already uses programmed steps in its implementation. There is no need to perform the operation in bit-serial. The same approach as in Section 4.2 can be applied to implement a global reduction of maximum (or minimum). Partial maximum (or minimum) is updated by iterative comparisons. The previous intermediate larger (or smaller) data value is kept as the partial result for next round comparisons until the last result is obtained. We process data from all threads within a block in the share memory and then process all data from all blocks. The final result is sent to the global memory and read by the host.

Since this reduction operation goes through the same iterations as a global OR/AND as in section 4.2. It takes the same extra time in $O(\log n)$ for n threads.

4.4 Broadcasting

Broadcasting on GPU is fairly easy as there are different levels of memory providing read/write access for the host and/or all units on the device(s). The data item that needs to be broadcast can be placed by the sending thread in a specified shared memory location. Then blocks/threads can read from it directly. This does not take any extra time.

4.5 Associative search

Search on the ASC model is an operation combing broadcasting and global reduction of OR. On MASC, the IS broadcasts the search pattern to all PEs first. Each PEs compares this pattern with its local data. A PE with matching data sets a flag and is called a responder. Otherwise, the PE resets the flag and is called a non-responder. A global

reduction of OR is performed and sends the search result back to the IS. A global reduction of OR on the designated flags can be performed on GPU as described in 4.2. The final result of 1 or 0 indicates the search is successful or unsuccessful.

After an associative search, all responders are normally instructed to do specified tasks and non-responders are set inactive to wait for future activation. On GPU, this can be executed by running these specific tasks on corresponding threads and set other threads idle without doing anything until they are activated later. It is similar to executing an if-then-else statement on GPU for any general computation. Once again, device divergence could be a potential problem.

The time complexity is dominated by the time of the global reduction of OR, which is $O(\log n)$, as broadcasting and setting flags only take constant number of steps by all threads in parallel.

4.6 PickOne

An associative search may also be followed by a PickOne operation. An arbitrary PE (or a thread on GPU), normally the first responder in the responder group, is selected to perform individually instructed tasks. Using the function of `CUDAGetDevice` provided by the CUDA framework on GPU, this can be implemented. It takes the same time as the function.

It is noted that responder execution after a PickOne operation can cause the device divergence problem as mentioned in 4.5. Depending on algorithm design, we may expect different levels of performance degradation due to this problem.

5 Concluding remarks and Future Work

MASC and GPU present many similarities. Both of them are in the SIMD category (SIMD vs. SIMT) in parallel computation. They are especially advantageous in fine grained parallel processing for massively data-intensive problems. They both have an ability of restrictive control parallelism with multiple ISs and the multi-core CPU, respectively. Both of them possess features of easy programmable (ASC vs. CUDA, respectively). They are also highly scalable because the array of cells and threads can be easily extended. Normally there are light or no overheads due to single instruction stream.

The two architectures have some differences as well, however. There are constructed significantly different in hardware. MASC is a bit-serial model. On the other hand, GPU assumes each processing unit in a traditional manner. MASC allows cells communicate through the cell network. On GPU, there is no separate network connecting blocks and threads. All communications are through different levels of memory reads/writes. It is fast but lacks flexibility to dynamically partition threads. Threads within a block can be collaborated. However, threads across blocks cannot. In a MASC algorithm, an advantage to represent complex data structures is tabular representation. All data on a PE is stored and located by its content instead of by its address. This allows fast locate data and return the search results in constant time through the hardware construction – the resolver network. However, GPU locates data by memory address, in particular, block indices and thread indices.

With these similarities and differences in mind, in this paper, we have presented outlined implementation of associative operations of the MASC model on the GPU architecture. As shown in Section 4, most of these associative operations can be implemented on GPU with an extra $O(\log n)$ efficiency loss. This is due to the fact that software programmed steps on GPU are used to replace hardware wiring as the resolver (broadcast/reduction) network on MASC. These steps can be directly used to convert a MASC algorithm so to be implemented on the GPU-CUDA platform.

Due to time limit and space constraints, the actual implementation results and performance analysis are not included in this paper. Future work is to analyze these results and evaluate their performance. Also, for some problems, if there exist an algorithm directly designed on GPU and it also has a MASC algorithm, it would be interesting to compare the performance of the GPU algorithm and the converted MASC algorithm running on GPU.

6 Acknowledgement

This work is partially supported by the FSU Integrated STEM Academic Success (ISAS) program.

7 References

[1] M. M. Atwah and J. W. Baker "An associative static and dynamic convex hull algorithm", in Proc. of the 16th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing), abstract on page 249, full text on CDROM, April 2002

[2] M. Esenwein and J. Baker, "VLCD string matching for associative computing and multiple broadcast mesh", in Proc. of the IASTED International Conference on Parallel and Distributed Computing and Systems, pages 69-74. 1997

[3] M. Jin, J. Baker, K. Batcher, "Timings for associative operations on the MASC model", in: Proc. of the 15th International Parallel and Distributed Processing Symposium, IEEE Workshop on Massively Parallel Processing, San Francisco, CA, 2001, pp. 193–200

[4] M. Jin, J. Baker, "Two graph algorithms on an associative computing model", in Proc. Of International

Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA, Las Vegas, 2007.

[5] D.B. Kirk and W. W. Hwu, Programming Massively Parallel Processors, Morgan Kaufmann Publishers, 2010

[6] I. Park, N. Signhal, M. Lee, S. Cho, and C. Kim, "Design and performance evaluation of image processing algorithms on GPUs", IEEE Transactions on Parallel and Distributed Systems, Vol. 22, No.1, January 2011

[7] J. Potter, Associative Computing: A Programming Paradigm for Massively Parallel Computers, Plenum Press, New York, 1992

[8] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, C. Asthagiri, "ASC: an associative-computing paradigm", Computer 27 (11) (1994) 19–25

[9] J. Trahan, M. Jin, W. Chantamas, J. Baker, "Relating the power of the multiple associative computing model (MASC) to that of reconfigurable bus based models", Journal of Parallel and Distributed Computing, Elsevier Publishers, Vol. 70, No. 5, (2010) 458–466

[10] H. Wang, and R. Walker, "Implementing a scalable ASC processor", in Proc. of the 17th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing), April 2003

[11] W. W. Hwu, GPU Computing Gems Emerald Edition (Applications of GPU Computing Series), Morgan Kaufmann; 1 ed. February 2011

[12] M. Yuan, J. W. Baker, W. C. Meilander, "Comparisons of Air Traffic Control Implementations on an Associative Processor with a MIMD and Consequences for Parallel Computing", Journal of Parallel and Distributed Computing, Elsevier Publishers, Volume 73, Issue 2, February 2013, pages 256-272, ISSN 0743-7315

Implementation and Analysis of Parallelized Binary Decision Diagram Manipulation on Multicore Processors

Myoung Ha Kim, Jong Kang Park, and Jong Tae Kim

Department of Electrical and Computer Engineering
Sungkyunkwan University
Suwon, Korea

Abstract – *As multi-core or many-core era has emerged, multi-core programming became important to System-on-a-chip designs. We applied the parallelization tools to Binary Decision Diagram (BDD) manipulation and analyzed the performance result using profiling tools. BDD is useful data structure for synthesis and verification of circuits in CAD software. Many studies made fault propagation model using Binary Decision Diagram (BDD). We implemented BDD manipulation program based on previous researches and parallelized the program. Using a profiling tool, Intel Vtune Amplifier XE 2015, we observed the performance of the program and analyzed experimental results according to parallelization. Finally, we suggest future work with consideration of analyzed result.*

Keywords: Multi-core programming, Parallelization, Binary Decision Diagram

1 Introduction

Since 2005, raising CPU core frequency has been difficult due to high power consumption and overheating. As a result, Single Instruction Multiple Data (SIMD) process and multi-core or many-core system became main ways for increasing the performance of applications. In this perspective, multi-core programming is becoming more important to commercial system-on-a-chip designs. In recently emerging issues like cloud computing and Internet of Things, multi-core programming is in a general trend for new technology developments. Some APIs which help programmers to implement the multi-core program are being provided. POSIX thread, OpenCL, OpenMP and Message Passing Interface (MPI) are typical APIs for multi-core programming [1]. GPU is also a good example of multi-core system. The original purpose of GPU is image processing, but GPGPU which takes advantage of GPU for general purpose exists [2] and Compute Unified Device Architecture (CUDA) platform is provided for GPGPU programming. However, there are some limitations for fully exploiting the resources of multi-core system. The representative cases are synchronization issues in shared memory or race condition problems.

This paper presents the analysis of a parallelized application which deals with Binary Decision Diagram (BDD) in the digital circuits [3]. Some studies used BDD for circuit fault expectation model [4][5]. However, as the complexity of the circuit rises, the size of BDDs exponentially increases and the execution time of the program also increases. To compensate this disadvantage, we parallelize the procedure of BDD manipulation and analyze the problems after observing the amount of the performance improvement.

In this paper, we introduce how to analyze circuit faults using BDD in Section 2. Section 3 briefly explains parallelization and analysis tools. Section 4 presents how to implement BDD manipulation program. Parallelization of the program and analysis of the results are shown in Section 5. We conclude our tasks and suggest future work in Section 6.

2 Circuit fault analysis using binary decision diagram

Our case study is BDD manipulation program for circuit analysis. As a circuit fault expectation model, we used the results of the previous works utilizing the symbolic frameworks based on binary decision diagram [4][5]. Binary Decision Diagram (BDD) is a useful data structure for representing boolean functions and related manipulation algorithms [6]. In this approach, BDDs have logical property besides electrical property of the target circuit, so more accurate fault rate calculation is possible.

Figure 1 shows the generation of the voltage pulse for a transient fault when a noisy problem occurs during circuit operation. Prior to the generation of the transient fault, the terminal nodes of BDD only include logical 0 or 1 value and it is called static BDD. On existence of noise sources, the transient fault occurs according to cell library on input bias conditions. Bottom part of Figure 1 reflects this property and this type of BDD is called event BDD [4].

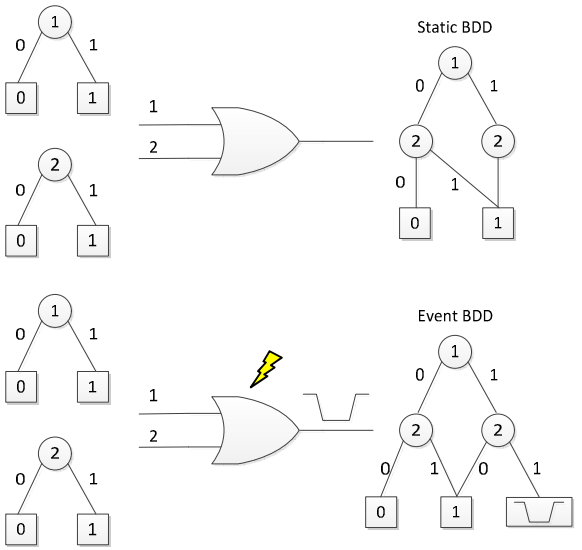


Figure 1. Difference of static BDD with normal condition and event BDD generated by the transient fault with bias condition '11'.

When some static BDDs or event BDDs are combined on a certain logic gate, the BDD construction function should be executed as proposed in [6]. Only the process of handling the terminal nodes which contain voltage pulse information of event BDD is different. For example, if terminal nodes of each BDD are combined in the construction process and those terminal nodes have transient faults, output terminal node would be generated based on the corresponding cell library information. The example of this process is shown in Figure 2. The error pulse propagates to the primary output or attenuated, and finally, the fault probability can be calculated with the BDD on the primary output. This process will be executed iteratively on the all of the internal circuit nodes and the variation of noisy conditions.

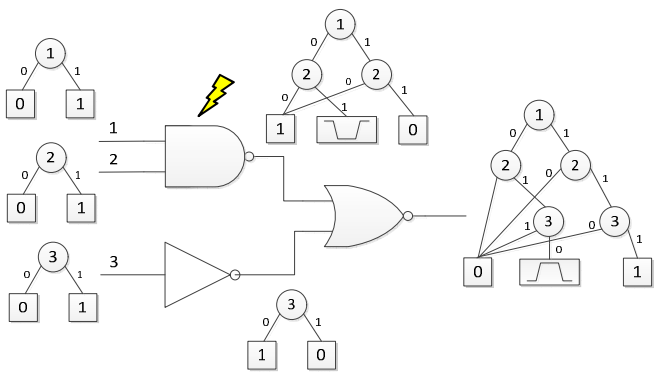


Figure 2. An example of the fault generation and propagation on BDDs.

3 Parallelization and analysis tools

We exploited OpenMP API to parallelize the problem of BDD manipulation, and Intel Vtune Amplifier XE 2015 for analyzing parallelization result.

3.1 OpenMP

OpenMP is an API which contains compiler directives and runtime library routines for parallelism in shared memory system [7]. OpenMP is consistent and portable interface which makes parallelization on various architectures and systems. In parallelized program built by OpenMP, master thread executes the program alone before joining the #pragma directives. After joining the #pragma directives, the master thread creates other threads which are able to progress in parallel. After the works in parallel region are finished and the implicit synchronization is completed, then master thread destructs other threads of operation. The notable advantage of OpenMP is simple implementation and it can make programmer modify just a few amount of original code.

3.2 Intel Vtune Amplifier XE 2015

Intel Vtune Amplifier XE 2015 provides much information for tuning the program. It collects the data for analyzing hotspots, threading, locks and waits. Also, it helps profiling through hardware event-based collection using performance monitoring unit. By this profiled information, it shows us CPI, last level cache miss, branch misprediction, memory latency and etc.

4 Implementation

We implemented the BDD manipulation code and its parallelized program. The entire procedure is illustrated in Figure 3. At first, the code parses synthesized gate-level Verilog HDL code and represent the circuit as a graph. Then the software reads the cell library to handle the electrical property of the faults. To make static BDDs along the fault propagation paths, topological sort is conducted. The static BDDs are used for generating event BDDs during the subsequent fault propagation stages. After making static BDDs completed, propagation process on each circuit node can be executed. The box marked as "Propagation" in Figure 3, generates the transient fault and the corresponding event BDD at a certain circuit node and propagates event BDDs to the primary outputs or the inputs of the flip-flops. The percentage values next to some procedures are the proportions of the total execution time. The construction and reduction functions for BDDs [6] which manipulate BDDs occupy almost all execution time of the program.

The parallelizable region of this program is clear. The procedure that generates and propagates the error pulse is a hotspot and almost independent work, thus propagation box can be parallelized. The separated threads taking each node

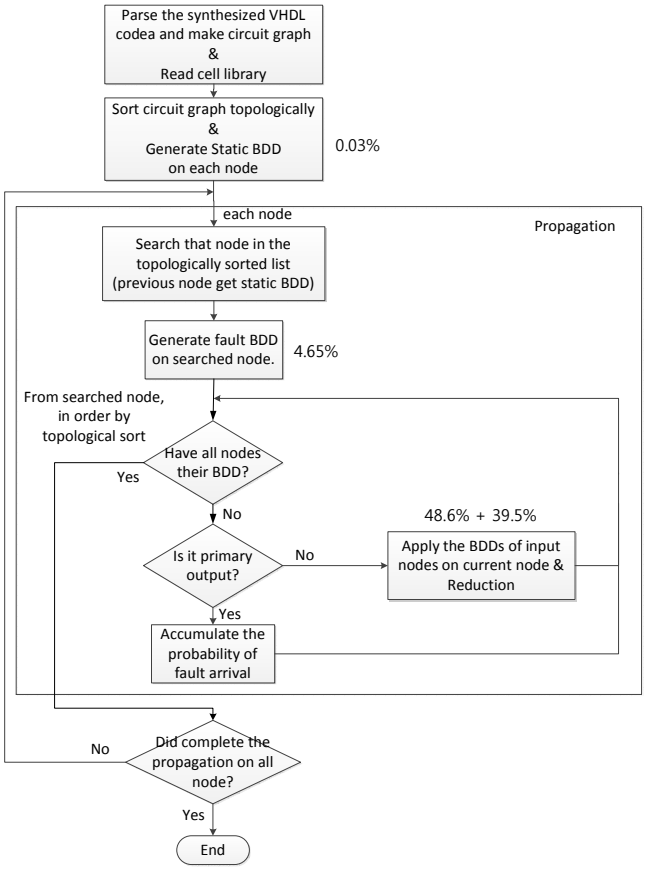


Figure 3. Flow chart for BDD manipulation program

share the data of static BDDs and the probability of fault arrival on primary outputs. Static BDDs are needed to all threads, but threads only read the data so there is no need to synchronization. The data storing the probability of fault arrival value on primary output is written by all threads so this action should be atomic. OpenMP provides some constructions like critical or reduction for these type of actions.

Our application is executed on following environment. CPU is Intel Xeon CPU E5-2687W 3.10GHz double octa-core. Each core has 32KB L1 cache, 256KB L2 cache, and one octa-core has 20MB L3 cache. The size of main memory is 8x8GB.

5 Result and analysis

5.1 Result of parallelism

BDD manipulation is used for Soft Error Rate (SER) estimation of the gate-level digital designs [4]. Because the individual fault generation can be fully parallelized in this problem, the ideal execution time is identical to the execution time of sequential program divided by the number of threads. Figure 4 represents the effect of parallelism. The values of the

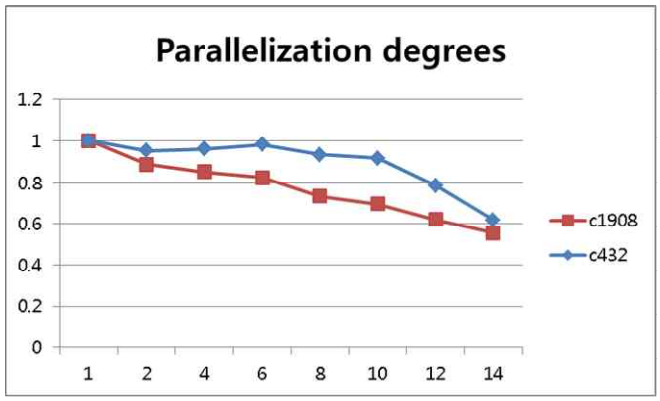


Figure 4. Parallelization degree of c432 and c1908 benchmark circuit.

graph are generated by a quantity called as parallelization degree as follows.

$$paralleli\ zation\ \ \deg ree = \frac{the\ \ real\ \ execution\ \ \ time}{the\ \ ideal\ \ execution\ \ \ time} \qquad (1)$$

Therefore, it is better for these values to be closer to 1. But the values start to decrease when the number of threads is more than a certain number.

5.2 Analysis of the result

We obtained several metrics of the program with Intel Vtune amplifier XE 2015 as shown in Figure 5. CPI and unfilled pipeline slots are measured by hardware event-based sampling with performance monitoring unit.

At first, we can see that CPU utilization is uniform, regardless of the number of simultaneous threads. It means all cores taking each thread are busy with no rest. However, the values of CPI and unfilled pipeline slots are not constant. When the number of threads exceeds 10, it starts to increase. Especially unfilled pipeline slots in front-end do not change much, but unfilled pipeline slots in back-end which are the part of subsequence of instruction fetch increase. Figure 6 also represents similar results. CPI and unfilled pipeline slots in back-end also starts to increase from eight threads.

These results correspond to the result of parallelization degree. The result that CPU utilization is consistent and the reason of an increase in CPI is due to unfilled pipeline slots in back-end, can be interpreted in the terms of latency for memory access. Because all cores cannot access simultaneously last level cache or main memory, as the number of threads which try to access low level memory increases, the threads should wait before the other threads finish the accesses. The reason that the points where parallelization degree starts to decrease are different can be analyzed as because of the size of shared data. We can expect that the difference of the points where CPI and unfilled pipeline slots in back end decreases is due to same reason.

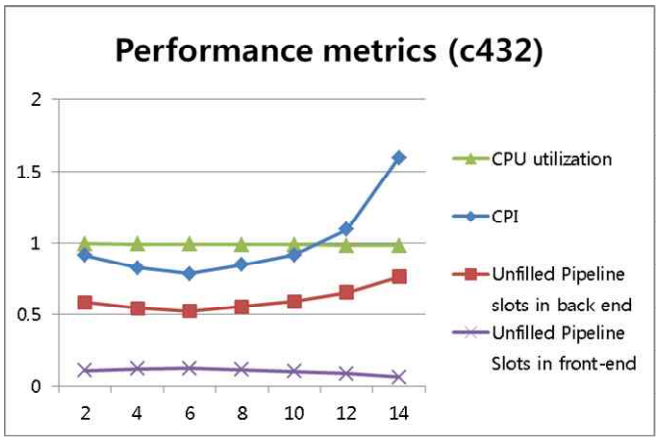


Figure 5 Performance metrics of the parallelized program for c432 benchmark circuit.

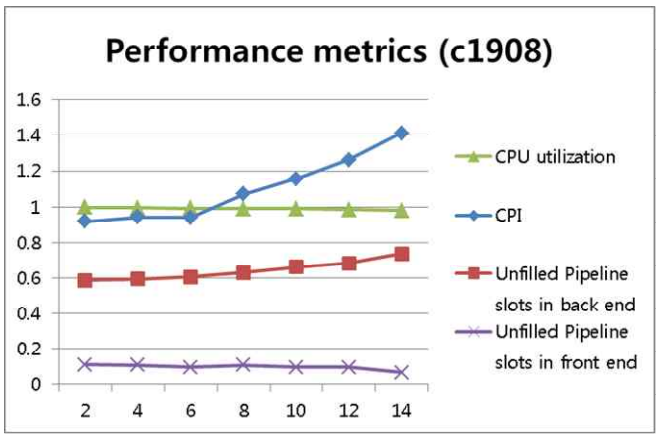


Figure 6 Performance metrics of the parallelized program for c1908 benchmark circuit.

6 Conclusions

In this paper, we observed parallelization results of the BDD generation and propagation algorithms in a case study for soft error rate estimation. We analyzed the effects of the parallelization with performance metrics. For propagating transient faults along the circuit nodes, we utilized BDD structure and the existing construction and reduction methods. We implemented this structure and the parallelized program with change of the number of threads. As increasing the number of threads, we could observe some tendencies of the performance metrics. The observed characteristics represent uniform CPU utilization and increasing CPI and unfilled pipeline slots in back-end. These could be interpreted as the increase in the latency of memory.

Our simple parallelization method occupy so much memory so parallelization effect starts to decrease when the number of threads exceed certain the number. Our future work will modify parallelization strategy focusing on the use of memory.

Acknowledgment

This research was supported by Basic Science Research Program (NRF-2013R1A1A2060954) funded by the Ministry of Education.

References

[1] J. Diaz, C. Munoz-Caro and A. Nino, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," IEEE Trans. on Parallel and Distributed Systems, Vol. 23, pp. 1369-1386, 2012.

[2] M. Macedonia, "The GPU Enters Computing's Mainstream," IEEE Trans. Computers, Vol. 36, pp. 106-108, 2003.

[3] J.D. Andrews and S.J. Dunnett, "Event-tree analysis using binary decision diagrams," IEEE Trans. on Reliability, Vol. 49., pp. 230-238, 2002

[4] B. Zhang, W. –S. Wang and M. Orsharnsky, "FASER : Fast Analysis of Soft Error Susceptibility for Cell-Based Designs," Proc. of 7th Int'l Symposium on Quality Electronic Design, pp. 755 – 760, 2006.

[5] N. M. Zivanov and D. Marculescu, "MARS-C: modeling and reduction of soft errors in combinational circuits," Design Automation Conference, pp. 767-772, 2006.

[6] R. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Computers, Vol. 35., pp. 677-691, 1986.

[7] L. Dagum and R. Menson, "OpenMP: An Industry Standard API for Shared Memory Programming," IEEE Computational Science and Engineering, Vol. 5., No. 1, pp. 46-55, 1998.

Scalability of Parallel Applications: An approach to predict the computational behavior

Javier Panadero[1], Alvaro Wong[1], Dolores Rexachs[1] and Emilio Luque[1]
[1]Department of Computer Architecture and Operating System (CAOS),
University Autonoma of Barcelona, Spain
javier.panadero@caos.uab.es, alvaro.wong@caos.uab.es, dolores.rexachs@uab.es, emilio.luque@uab.es

Abstract— *When a message-passing application is executed many times over a long period of time, using an elevated number of resources, it is critical to predict its behavior before executing it. We propose a methodology to predict the strong scalability behavior for message-passing applications in specific systems. It is focused on characterizing and analyzing the communication and computational application patterns, from a set of executions in small scale, to project their behavior when the number of processes increases. The methodology strives to use a reduced number of resources. This paper presents the general methodology, focusing on validating the computational time model, which is a regression based approach. This model allows us to predict the computation time with high accuracy for a large number of processes. We executed from 16 to 256 processes and we predicted the computation time up until 4,096 processes. For the applications tested, we obtained an error of less than 9%.*

Keywords: Performance Prediction, Scalability, MPI Applications, Computation time prediction

1. Introduction

In recent years, High Performance Computing clusters have increased the number of cores significantly [1]. As a consequence, the users of these systems want to get the maximum benefit from this large number of cores, scaling their applications [2] .

Due to the complex interaction between message-passing applications and the HPC system, many applications may suffer performance inefficiencies, when they scale to a large number of processes. In order to achieve an efficient use of the system, it is critical to know the application behavior in the system before executing it, using an elevated number of resources.

We propose a methodology to analyze and predict the strong scalability [2] behavior for message-passing applications on a given system, by running representative phases of the application, signatures, in small scale. Moreover, the methodology could also be useful for scheduling and code optimization.

Message-passing applications are composed of a set of phases that are repeated throughout the application [3]. These phases were written in the application using specific communication and computational patterns, which follow behavior rules. To obtain these phases automatically, we use the PAS2P tool [4]. PAS2P allows us to generate the PAS2P signature, which contains only the relevant application phases and their repetition rates (weights).

The methodology focuses on characterizing and analyzing the communication and computational patterns of each phase in a transparent way, from a set of signature executions in small scale. By executing this set of signatures, we can obtain quick information about the phases's behavior, as the application scales, to model the general behavior rules of each phase. These rules specified the phase behavior and they allow us to predict their behavior as the number of processes increases. From these rules, the logical application trace is generated for a specific number of processes. This Logical trace has to be complemented with the communication and computational time, to predict application performance.

To predict the computational time, we use a regression-based model by phase, which uses as input data the computation time of each phase of the initial signatures. In many cases, the regression models are limited by the scope of prediction, obtaining a high prediction error when a distant point of the points used to generate the model is predicted.

In order to improve the prediction quality of the model, allowing us to predict distant points with high accuracy, we carry out a change of workload domain, using a workload much less than the original, to emulate the computation time for the original workload with a large number of processes. In this way, we are able to measure a distant point without executing for that large number of processes, to fit the model.

Once the computation time has been predicted for all the application phases, the physical trace is generated, which will be used to predict the communication time and obtain the performance prediction of the application.

In order to validate the method to predict the computational time, we executed from 16 to 256 processes and we predict the computational time 4,096 processes. For all the applications tested, the prediction error is less than 9%.

There are similar works which are related to predicting the computation time based on regression models, from executions for a small number of processes. Barnes et al [5] [6] propose studying the scalability using linear and

logarithmic regression functions, isolating computation and communication to predict the application performance. They use black models, where the internal application behavior is unknown. Calotoiu et al [7] present a tool to find scalability bugs. This tool automatically generates asymptotic scaling models for each part (kernel) of the application. The model is based on regression models, to fit the performance data from a set of small-scale performance experiments. Another similar work, presented by L. Carrington et at [8], offers a methodology for extrapolating the computational time of large scale applications by capturing the details of computational behavior at a series of smaller core counts. Unlike these works, our method proposes measuring a distant point using a reduced set of resources, in order to fit the computation regression model, improving the quality of prediction for far points.

This paper is organized as follows: Section II presents the proposed methodology to predict the scalability behavior, Section III presents the computation time prediction model, Section IV presents the experimental validation and finally Section V, the conclusions and future work.

2. Proposed methodology

The main goal of the methodology is to model the parallel application to analyze and predict the strong scalability behavior on a given system, by executing a limited set of application signatures in small scale, as is shown in Fig. 1.

Parallel applications are typically composed of patterns of computation and communication that are repeated throughout the application. These patterns are grouped in phases, which compose the application signature. If we execute the signature for different number of processes, we can observe that the number of phases remains constant, but their patterns change their behavior following behavior rules.

Analyzing the behavior of the phases, we know that the communications, the communication volume, the number of instructions and the computational and communication time can change, modifying their behavior, but the work to be carried out will still be the same, distributed among more processes, because we are working with strong scalability. In order to model the general behavior rules of communication, computation and weight of each phase, to project their behaviors as the number of processes increases, the phases of the signature for a different number of processes will be related by functional similarity.

Once these general rules have been modeled, we can generate the logical trace for any number of processes. This trace is composed of the communication events, the number of instructions and the weight of each phase. The trace is generated per process instead of a global trace with the objective being to model each process indepently.

To predict the application performance, the logical trace has to be complemented with the communication and computational time. To predict the computation time, we use

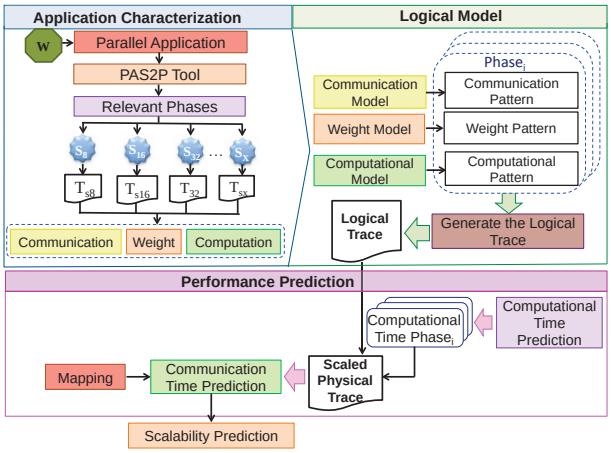


Fig. 1: Proposed Methodology

a regression-based model by phase, which uses the computation time of each phase of the initial set of signatures as input data. In order to improve the prediction quality of the regression computation model, allowing us to predict points for a large number of processes with a high accuracy, we use a method based on a change of workload. This method allows us to measure the phase computation time for an elevated number of processes (distant point), executing the signature for a reduced number processes and a small workload. In this way, we introduce a new distant point in the model, which allows us to fit the initial computation regression model, improving its quality of prediction.

Once the computation times have been provided to the logical trace, the physical trace is generated, which will be executed by pieces in a small number of cores, in a iterative way, until all the process has been measured, to predict the application performance.

In the next subsections, the steps of the methodology are presented.

2.1 Application Characterization

This step consists of characterizing the application behavior (communication and computation) to obtain information to build its logical trace. In order to do that, we carry out a set of signature executions for a small and different number of processes, which will be analyzed to extract information from each phase. The application signature extracts information of the application phases, which will be saved in a trace file per process. Fig. 2 shows an example of trace file obtained with the signature. The trace provides information

#Process	Phase	Type of primitive	Source	Dest.	Comm. Volume	#Inst.	Comm. Time (ns)	Comp. Time (ns)
4	1	MPI_Irecv	0	1	4000	756	4000	326
4	1	MPI_Send	0	1	4000	456	2345	134
4	1	MPI_Wait	0	1	4000	456746733	83593535	7654
4	1	MPI_Irecv	0	2	2000	975	7533	454
4	1	MPI_Send	0	2	2000	875	5366	332
4	1	MPI_Wait	0	2	2000	357876543	45326854	3466

Fig. 2: Trace file of the phase 1 for the process 4.

about the phase id, the type of MPI primitive, the source and destination of the communication, the communication volume in bytes, the communication time in nanoseconds and the computational time in nanoseconds.

It is noteworthy that the signature execution time is about 1% of the application execution time, covering approximately 95% of the whole application.

2.2 Logical Model

Once the phases have been characterized, the communication and computation patterns and the phase weight have to be analyzed and modeled to generate the general behavior rules of each phase. These rules will be used to generate the logical application trace for a greater number of processes.

2.2.1 Communication pattern modeling

The communication pattern comprises the general behavior equations and the data volume equations for each communication of each phase. The general behavior equations calculate the message destination from the source, while the data volume equations calculate the size of the message. To model the behavior of each phase, all the phases of the signature for a different number of processes will be related by functional similarity. To relate the phases, we use a method which is based on how the sequence of phases occurs, since it does not depend on the number of processes, only in the way in which the application was developed. Once the phases have been related, the predicted data volume equation of each communication will be obtained by mathematical regression models, while for obtaining the general communication rules (Source-Destination), an algorithm has been proposed. This algorithm is based on the fact that the application is well-developed, and it executes a deterministic communication pattern for all the processes, without non-predictive conditional sentences as the number of processes increases. Fig. 3 shows an overview of the procedure. This algorithm is based on obtaining the communication equations (eq.$_{processes}$) for each phase (local equations), which represent its communication pattern. From these equations, the general equation is modeled by each phase, which allows us to predict the evolution of the communication pattern of the phase for a greater number of processes.

2.2.2 Weight modeling

In order to model the weight behavior, regression models are used. Due to the deterministic way of the weight behavior as the application scales, there is a linear dependence between the number of processes and the weight of the phase. For this reason, linear regressions are initially more appropriate to fit the weight, because it allows us to obtain a prediction equation such as $y = a + bx_0$, using as an independent variable the number of processes to execute the application, which represents exactly the phase weight behavior, obtaining a $R - square = 1$.

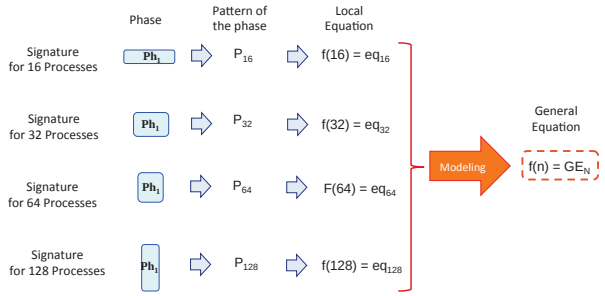
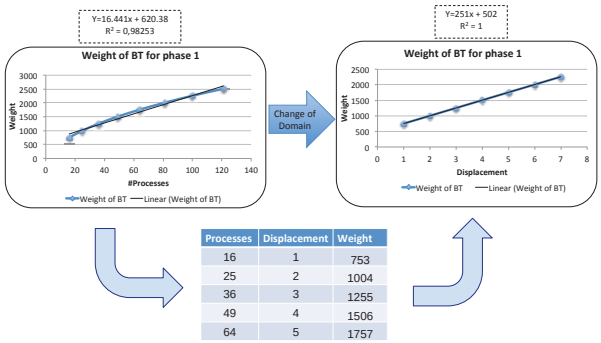


Fig. 3: Modeling the general equation of the communication pattern from a set of application signatures

Scientific applications cannot be executed for any number of processes, but they also follow execution rules. Depending on the number of processes required to execute the application, it can be possible that the linear regression does not fit properly, obtaining a correlation index R-square distant to 1. In this case, another kind of regression could be more appropriate. This happens by the distance between the input samples (Number of processes to execute the application) used to fit the regression. In fig. 4 we show an example, where through the limitations of the application (the users can only execute the application using a square number of processes), we use this to generate the model for the input points: 16, 25, 36, 49 and 64 processes.

As we can see in the figure, the distance of the input points used to model the regression is non-uniform, so, if we fit the points by a linear regression, we obtain a $R - square = 0.98253$. Through the theory of statistical regression models, we know that if we use an equation with this correlation index, the prediction error will be considerable and it will be higher as we move away from the executed points. In order to use a linear regression with an $R - square = 1$, we make a linealization process based on a change of domain, where the objective is to obtain an uniform distance among all the points. As we can see in the figure, we change the number of processes by a sequential index (displacement), making a distance of 1 for all the points. In this way, we obtain a $R - square = 1$ using a linear regression.



Processes	Displacement	Weight
16	1	753
25	2	1004
36	3	1255
49	4	1506
64	5	1757

Fig. 4: Modeling the weight of the phase

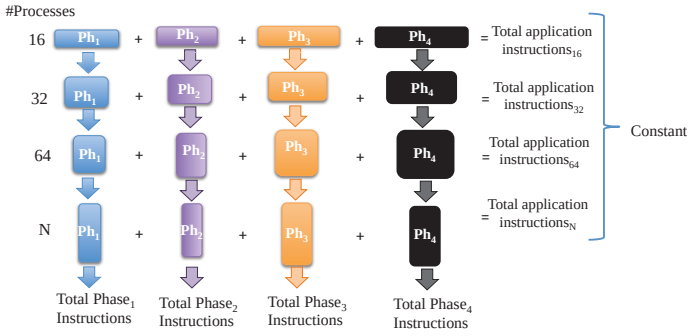


Fig. 5: Behavior of the strong scalability by phase.

2.2.3 Computational Pattern Modeling

In strong scalability, the application workload remains constant as the application scales. The workload is distributed among all the processes, and the instructions executed by each process decrease, as the number of processes increases, being the total number of instructions practically constant. We can extrapolate this concept to the application phases, maintaining its total number of instructions constant, as the application scales, as is shown in fig. 5.

To predict the number of instructions of each process by phase, we model as the instructions are distributed in the phase when the number of processes increases. To model these equations, the processes with a similar behavior in computation, that is a similar number of instructions (95% similarity), are grouped in Instructions Groups (IG_i). The total number of instructions of each Instruction Group remains constant. Then, each Instruction Group is modeled as the instructions are distributed as the number of processes increases. The sum of instructions of each group multiplied by the weight of the phase is the total number of instructions of the phase, as is shown in Eq. 1, where x is the total number of groups.

Fig. 6 shows an example of a phase with 4 processes with a different number of instructions. Processes 0 and 1 have a similar number of instructions, and processes 2 and 3 another. Scaling the application for 8 processes, processes 0 and 1 distribute their instructions between processes 0 to 3, while processes 2 and 3 distribute their instructions between processes 4 to 7, following their computation rules. Then,

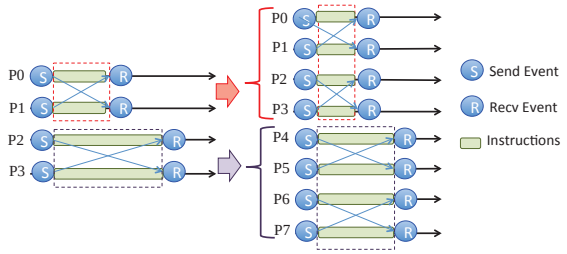


Fig. 6: Distribution of instructions as the number of processes increases.

for this example, we have 2 different groups, IG1 and IG2, where each group distributes its number of instructions in a specific way, following a behavior rule of distribution. As we show in the Eq. 2, which calculates the total number of instructions of the set of processes of a group *(n)*, the total instructions of the group is the sum of the instructions of each process *(Pi)* involved in the group, multiplied by the weight of the phase.

If we focus on Eq. 2, the aim is to predict the term "P_i" for a greater number of processes. We know that the term $TotalIG$ is constant, the weight of the phase is predicted by linear regression methods, and the number of processes between the instructions distributed in the group have been modeled. Then, we can predict the instructions of each process, isolating the term "Pi", as is shown in Eq 3.

$$TotalPhaseInstructions = \sum_{i=1}^{x} TotalIG_i * weight \quad (1)$$

$$TotalIG = \sum_{i=0}^{n} (P_i) * weight \quad (2)$$

$$P_i = \frac{\frac{TotalIG}{n}}{weight} \quad (3)$$

2.3 Performance prediction

The logical trace has to be complemented with the computational time, in order to generate the physical trace. To predict the computation time, we use a regression-based model by phase. As this is the main point of this work, this procedure will be explained in detail in the next section.

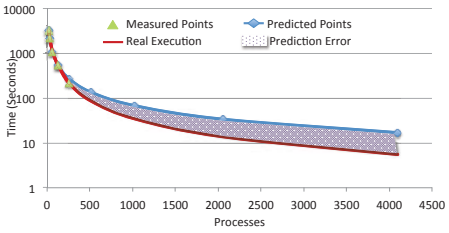
Once the physical trace has been generated, the communication time is predicted. To predict the communication time, the physical trace will be executed by range of processes in a reduced number of cores, in an iterative way, until all the processes have been executed. Once the communication time has been predicted, we will have the predicted execution times of each phase and their weight. Then, we apply eq. 4 to obtain the application performance, where PET is the Predicted application Execution Time, *m* is the number of phases, *TEPhasei* is the Phase i Execution Time and *Wi* is the weight of the phase i.

$$PET = \sum_{i=1}^{m} (TEPhasei)(Wi) \quad (4)$$

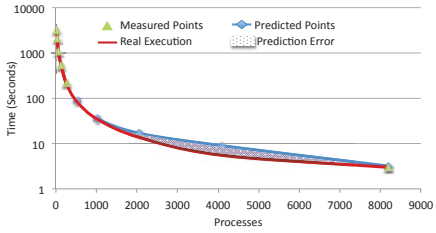
3. Computational time prediction

To predict the computational time of each phase, we use a regression-based approach named Computational Regression Model (CRM), which uses as input data the computational time of each phase for the initial set of signatures.

Despite predicting the computational time by phase instead of the whole application, the prediction error improves



(a) Computation Regression Model (CRM)



(b) Computation Regression Model (CRM) with distant point

Fig. 7: Regression models used to predict the computation time

considerably, because each phase has a different computation behavior which has to be approximated by a specific regression function, we know that the regression models are limited by the scope of the prediction. If we predicted a distant point from the real points used to generate the model, we would obtain an elevated prediction error as we move away from the measured points, as we can see in fig. 7a.

In order to avoid this problem, and therefore to improve the quality of prediction for a large number of processes (distant points) of the model, we propose a method which consists of measuring a far point, without using a large number of application processes and resources, to fit the initial Computational Regression Model (CRM) with this new point, as we can see in fig. 7b. Using this method, we manage to improve the accuracy of the model, predicting far points with a high level of accuracy.

The proposed method to measure this new point is based on doing a change of workload domain, using a workload much less than the original, to emulate the application computation time of each process for the original workload with a large number of processes.

A phase is a reduced segment of code, which executes a specific function. We can select a new workload for the phase, smaller than the original, which will be executed over a small number of processes. The objective is to achieve a similar number of instructions and cache misses by each process, rather than the original workload executed over a large number of processes, to emulate the computational time by process of the phase. In fig. 8 we show an example. We have an application, which is executed for 64 processes with a workload *W*. This workload is distributed between the application processes in a uniform way, with each process receiving a work *w'* . If we executed the same application for 4 processes with a new workload *M*, which is *w'* x 4, the processes are carrying out the same work (same instruction number and cache misses) as when executing the application for 64 processes with a workload *W* .

In fig. 9 we show a flowchart of the method used to predict the computation time of a phase:

1) From the logical trace obtained in the modeling step, we generate a new table, named Instruction Prediction Table (IPT), which contains a computational global vision of each phase. IPT contains the information about the number of instructions by process, the number of processes, the weight of the phase and its displacement, and finally the total number of instructions, as the application scales. Moreover, contains two different parts, a measured part and a predicted part. The measured part is generated from the information obtained during the execution of the initial set of signatures, while the predicted part is generated from the equations of computation generated in the logical trace. The measured part allows us to validate the accuracy of the model.

In order to obtain more than one Instruction Group in the computation pattern modeling, we generate as many Instruction Prediction Tables as Instruction Groups. The total number of instructions of each Instruction Prediction Table will be the total number of instructions of the phase.

2) At the same time, to generate the IPT table, we use the computation time of each phase of the initial set of signatures, in order to generate the initial Computation Regression Model (CRM), which will be used to predict the computation time of each phase.

3) From the IPT, we check if the total number of instructions is practically constant, as the number of processes increases. If this assumption is not met, the method is not applicable and we cannot obtain a distant point to improve the initial CRM model. In this case, we use the regression equation obtained in the CRM model, generated in the last step, to predict the computation time. Otherwise, we follow on to the next step to obtain a distant point.

4) In order to measure this new point, we select a small workload and we execute the signature with this workload for a small number of processes. We start executing the

Fig. 8: Emulating the computation time of a process changing the application workload

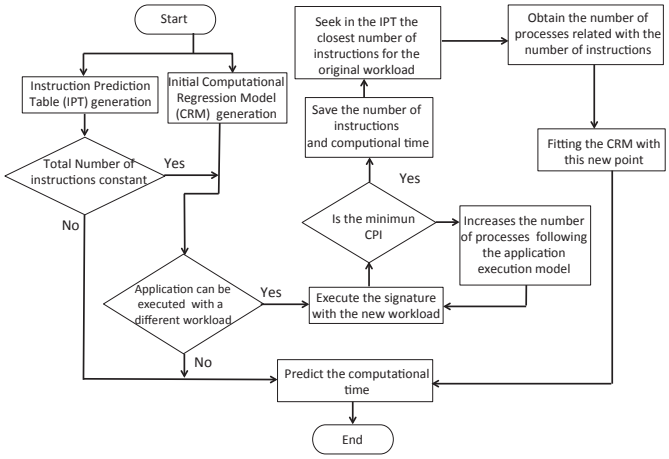


Fig. 9: Flowchart of the method used to predict the computation time

signature for 16 processes, and we increase the number of processes following the application execution model, until we find the minimum CPI of the phase. We select the minimum CPI because of two main goals, the first being that by using the minimum CPI we are sure that the workload is small enough to obtain a sufficiently distant point to fit the regression. The second reason is to avoid the effect of cache misses, since if we predict the computation time between the points of signatures used to generate the CRM model and the distant point, the model considers the cache effects. In addition, if we predict the computation time from the distant point, the cache effects practically do not have any influence in the model. Once we have executed the signature, we save this number of instructions and the computational time. We know that in some cases, because of limitations of the parallel application, it is not feasible to generate a different workload from the original. In these cases, it is not possible to obtain the distant point.

5) Then, we relate the number of instructions obtained in the previous step, with the number of processes for the original workload. In order to obtain this number of processes, we used the IPT generated in step 1. In the table, we seek the number of instructions closest to the number of instructions obtained by the small workload. Then, we will obtain the number of processes for this number of instructions.

6) Once we have the number of processes for the original workload, and the computational time, measured in step 4, we incorporate this new point to the the CRM model, to generate a more accurate new regression function.

7) We use this new regression function to predict the computation time.

4. Experimental Validation

In this section, the method to predict the computation time has been validated. We used different benchmarks such as: BT, CG, SP and LU from the NPB NAS [10] suite, using as input class D. Moreover, we used two applications: QCM [11] and N-Body. We predicted the computation time of each phase for BT and SP for 1024 processes, CG, LU and N-body for 4096 processes and QDIM for 2048 processes.

Table 1: Instruction prediction table for phase 1 of BT

Number of Instructions	Number of Processes	Weight	Weight Displacement	Total inst. Number
Measured Values (Initial Set of Signatures)				
1539358893	16	1255	3	30910326571440
820993294	25	1506	4	30910397519100
488835328	36	1757	5	30919812166656
314154733	49	2008	6	30910312489336
213799791	64	2259	7	30910318583616
152035493	81	2510	8	30910336081830
111953973	100	2761	9	30910491945300
Predicted Values (Instruction Prediction Table)				
1539358780	16	1255	3	30910324302400
.....	.....	.....	.....	.....
152035395	81	2510	8	30910316157450
111953349	100	2761	9	30910319658900
84813133	121	3012	10	30910315829644
65784545	144	3263	11	30910315829644
.....	.....	.....	.....	.....
1877266	1600	10291	39	30910315829644
1744266	1681	10542	40	30910315829644
1623539	1764	10793	41	30910315829644
1513701	1849	11044	42	30910315829644

For the BT, we predicted the computational time of each phase for 1,024 processes. We executed the signatures for 16, 36, 64, 81 and 100 processes, using the workload D, to generate the initial Computation Regression Model (CRM). We obtained 6 phases for this application.

For the case of phase 1, first of all, we modeled the computation pattern and the weight pattern in the logical model step. Through the modeling of the computation pattern, we obtain that all the processes have the same number of instructions, so we have only one Instruction Group, therefore, one IPT. Regarding the weight modeling, we used the linear regression equation $y = 251 * x + 502$, where "y" is the predicted weight and "x" is the displacement, as we increase the number of processes. This regression equation has a $R - square = 1$. From this information, we generate the IPT table, which is shown in Table 1. As we can check on the top of the table, the total number of instructions for the phase is practically constant, as the number of processes increases. For this reason, we look for a far point in order to be provided to the CRM.

To obtain a far point, we executed the signature of the BT using workload B (much less than workload D) for a reduced number of processes, until we found the minimum CPI. In table 2, we show the information of the different signature executions for phase 1 for this workload. We

Table 2: Signature executions for BT using the CLASS B

Number of Processes	Number of. Instructions	Number of Cycles	LLC Misses	CPI	Computation time (nsec.)
16	34356426	27621512	20241	0.803	17263445
25	17836224	14324977	10275	0.803	8953111
36	9915863	7861580	5479	0.792	4913488
64	4027761	3104163	1467	0.770	1940102
81	2392021	1721033	865	0.719	1075646
100	**1759558**	**1265527**	**695**	**0,719**	**790954**

show the number of processes for which the signature was executed, the number of instructions by process, the cycles, the misses of Last Level of Cache (L2), the CPI and the computation time in nanoseconds. As we can see in the table, from 81 processes we obtain the minimum CPI (0.719). We know what the minimum CPI is because for the next execution (100 processes), we obtain the same CPI and the number of misses is insignificant. Then, we select the number of instructions for the last execution of 100 processes (1759558) and its computation time (790954 ns).

The next step is to relate the instructions obtained for workload B with the number of processes of workload D. In order to do that, we seek the number of instructions obtained (1759558) in the Instruction Prediction Table for this phase.

As we can see in table 1, the closest number of instructions is 1744266, which has a difference of 0.87% with the number of instructions of workload B (1759558). This number of instructions corresponds to the execution of class D with 1681 processes. Thus, we select this number of processes to improve our model.

We introduce this new point in our model (number of processes and execution time), obtaining the regression equation $y = 9 * 10^9 * x^{-1.547}$, where the variable y is the computation time and the variable x is the number of processes to be predicted. We used this equation to predict the computation time for the phase for 1024 processes. As we can see in table 3, we obtain a prediction error of 2.11%.

In the same table, we show both the prediction error for the other phases of BT and the other applications tested, which were predicted to carry on with the same procedure. For the CG, LU, N-body and QDIM, we execute the signatures from 16 to 256 processes, while for SP, we execute the same number of signatures as for BT. All the phases of the application fulfilled the condition that the total number of instructions is constant. Therefore, the method to find the distant point was applied. For all the phases of the applications, the prediction error is below 9%.

Table 3: Summary of error prediction for the application phases

Summary of phases of BT (prediction for 1024 processes)				
Phase Num.	Real time(ns)	Predicted Time(ns)	Prediction Error (%)	Regression Equation
Phase 1	1941784	1982934	2.11%	$y = 9 * 10^9 * x^{-1.547}$
Phase 2	35960361	39114740	8.77%	$y = 1 * 10^{11} * x^{-1.132}$
Phase 3	165862020	160857620	3.01%	$y = 2 * 10^{11} * x^{-1.028}$
Phase 4	451140	480522	6.51%	$y = 3 * 10^{10} * x^{-1,593}$
Phase 5	2214673	2096107	5.35%	$y = 8 * 10^{10} * x^{-1,522}$
Phase 6	36062311	39114740	8.46 %	$y = 1 * 10^{11} * x^{-1.132}$
Summary of phases of CG (prediction for 4096 processes)				
Phase 1	2698523	2796674	3.63%	$y = 2 * 10^{10} * x^{-1.067}$
Phase 2	137928	149985	8.74%	$y = 3 * 10^7 * x^{-0.637}$
Phase 3	344297	375251	8.99%	$y = 2 * 10^7 * x^{-0.478}$
Phase 4	601238	562501	6.44%	$y = 8 * 10^7 * x^{-0.596}$
Summary of phases of LU (prediction for 4096 processes)				
Phase 1	51549	49647	3.83%	$y = 2 * 10^8 * x^{-0.998}$
Phase 2	34645	32573	6.36%	$y = 4 * 10^8 * x^{-1.132}$
Summary of phases of SP (prediction for 1024 processes)				
Phase 1	165110327	163103109	1.21%	$y = 4 * 10^{11} * x^{-0.926}$
Phase 2	561715	598590	6.56%	$y = 4 * 10^{10} * x^{-1.635}$
Phase 3	240359	260203	8.25 %	$y = 2 * 10^{10} * x^{-1.623}$
Summary of phases of N-BODY (prediction for 4096 processes)				
Phase 1	449150	430921	4.23%	$y = 2 * 10^{10} * x^{-0.975}$
Summary of phases of QDIM (prediction for 2048 processes)				
Phase 1	16428976	17724522	7.88%	$y = 3 * 10^{13} * x^{-2.039}$
Phase 2	26478907	25170124	4.94%	$y = 5 * 10^{10} * x^{-0.996}$

5. Conclusions and future work

In this paper, we have presented a methodology that allows us to analyze and predict strong scalability for message-passing applications on a given system, by executing a limited set of application signatures in small scale. The methodology has been explained focusing on validating the method to predict the computational time of each application phase. The proposed method allows us to predict the computation time for a large number of processes with a high accuracy using a reduced number of processes. For all the applications tested, the prediction error is less than 9%.

As future work, we are working on extending the computation model to measure far points of phases which do not have a similar number of instructions, as the number of processes increases.

Acknowledgment

This research has been supported by the MINECO (MICINN) Spain under contract TIN2011-24384

References

[1] N. Attig, P. Gibbon, and T. Lippert, "Trends in supercomputing: The european path to exascale," *Computer Physics Communications*, vol. 182, no. 9, pp. 2041 – 2046, 2011.

[2] R. Nishtala, P. Hargrove, D. Bonachea, and K. Yelick, "Scaling communication-intensive applications on bluegene/p using one-sided communication and overlap," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–12.

[3] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature for performance analysis and prediction," in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014 (Acepted).

[4] J. Panadero, A. Wong, D. Rexachs, and E. Luque, "A tool for selecting the right target machine for parallel scientific applications," in *ICCS*, 2013, pp. 1824–1833.

[5] B. J. Barnes, J. Garren, D. K. Lowenthal, J. Reeves, B. R. de Supinski, M. Schulz, and B. Rountree, "Using focused regression for accurate time-constrained scaling of scientific applications," in *IPDPS*, 2010, pp. 1–12.

[6] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ser. ICS '08, 2008, pp. 368–377.

[7] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13, 2013, pp. 45:1–45:12.

[8] L. Carrington, M. Laurenzano, and A. Tiwari, "Characterizing large-scale hpc applications through trace extrapolation," *Parallel Processing Letters*, vol. 23, no. 4, 2013.

[9] J. Dongarra, A. D. Malony, S. Moore, P. Mucci, and S. Shende, "Performance instrumentation and measurement for terascale systems," in *European Center for Parallelism of Barcelona*, 2003, pp. 53–62.

[10] D. Bailey, E. Barszcz, J. Barton, and D. Browning, "The NAS Parallel Benchmarks," *International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 66–73, Jan 1991.

[11] S. Hioki, "QCDMPI—pure QCD monte carlo simulation code with mpi," *Nuclear Physics B-Proceedings Supplements*, vol. 63, pp. 1000–1002, Apr. 1998.

SESSION

CLOUD COMPUTING AND APPLICATIONS

Chair(s)

TBA

DISCO: Unified Provisioning of Distributed Computing Platforms in the Cloud

P. Harsh[1], **and T. M. Bohnert**[1]
[1]InIT, Zurich University of Applied Sciences (ZHAW), Winterthur, Zurich, Switzerland

Abstract— *Big data is ubiquitous. The modus operandi recently has been to collect all possible data and then process later. Advances in distributed systems, fault tolerance and resiliency together with computational shift to cloud computing has enabled cheap storage and fast processing of huge amounts of data. Cluster computing has been there for decades, clouds bring the benefits of on-demand, pay-as-you-go, multi-tenancy, rapid elasticity to cluster/grid computing. There is an explosion of high quality open source distributed processing platforms (Spark, Hadoop, Storm, etc.). Furthermore, there is a plethora of add-ons for these platforms to enable specialized data processing needs of the scientific and business community. A unified semi-automated provisioning mechanism is necessary to reduce the complexity in using such platforms. In this paper we present our architecture and rationale for such a provisioning system which would take into account the data processing requirements of the researcher, and provide a unified experience for on-demand creation of a specialized data processing platform over IaaS clouds. We further show how our solution incorporates all major tenets of the cloud.*

Keywords: cloud computing, provisioning, big data

1. Introduction

Big-data [1] is defined as the increase in the volume, variety, velocity of data to such an extent that it becomes increasingly difficult to process using common statistical methods, and analyze through traditional databases. The world is becoming increasingly connected, with efforts being made by large technology firms to bring basic connectivity to every nook and corner [2] [3] of this planet. With cyber-physical systems [4], Internet of Things (IoT), connected cars [5] and home automation - billions of devices will be connected to the Internet. There will be an explosion of data which will provide huge opportunities as well as challenges to the entities hoping to make use of them. These big-data sources will present challenges of collection, storage, privacy, security, etc. but despite these, they will prove invaluable for future smart city planning, better and targeted advertisements, timely health care regimen, etc. The potential of such a scenario is limitless.

Distributed systems and processes, in particular cloud computing, have provided a reasonable technology platform for storage and processing of big data. Scientific community has traditionally relied on specialized super computers, huge government funded data clusters and grids for processing massive data sets in the past. In addition to scientific datasets, modern day general consumers are constantly generating big data through their social networking activities and online behavior. Naturally, businesses are increasingly becoming interested in using the power of data for better marketing of their services. Clouds provide businesses with a cheap alternative (to scientific grids) for processing big-data.

Last few years have seen an explosion of mature open source toolkits and platforms that allow scientific and business community perform distributed computing efficiently. There are numerous plugins [6] available for any popular platform for supporting variety of use cases. Every platform comes up with it's own challenges, configuration optimizations, provisioning specificities, and additionally, there is the challenge of making the platform operate seamlessly in a cloud environment.

Our work focuses on reducing the challenges in orchestrating and provisioning a desired distributed computing solution in the cloud from an end user perspective. We will outline the architecture of our distributed computing provisioning framework that would allow the user to specify the nature of the processing task, capacity requirements, and other relevant parameters following which the provisioning system takes over, creates the cluster computing environment over cloud, and returns necessary endpoints to the researcher / scientist for data uploads, job submissions, and result collection. We will also show how the platform could hook into cloud billing solutions so that metered consumption can be facilitated.

The rest of the paper is organized as follows: candidate data processing platforms and tools would be analyzed in section 2, architecture of our framework - DISCO will be presented in section 3, and work-flow analysis of DISCO will be carried out in section 4. We will present our analysis of a few selected related projects in section 5, and then we will conclude with a summary and plans for the future course of our work.

2. Popular Platforms

Distributed computing world has used map-reduce programming paradigm since a few decades, but it was made popular in the mainstream by the Google MapReduce [7]

paper that first came out in 2004. Since then rapid progresses were made in the maturity of the paradigm after Yahoo developed Hadoop for optimizing its search engine processes, which in part were also inspired from the Google File System [8] work[1]. Hadoop has gained popularity ever since, with an impressive ecosystem of tools and enhancements around it. In recent times, the focus of data analysis is getting broadened, to not only include batch processing tasks with stored data objects, but also handle non map-reduce jobs, and streaming data processing for more time sensitive (real-time) analytics. Stream processing is gaining significance as businesses want more real time trends from daily or even hourly data. Yahoo! uses S4 [9] for analyzing users' query submissions and click-through rate, Twitter uses Storm for real time classification and organization of tweets [10]. In some incident management systems, streaming data processing forms the core of the offering. Although there are numerous plugins available that allow most of the popular data processing platforms to support all kinds of the computations (batch, stream, graph, query, etc.) tasks, we will look into these following candidate platforms for supporting (initially) within our distributed computing provisioning framework DISCO. DISCO framework will support the full Berkeley Data Analytics Stack (BDAS)[2] in the later releases.

2.1 Apache Hadoop

Apache Hadoop is a mature open source map-reduce platform with a vibrant ecosystem of tools around it to support a myriads of data processing needs. The Hadoop map-reduce ecosystem consists of projects such as Pig, Mahout (for machine learning), Hive, etc. The Hadoop distribution uses YARN [11] as cluster manager from version 2 onwards, which is a monolithic (single-stage) scheduler. Application and resource scheduling were built inside Hadoop core in previous versions. The datasets are hosted in a distributed file system (HDFS) [12] that ensures recoverability in the face of node failures. Due to separation of scheduler logic from version 2 onwards, Apache Hadoop has started supporting not only map-reduce tasks, but other tasks as well such as matrix, graph, machine learning computation, etc. in a more efficient manner. In Hadoop, when a client submits a task, the YARN resource manager schedules and manages the task.

2.2 Apache Spark

Apache Spark [13] is a distributed data processing platform from UC Berkeley's amplab[3] and is the key piece in the Berkeley Data Analytics Stack (BDAS) [14]. Spark runs over a distributed memory-centric storage system called Tachyon [15], but also supports HDFS, Amazon S3, and GlusterFS. Spark execution engine supports not only map-reduce batch tasks, but stream processing, graph computations, machine learning, query processing, etc. Being largely based upon in-memory processing, Spark achieves higher processing speedup by manifolds compared to Apache Hadoop, especially for repetitive tasks through intelligent use of in-memory caches. The datasets are distributed as read-only resilient distributed datasets (RDDs). Spark runs over Mesos [16] which provides 2 level scheduling which enables variety of applications to manage in application task scheduling in a fine-grained manner.

2.3 Apache Storm

Apache Storm[4] is an open source, real time data processing platform originally released by Twitter after acquiring BackType [17]. It allows processing of infinite streams of data using tuple-at-a-time computational model. It uses *Nimbus* as the master node, Apache Zookeeper[5] for cluster coordination and a number of *Supervisors* which are the worker nodes. The Storm job topology consists of *Spouts* and *Bolts* arranged in a directed acyclic graph (DAG). *Spouts* are simply sources of streaming data. And *bolts* process a small batch of tuples from the stream. Storm platform guarantees data processing through the network of *spouts* and *bolts*, and supports at-least-once delivery, and transactional topologies [18].

3. DISCO Architecture

The DISCO architecture is influenced by the requirements of properly deploying and managing the candidate technologies listed in the previous section based on the user requirements. The high-level architecture is shown in the Figure 1. It shows the user facing elements, the core platform components, and also the relevant external cloud components for completeness. The architecture provides for an integration point towards the cloud rating-charging-billing platform Cyclops [19] [20] [21]. It supports both web-based (DISCO UI) as well as command line interfaces (DISCO CLI) towards end users. These communicate with the DISCO provisioning platform via RESTful [22] HTTP calls. The REST API Server is the only publicly exposed service. Every REST call is properly authenticated and authorized, although the authentication and authorization services are not shown in the architecture for brevity[6].

[1] part of the hadoop history was based on Gigaom article https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/ [retrieved: 11.03.2015]

[2] BDAS: https://amplab.cs.berkeley.edu/software/ [accessed: 18-03-2015]

[3] amplab: https://amplab.cs.berkeley.edu/

[4] Apache Storm: https://storm.apache.org/

[5] Apache Zookeeper: https://zookeeper.apache.org/

[6] OAuth protocol can be used to secure the internal services that makes up the core of the DISCO platform. Again, for brevity such inter services interactions have been left out.

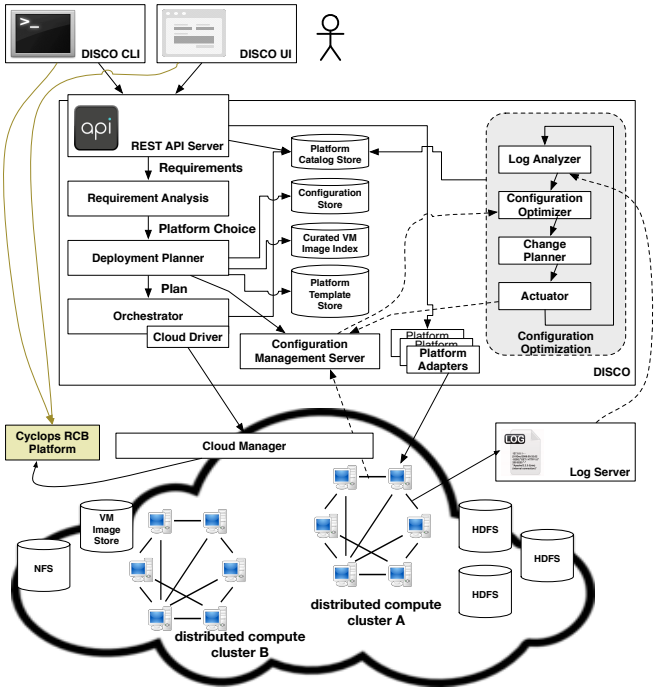


Fig. 1: DISCO Architecture

3.1 Core Components - Essential

The core of the platform is made up of these components:

Requirement Analysis Module
: As the name suggests, the task of this module is to analyze the end users platform requirements, type of computation, resource requirements, nature of the data source, etc. to determine which of the supported distributed computing technologies is best suited for their needs. The *Requirement Analysis* module provides the technology selection along with other parameters (capacity hints, placement requirements, etc.) to the *Deployment Planner* module.

Deployment Planner Module
: The deployment planner creates the proper cloud deployment strategy, it uses the technology suggestion, together with resource requirements to query the *Platform Template Store*, *Curated VM Image Index*, and *Configuration Store* to generate the deployment template for the *Orchestrator*. This module also initializes the base-line configuration of the soon to-be deployed *distributed compute cluster* in the *Configuration Management Server*.

Orchestrator
: The *Orchestrator* is responsible for deployment of the selected distributed computing platform over the cloud. The *orchestrator* uses the *Cloud Driver* to interface with the Infrastructure (IaaS) Cloud Manager (popular open-source choices being OpenStack[7], CloudStack[8], OpenNebula[9], etc.), and sends the correct sequence of commands to create the collection of virtual machines (VMs), in a predefined order, with correct software packages and configurations to make the data-processing service usable by the requesting person. Once the VMs are properly deployed, it returns the deployed virtual-compute-platform (VCP) entry points (IPs, other access details) to the calling environment. This way the access details are eventually passed on to the requesting user.

Configuration Management Server
: The *Configuration Management Server* keeps the current and past (checkpoints) configuration values for every VCP currently provisioned and maintained within the DISCO environment. The VMs are pre-configured to periodically contact this service and get any updates to the configuration parameters. The rationale behind this is runtime configuration optimization for long running services. The updated configuration can be applied in the VMs if there are no active jobs within the VCP. This is subject to acceptable configuration management policy agreed or specified by the user[10].

Configuration Store
: The baseline configuration for various distributed data processing technologies are stored in this database.

Curated VM Image Index
: The DISCO platform keeps track of the prepared VM images for each of the supported technologies in this index. This comes in handy during the deployment planning stage. The entries depend a lot on the underlying IaaS cloud platform.

Platform Template Store
: This is a collection of various deployment templates for each of the supported data processing platforms. The template is filled with missing details before being sent to the *Orchestrator* module depending on the user requirements.

Platform Catalog Store
: This store keeps track of the state of currently active virtual compute clusters managed through the DISCO framework. It stores the access details, relevant endpoints, health parameters, incidents / events, etc.

[7]OpenStack: http://www.openstack.org/
[8]Apache CloudStack™: http://cloudstack.apache.org/
[9]OpenNebula: http://opennebula.org/
[10]This functionality is very similar to features offered by popular configuration management tools such as Puppet, Chef, Ansible, etc.

These are the bare minimum required modules for implementing a usable *provisioning and management framework* for various distributed compute solutions.

3.2 Core Components - Optional

The core of the platforms consists of these (optional) value-addition components:

Platform Adapters / Drivers
: These driver programs enable the DISCO framework to periodically query the VCPs, which for the initial prototype would be Hadoop, Spark or Storm clusters. Using these drivers, the provisioning system could track cluster statistics, job status, etc. and could enable the REST API server to fetch on-demand status and statistics to be returned back to the user. This would enable us to design and implement a more functional CLI and web UI for end users.

Configuration Optimization
: *Configuration Optimization* service will periodically check log server entries from various VCPs, analyze them, and if errors or warnings are found, look into the applied configuration parameters, and past check-pointed configurations (if available). This module will look into knowledge-base (not shown in the picture) to locate safe and optimal configuration parameters (from previous runs). If not found, then the problem can be propagated to the operators for manual intervention. The idea behind this module borrows a lot from Incidence Management [23] [24] domain and aligns with the widely accepted MAPE-K [25] loop reference model. The *change planner* plans the correct order of the configuration updates to be applied among various VMs, and services in them, which comprises the concerned VCP. The actuator simply updates the *Configuration Management* service which in turn checkpoints the previous values, and prepares them for storage into the knowledge-base.

3.3 External Components

For sake of completeness, here we briefly describe the external components / services with which DISCO framework has either required or optional dependencies.

Log Server
: All the VMs are pre-configured to send logs to a remote log server in addition to storing them locally in the VM's scratch space. The remote logging can be enabled on request by the user if they wish to enable run-time optimizations through the *Configuration Optimization* module in the DISCO-core.

Cloud Manager
: Cloud manager is the external component that not only enables life-cycle management of VMs running in the cloud, but also creation of the IaaS cloud from physical servers in the data-center. In our prototypical implementation, this is *OpenStack*. Other popular cloud managers are Eucalyptus, CloudStack, OpenNebula, etc. DISCO interacts with various IaaS clouds through the appropriate *Cloud Driver* which enables the *Orchestrator* to communicate with the cloud managers through their APIs.

VM Image Store
: All customized VM images that allows DISCO to provision and manage the requested (supported) distributed data-processing platform are kept in the *VM Image Store*. Most *Cloud Management* softwares, including *OpenStack* come with a preferred *Image Store*. In our case, this will be *OpenStack Glance*[11].

NFS Service
: Sometimes it become necessary to enable end users to easily upload files for processing in the VCP. In some cases (esp. Apache Spark), the files in all the worker nodes in the (virtual) cluster must reside at the same path. This can be easily achieved if all VMs have a common remote network file system (NFS) [26] share mounted at the same absolute path in their local file-system.

Cyclops
: *Cyclops* [19] [20] is an open source framework that allows custom rating-charging-billing solution for cloud based services. It exposes a REST API for external (non IaaS) services to send in metered data. Using a combination or variety of metered data, various billing strategies can be implemented. DISCO will integrate with *Cyclops* and send in any non standard, framework specific metered data into it. This will enable measured, pay-as-you-go service utilization cloud principle.

4. Workflows & Platform Analysis

DISCO framework's unique features are -

- unified provisioning
- runtime configuration optimization
- virtual cluster lifecycle management & enabling measured service

The sequence diagrams and algorithms shown here will analyze how these features are operationalized within the architectural framework described earlier.

4.1 Unified Provisioning

Sequence diagram shown in Figure 2 presents a highly simplified work-flow involved in a *unified provisioning* of

[11] OpenStack Glance: http://glance.openstack.org/

any selected distributed computing platform in the target IaaS cloud. It presents the basic steps from user filling a request form / questionnaire in the Web-UI and from clicking *submit* until the selected compute cluster is provisioned over the cloud. Upon successful provisioning the access details for the cluster is returned back to the user. The various

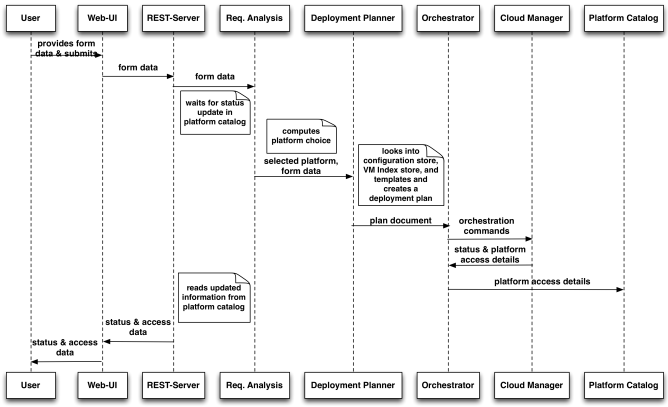


Fig. 2: Unified Provisioning Work-flow

other usual steps such as data preparation and on-boarding, authentication and authorization, etc. are not shown in the steps for simplicity.

4.2 Runtime Optimization

The DISCO platform periodically looks for log entries from all active virtual compute platforms (VCPs), and then attempts to identify suboptimal state from them. The *runtime optimization* component searches the knowledge-base for historical occurrences of similar incidences, and prepares the new configuration to help the VCP reach efficient state (if possible). If a discovered incident can not be resolved by the module independently, then an alert is raised to the operator, to bring in human in the loop. A new entry is then entered by the operator in the knowledge-base so that similar incidents can be automatically handled in the future. The pseudo-code for a possible platform *runtime optimization* approach is shown in Algorithm 1.

The sys-admin upon receipt of the alert would manually intervene, investigate the incident, and apply the fix manually to the virtual cluster. In this process he will update the resolution notes in the placeholder knowledge-base entry enabling similar incidences to be handled automatically.

4.3 Cluster Life-cycle Management & Cloud Principles Enablement

The DISCO framework allows users to provision a distributed compute cluster through a web interface or a CLI client-tool. This enables on-demand self-service provisioning of any supported platform. The cluster is made up of a number of virtual machines running over an IaaS cloud, therefore the usage measurement taken at the IaaS level

Data: Log entries from the log server
Result: Optimal VCP entity configuration to resolve any identified adverse incident / event
initialization;
filter log messages to type WARN or ERROR from log files;
foreach *entry in WARN or ERROR list* **do**
  identify the VCP id for this log;
  collect all WARN or ERROR level messages for this VCP instance;
  look in knowledge-base for the possible fix;
  if *fix located* **then**
    create a patch configuration based on knowledge base entries / notes;
    create the patch application plan based on the number of VMs in the VCP;
    write the patched configuration and application plan into configuration management server;
    update the knowledge-base;
  end
  else
    //no knowledge-base entry was located;
    create a ticket for the system admin;
    create a placeholder knowledge-base entry;
    send alert with ticket-id to administrator;
    //unresolved events are tracked inside the platform catalog;
    create a pending resolution entry for this event in the platform catalog store;
    //duplicate entries (if exist) are merged;
  end
end

Algorithm 1: Runtime Optimization Process

covers the major part of DISCO metering. Integration with *Cyclops* would allow DISCO to send in non-IaaS metered data into the charging and billing platform. *Cyclops* allows measured use of the DISCO framework. On demand elasticity is supported by virtue of the cloud use. Dynamic scaling of the compute cluster is generally supported in all major open-source distributed computing platforms.

The compute cluster life-cycle management - from cluster deployment, to configuration management, runtime optimizations, and cluster disposal when no longer needed, is handled in the DISCO framework. Most of the processes and work-flows are intuitive based on the architectural element described earlier, and are facilitated by the cluster states and access data is maintained in the *platform catalog store*. The DISCO framework uses easy primitives and simple work-flows, an example of which is the way it handles the unresolved incident / event (as reported by the *Configuration Optimization* module) in the *Platform Catalog Store*. This is shown in Algorithm 2.

```
Data: Platform State Data from Platform Catalog Store
Result: Update of Platform State in the Platform
        Catalog Store
if VCP event flag is set then
    if VCP event is in unresolved state then
        resend alert to the sysadmin;
        //this allows alerts to be sent periodically until
        the event is resolved;
    end
    else
        //sysadmin has manually intervened and fixed
        the errors in the VCP;
        //after which he has likely updated the
        knowledge-base with resolution notes for this
        class of events;
        //the manual update of the knowledge-base
        already marks the event state as resolved;
        reset the VCP event flag;
        if Configurations in Configuration Management
        Server is not updated then
            update the VCP configurations in the
            Configuration Management Server;
            //this would make sure that the older
            configurations are accidentally not applied
            during future configuration syncs;
            //ideally the new configurations for the VCP
            is immediately updated in the Configuration
            Management Server.
        end
    end
end
```

Algorithm 2: VCP Unresolved State Resolution Process

5. Related Projects and Service Offerings

Almost all major cloud providers support on-demand big-data analytics platform provisioning. There are a few open-source projects that look into similar service. A small subset of prominent alternatives are discussed next.

5.1 OpenStack Sahara

Sahara [27] is an integrated, community driven, open-source project in the OpenStack ecosystem where the principal goal is to provide Apache Hadoop as a service to users. Their road-map includes providing support for Apache Spark platform in the near future and an early prototype towards that goal was demonstrated by Eurocom [28]. The project goals are similar to DISCO architecture, but is intimately tied to OpenStack cloud platform and its various internal services. The project allows user to choose certain characteristics of their hadoop cluster - cluster size, heap size, etc. For data analytics, it supports a number of Hadoop plugins such as pig [29], hive [30] and allows upload of custom jar files. Data files are assumed to reside in Swift[12] object store. DISCO additionally plans to support continous cluster optimization wherever possible in addition to support multiple platforms over possibly many cloud environment.

5.2 Apache Ambari

The architecture of Apache Ambari [31] is similar to the one proposed for DISCO. Ambari allows provisioning, managing and monitoring of Hadoop clusters. It tracks the health for worker nodes, and sends out alerts to sysadmins if a node is unreachable. It also handles configuration management. Similar to the proposed DISCO features, Ambari provides a REST interface and provides both Web-UI for interacting with the core service, and various command line clients. Conceptually, it is similar to bare-metal provisioning systems like Foreman[13] and Juju[14] but designed specifically for the Hadoop provisioning. DISCO architecture is not tied to any specific data processing platforms, although it will support Hadoop, Spark and Storm to start with. The run-time configuration optimization and domain-specific encoding for platform selection algorithm is novelty in DISCO.

5.3 Pivotal Big Data Suite

Pivotal's Big Data Suite recently was made open-source and the development in this effort is governed within the ambit of *Open Data Platform* (ODP)[15]. The principal goal of ODP initiative is to prevent the fragmentation in the Apache Hadoop ecosystem. The Pivotal's *Open Data Platform* provides Hadoop based big data processing solutions, governed by the same goals targeted by DISCO. The DISCO framework tries to go a bit farther by aiming to encode the scientific and research communities' data platform selection logic into a decision making module to automate this process; this would enable faster convergence towards the optimal platform choice for DISCO's clients. Furthermore, continuous configuration optimization in the light of differentiated use cases is a unique feature in DISCO. Furthermore, the core goal of ODP and *Pivotal's Big Data Suite* is faster monetization of big-data solutions and supporting enterprise business logic through promotion of *data driven applications*. DISCO is geared towards scientific and research community, but in the process would facilitate *data driven applications* in enterprises also.

5.4 Azure HDInsight

Microsoft's popular cloud offering Azure recently started supporting map-reduce data analytics called HDInsight[16]. Their platform leverages Hortonworks Data Platform (HDP)

[12] OpenStack Swift: http://docs.openstack.org/developer/swift/

[13] http://theforeman.org/

[14] https://jujucharms.com/

[15] ODP: http://opendataplatform.org/

[16] HDInsight: http://azure.microsoft.com/en-us/solutions/big-data/ [accessed 16-03-2015]

Hadoop distribution and provide support for Apache Hadoop, HBase and Storm platforms. Additionally they support Ambari[17] for provisioning, monitoring, and management of Hadoop clusters, Hive, Mahout[18] for machine learning, Pig and other projects from the Hadoop ecosystem. HDFS [12] is the standard file system in HDInsight. It allows users to export results in Excel among other Microsoft supported platform formats.

5.5 Amazon EMR

Amazon supports data analytics through its Amazon Elastic MapReduce (EMR) service. It enables data stored in Amazon S3 and DynamoDB to be processed in EMR clusters and supports many of the popular Apache Hadoop ecosystem modules & plug-ins including Pig, Hive, Spark, among several others. Amazon EMR is well integrated with other Amazon EC2 solutions including virtual private cloud. It allows resizing of a running EMR cluster[19]. DISCO in comparison is an open source solution that aims to achieve most of the functionality, but using a modular, cloud platform agnostic approach. Our aim is not only to support map-reduce jobs but other statistical tasks using a semi automated platform selection process which is optimized for the task at hand.

Brief Analysis

DISCO aims to bring distributed computing in the cloud to scientific and business community through a highly intuitive, unified and semi-automated provisioning framework. The proposed framework will encode various scientific domain expertise in the implemented algorithms for platform selection, configuration management subsystems, and will have an element of learning built-in to continually update the optimal configuration knowledge-base based on past and present run experiences. This is in stark contrast to other open source initiatives including OpenStack Sahara.

6. Conclusion

In this paper we have presented and defended the architectural choice of our unified cloud provisioning framework for big-data and distributed computation in the cloud, called DISCO. The design is generic and independent of any specific cloud platform, the integration with various clouds is achieved through specific drivers leaving the general mechanism unchanged. In the initial prototypical implementation of the platform we have chosen Apache Hadoop, Spark and Storm open source solutions to support over OpenStack cloud, but this does not limit us from integrating and supporting other platforms in the future. One feature of our platform is the manifestation of all the key cloud principles - self-service, on-demand, elastic, measured pay-as-you-go, etc. This is similar to commercial services such as Amazon EMR, but DISCO framework will be open-source, in principle, and will provide support for wide variety of distributed computing paradigms and not just map-reduce.

We realize that many scientific groups have their own in-house developed, supported data analysis solutions, hence our focus will also be on enabling such teams to bring in their custom distributed computing application through our DISCO framework onto popular IaaS clouds. In [32], authors have proposed a cost-optimized configuration management and deployment strategy of a data analytics workload in the cloud. Their work attempts to search the optimal configuration that satisfies the Service Level Objectives (SLOs) of the workload while minimizing the cost of deployment over a public cloud. This could form the basis for SLO and SLA aware deployment and configuration management in DISCO.

[17] Ambari: http://ambari.apache.org/

[18] Apache Mahout: http://mahout.apache.org/

[19] AMR product Details: http://aws.amazon.com/elasticmapreduce/details/ [accessed: 16.03.2015]

Acknowledgment

The work is inspired partly by the MCN orchestration framework and is supported by the European Community Seventh Framework Programme (FP7/ 2001âĂŞ2013) under grant agreement no.318109.

References

[1] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of big data on cloud computing: Review and open research issues," *Information Systems*, vol. 47, no. 0, pp. 98 – 115, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437914001288

[2] "Loon for All - Project Loon - google," http://www.google.com/loon/, accessed: 2015-03-11.

[3] "internet.org by facebook," http://internet.org/, accessed: 2015-03-11.

[4] P. Bogdan and R. Marculescu, "Towards a science of cyber-physical systems design," in *Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, ser. ICCPS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 99–108. [Online]. Available: http://dx.doi.org/10.1109/ICCPS.2011.14

[5] "Open Automotive Alliance," http://www.openautoalliance.net/, accessed: 2015-03-12.

[6] A. Mostosi, "The Big-Data Ecosystem Table," http://bigdata.andreamostosi.name/, accessed: 2015-03-11.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43. [Online]. Available: http://doi.acm.org/10.1145/945445.945450

[9] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, Dec 2010, pp. 170–177.

[10] E. Chen, "Improving Twitter Search with Real-Time Human Computation)," http://blog.echen.me/2013/01/08/improving-twitter-search-with-real-time-human-computation/, accessed: 2015-03-19.

[11] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1–10.

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113

[14] M. Franklin, "The berkeley data analytics stack: Present and future," in *Big Data, 2013 IEEE International Conference on*, Oct 2013, pp. 2–3.

[15] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 6:1–6:15. [Online]. Available: http://doi.acm.org/10.1145/2670979.2670985

[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[17] Wikipedia, "Storm (event processor)," http://en.wikipedia.org/wiki/Storm_%28event_processor%29, accessed: 2015-03-18.

[18] Apache Storm Community, "Transactional Topologies," https://storm.apache.org/documentation/Transactional-topologies.html, accessed: 2015-03-19.

[19] "CYCLOPS - Rating, Charging, Billing solution for Cloud Providers," http://icclab.github.io/cyclops/, accessed: 2015-03-11.

[20] P. Harsh, K. Benz, I. Trajkovska, A. Edmonds, P. M. Comi, and T. M. Bohnert, "A highly available generic billing architecture for heterogenous mobile cloud services," in *Proceedings of the 2014 International Conference on Grid & Cloud Computing & Applications*, ser. GCA '14. CSREA, 2014, pp. 29–38. [Online]. Available: http://worldcomp-proceedings.com/proc/proc2014/gca.html

[21] S. Patanjali, B. Truninger, P. Harsh, and T. M. Bohnert, "CYCLOPS: a micro service based approach for dynamic rating, charging & billing for cloud," in *The 13th International Conference on Telecommunications (ConTEL 2015)*, Graz, Austria, July 2015.

[22] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[23] W. Guo and Y. Wang, "An incident management model for saas application in the it organization," in *Research Challenges in Computer Science, 2009. ICRCCS '09. International Conference on*, Dec 2009, pp. 137–140.

[24] J. Cusick and G. Ma, "Creating an itil inspired incident management approach: Roots, response, and results," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, April 2010, pp. 142–148.

[25] A. Keller, "Towards autonomic networking middleware," May 2005. [Online]. Available: http://www.research.ibm.com/people/a/akeller/Data/ngnm2005_slides.pdf

[26] B. Callaghan, B. Pawlowski, and P. Staubach, "NFS Version 3 Protocol Specification," June 1995, RFC 1813. [Online]. Available: http://tools.ietf.org/html/rfc1813

[27] OpenStack Sahara Community, "Sahara - OpenStack," https://wiki.openstack.org/wiki/Sahara, accessed: 2015-03-16.

[28] ——, "Sahara/SparkPlugin - OpenStack," https://wiki.openstack.org/wiki/Sahara/SparkPlugin, accessed: 2015-03-16.

[29] Apache Pig! Community, "Welcome to Apache Pig!" http://pig.apache.org/, accessed: 2015-03-16.

[30] Apache Hive Community, "Apache Hive TM," https://hive.apache.org/, accessed: 2015-03-16.

[31] Apache Ambari Community, "Ambari Design," https://issues.apache.org/jira/secure/attachment/12559939/Ambari_Architecture.pdf, accessed: 2015-03-18.

[32] R. Mian, P. Martin, and J. L. Vazquez-Poletti, "Provisioning data analytic workloads in a cloud," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1452 – 1458, 2013, including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for {P2P} Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12000209

Crowdsourcing the Cloud: Energy-aware Computational Offloading for Pervasive Community-Based Cloud Computing

Kassahun Adem[1], **Caspar Ryan**[1], **and Ermyas Abebe**[2]
[1]School of Computer Science and IT, RMIT University, Melbourne, Australia
[2]IBM Melbourne Research Lab, Melbourne, Australia

Abstract *- Adaptive offloading systems achieve context specific optimization on mobile and pervasive devices by offloading computational components to a resource copious remote server or cloud. However, with the recent advancement in computational capacity of mobile and pervasive devices, adaptive offloading could facilitate the formation of ad-hoc cloud-like environments using collections of mobile and pervasive devices, with reduced reliance on centralized infrastructure. Therefore, in this paper, we formulate a decision-making strategy for global adaptive offloading that distributes application components to community-based clouds formed from multiple collaborating peers. The goal was to extend the collaboration and application lifetime by optimizing the Time to Failure (TTF) of devices due to energy depletion, while meeting application specific performance constraints. Specifically, a max-min technique was used to maximise the minimum TTF in order to balance energy consumption across collaborating devices. The efficacy, performance and scalability of the formulated model were evaluated with the proposed algorithm producing an optimal solution to the specified model, using integer linear programming, in affordable time and energy for a range of application and collaboration sizes.*

Keywords: Adaptive Offloading; Collaborative Application Partitioning; Energy Conservation; Pervasive Community Cloud; Time to Failure; Peer-to-peer Distributed Computing.

1. Introduction

In this era of cloud computing, there is a commercial push to run everything in the cloud, but nevertheless there exists a need for large-scale, cloud-like functionality where cloud infrastructure does not exist, such as in emergency situations, developing countries or for reasons of privacy or energy conservation [1-3].

This is because despite obvious benefits in terms of economies of scale, cloud-computing poses challenges in terms of privacy, monopolistic control by large cloud vendors and issues of centralized failure. To some extent, decentralized data centres can mitigate risk, but significant technical failures such as that of the Amazon Elastic Computing Cloud (EC2) [4] and Amazon (S3) cloud [5] demonstrated the cascading effect on organizations dependent on Amazon's infrastructure. Other potential examples of cloud infrastructure failure include natural and man-made disasters such as: earthquakes and tsunamis; war; terrorist attack, and chemical or nuclear contamination. In such situations, a crowd sourced community cloud based on collaborating mobile and pervasive devices could serve as an important backup for providing cloud like functionality in the absence of fixed cloud infrastructure. Other clear opportunities for community clouds are developing countries or NGOs that lack financial power to run their own centralized infrastructure or pay for vendor based cloud services at international rates; or countries where infrastructure is under tight control or where groups desire informal, ad-hoc or private computing collaboration.

Furthermore, cloud computing presents challenges due to its ever-increasing energy usage caused by the exponential growth of data centres and cloud infrastructure [6, 7]. This presents economic challenges due to resource scarcity and carbon pricing as well as environmental impact [6]. In contrast, pervasive community clouds could facilitate greener computing alternatives where individual devices are powered by existing local renewable energy resources such as residential solar panels or wind turbines, or by harvesting the kinetic energy of the human body. In addition, the major share of energy consumption in cloud computing data centres comes from facility management and plant infrastructure [8, 9] such as cooling equipment, uninterruptible power supplies (UPS), electrical distribution equipment, security equipment, fire suppression, generators, and so on. This energy can be conserved by migrating computation to mobile/pervasive devices that are distributed among pre-determined or opportunistic communities. Moreover, most of these devices are fitted with ARM based processors that use significantly lower energy than Intel [10-12] based data centres servers, which can further reduce energy consumption.

With current and future pervasive computing environments providing increasingly large-scale collaboration [1, 3, 13, 14], the crowd-sourced or community-based cloud is becoming closer to reality [1]. Nevertheless, running collaborative applications using pervasive computing devices has many challenges, such as device heterogeneity, battery power limitations, and collaborators joining and leaving due to either mobility or personal choice. Furthermore, given the dynamic and variable nature of such challenges, applications running in such environments have to adapt their behaviour to align with their frequently varying context.

Previous research has considered adaptive computation offloading (also known as cyber foraging [15]), which involves the runtime distribution of computational components

to devices in order to achieve context specific optimizations. However, most of this work has focused primarily on offloading components from constrained clients to resource copious servers in a client-server fashion. In contrast, this paper is concerned with computational offloading to form crowdsourced or community cloud infrastructure using scalable peer-to-peer configurations of pervasive and mobile computing devices.

Consequently, we present a decision-making strategy for global adaptive offloading that extends adaptation lifetime by optimizing the Time to Failure of collaborating devices in pervasive environments. This is done by balancing individual device energy consumption across the collaboration within application specific performance constraints. The scalability and performance of the formulated model is evaluated using synthetic data and test cases, with the proposed algorithm producing a global optimal solution, using an integer linear programming model, in affordable time and energy for a range of application and collaboration sizes on a laptop and a smartphone, which serve as typical examples of contemporary mobile devices. The remainder of the paper is structured as follows: Section 2 discusses exiting adaptive offloading approaches as a basis for an energy-aware optimization model for peer-to-peer adaptive offloading which is proposed in Section 3. Section 4 evaluates the model under a range of environmental scenarios to explicitly quantify performance and energy usage, and finally, Section 5 provides a summary, conclusion and outline of future work.

2. Background

The concept of adaptive computational offloading or cyber forging is an ongoing field of research [15-19].

Generally, computational offloading involves at least two essential processes. The first is the metrics collection and management process, which is responsible for monitoring device and environmental context (resource availability, network connectivity, movement etc.) as well as the behaviour of application components in terms of performance and resource utilization. The second is the decision-making process, which is responsible for optimizing the placement of components on candidate devices according to one or more objectives. Typically, such objectives relate to device resource availability, component resource usage and the coupling patterns between components.

Adaptive computation offloading can be implemented at different levels of component granularity. For an object oriented system, such levels (listed from coarse to fine grained) include process [20], class [21-23], object [24, 25], and method level [26]. A service or component-oriented system would share similar levels of abstraction. Furthermore, a hybrid granularity approach has also been proposed [27]. Given this variation, the model presented in section 3 is a general solution that could be applied at any level of granularity.

Additionally, the decision making process for computational offloading can be global (centralized) or local (decentralized) based on the location in which such decisions are computed [28]. In global adaptation, a single device performs adaptation decisions by mapping application components to collaborative nodes for the entire application in a single pass. Other collaborating devices periodically communicate their context (usually as formalised metrics) in order to inform adaptation decision making. In contrast to global adaptation, local (or decentralized) adaptation performs decisions on individual nodes in terms of components residing in their local memory space.

Although the focus of this paper is global adaptation, in practice, global and local adaptation are complementary rather than mutually exclusive, and if combined appropriately, could give offloading systems greater flexibility and completeness. For instance, adaptive offloading could be run globally to bootstrap system startup, as well as at times of significant system-wide change. In contrast, local strategies could be used for smaller or on-going adaptation. We will explore and address local adaptation and hybrid offloading system architecture in future work.

To date, most adaptive offloading research has focused on client-server architecture [21, 23, 25, 26, 29-31] in which portable devices mitigate one or more constraints by offloading components to a remote server or dedicated surrogate.

Of the fewer studies that have been done on peer-to-peer offloading [27, 32, 33] the focus has been on small collaborations using heuristics and local adaptive offloading schemes where efficiency and scalability is less of a challenge. In addition, none of the peer-to-peer work explicitly considers energy as an offloading decision factor. This is significant because even as there is progress in battery technology alongside processing capacity, memory size and network capacity, increasing application demands continue to negate such advances [34, 35]. Furthermore, the failure of one or more devices significantly affects the rest of the collaboration by triggering further adaptation that takes time and consumes additional resources and thus energy. This issue of progressive device failure due to energy depletion has not been addressed in previous work.

Therefore, in this paper, we propose a novel optimization model that maximises the time to failure of collaborative devices in the collaboration by formulating an energy-aware adaptive offloading decision-making strategy that conserves the energy usage of portable devices in scalable pervasive environments, within application specific performance constraints. Another critical aspect of computation offloading when applied to resource constrained environments is the computational complexity of the decision making process. This is especially true when full optimization is used instead of heuristic approaches. Thus, we consider the efficiency of the proposed optimization model, presenting the performance and energy over head of this model with a range of application and collaboration size in the evaluation of part IV.

Of particular relevance to this work is the MAUI system by Eduardo C. et al. [26], a fine-grained (method-level) code offloading framework that aims to minimize a smartphone's energy consumption within required performance constraints.

MAUI uses a global optimization model (integer linear programming (ILP)) to decide whether a method executes locally or on a resource copious remote server. The MAUI solver is notable for producing an optimal offloading solution with modest runtime and energy overhead. However, MAUI is limited to client server architecture based on a single dedicated server and thus does not meet the community cloud based aims of this paper.

Consequently, this paper, motivated by the MAUI approach, presents an energy-aware linear integer-programming optimization model to maximize time to failure of collaborative devices as a step towards scalable ad-hoc community clouds on pervasive and mobile computing platforms. The model is elaborated in the following section.

3. Formulated Model

Before presenting the optimization model, some background of the proposed scheme follows.

3.1 Background

Consider a collaborative application comprising un-offloadable and offloadable components. The components of an application and their interactions can be represented as an undirected graph G=(V,E) [23]. The vertex set V, v∈V represents the components of the application (components are an abstraction of objects/methods/classes/services or hybrid variations as discussed in section 2). The edge set e∈E represents an invocation or data access between components.

Assume there are N collaborative devices or nodes that are willing to execute certain parts of the application by hosting and executing offloadable components. The optimization model concerns how to partition the application i.e. distribute offloadable application components to N collaborative devices, in a way that maximizes the lifetime of collaborative devices due to energy depletion whilst meeting the performance constraint of the application in terms of resources such as shared energy, memory and network capacity.

Prior to an offloading decision, application components can reside in a single device or be distributed across the collaboration. For example, in Figure 1.A, all application components reside in device n1. In Figure 1.B, application components are distributed across different collaborative devices. In this second example, n1 and n3 have a complete application component graph (abstract application model), thus, both these devices can initiate the offloading decision-making process. Generally, any device in the collaboration could initiate offloading as long as it has a complete and up to date application component graph.

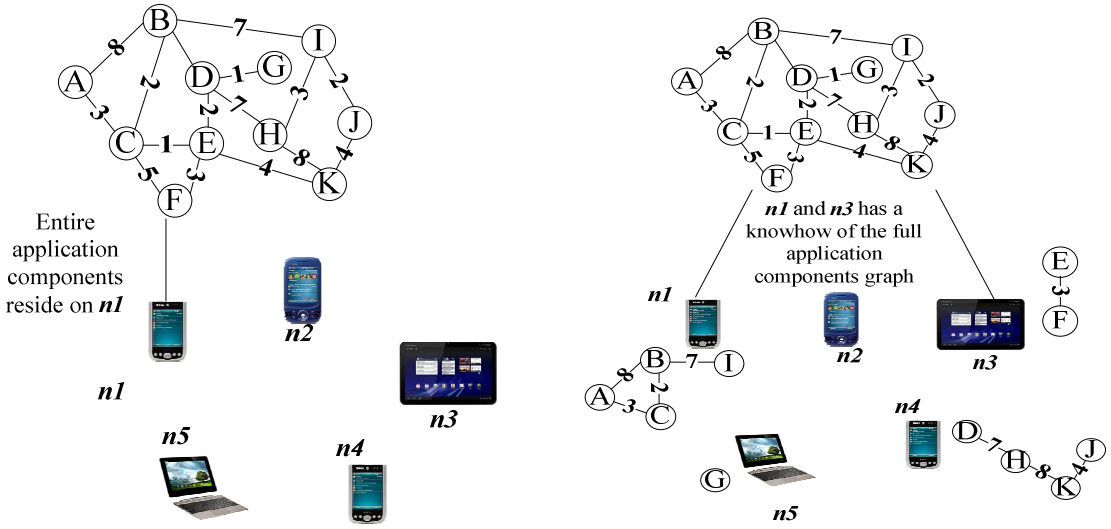


Figure 1. Example of application component graph and potential collaborating devices. A) The entire application resides on device n1. B) Application components are distributed across devices n1, n3, n4 and n5; devices n1 and n3 have a complete application component graph (abstract application model).

3.2 Optimization Model

Table 1 lists the parameters used in the following optimization model. As discussed above, the objective of the optimization model is to maximize the Time to Failure (TTF) of collaborative devices due to energy depletion while meeting the application performance constraints. In another words, the objective function will optimize the energy used on individual devices by allocating the best possible (least energy consumption) components to the least powerful devices to extend the lifetime of devices, while taking into account run time performance constraints, device energy and memory. The amount of resources dedicated to the collaboration can be specified on a per device basis.

Specifically, TTF for a particular device is determined by the computation energy of the components residing and executing in that device; communication energy (energy cost of transferring code modules and invocation data) between local and remote components; and the available energy, which is generally either a specified fraction or the complete remaining battery capacity of the device.

Each collaborative device specifies the amount of energy share (ES) to be available for collaboration. Hence, the TTF of a

particular device in the collaboration can be computed as the ES of the node divided by the sum of all component energy computation rates (ECR) on the device and energy transfer rates (ETR) that transfer data (parameters and results) between coupled remote components.

Table 1. METRICS USED IN PROPOSED OPTIMIZATION MODEL

Metric Symbols	Description	Unit
Energy Share (ES)	The energy shared by a device for collaboration	J
Energy Computation Rate (ECR)	The rate of computational energy used by components	J/s
Energy Transfer Rate (ETR)	The rate of energy to transfer data between components residing in different devices	J/s
Number of Invocations (NI)	Number of invocations to/from a component	Integer
Size of Serialized Parameters (SSP)	Size of serialized parameters and return values	Byte
Remote Invocation frequency (RIF)	Remote component invocation frequency	Integer/ms
Network Energy Cost (NEC)	Device energy cost of the network per KB of data	J/byte
Time (T)	Time taken to execute the component in a given device	ms
Network Availablity (NA)	Device network bandwidth available for data transfer	Byte/ms
Memory Utilization (MU)	Component memory utilization	Byte
Memory Availablity (MA)	The memory availability of a device	Byte

Component ECR is the total energy consumption of a component on a specific device at a given period of time i.e.

$$\mathrm{ECR_{vn}} = \mathrm{CCE_{vn}} \times RIF/MD \ , \forall \mathrm{v} \in \mathrm{V} \text{ and } \forall \mathrm{n} \in \mathrm{N} \quad (1)$$

Where, CCEvn is component computation energy of node N and MD is the measure duration of component execution.

ETR is the aggregate energy consumption rate of data transfer between a component (u) and coupled remote component (v).

$$\mathrm{ETR_{u,v}} = \mathrm{SSP_{u,v}} \times \mathrm{RIF_{u,v}} \times \mathrm{NEC_n} \ , \forall \mathrm{n} \in \mathrm{N} \text{ and } (\mathrm{v},\mathrm{u}) \in \mathrm{E} \quad (2)$$

Generally, the time to failure of a particular device is the aggregate time cost of both computation (CT_n) and network transfer (NT_((v,u)n)).

Firstly let us consider, computation time cost for an individual component v,

$$\mathrm{CT_{vn}} = \frac{\mathrm{ES_n}}{\mathrm{ECR_{vn}}}, \forall \mathrm{n} \in \mathrm{N}, \forall \mathrm{v} \in \mathrm{V}, \quad (3)$$

For all components in particular device (CT_(n)) it will be

$$\mathrm{CT_n} = \frac{\mathrm{ES_n}}{\sum_{\mathrm{v}\in\mathrm{V}} \mathrm{ECR_{vn}}}, \forall \mathrm{n} \in \mathrm{N} \quad (4)$$

In the above equation (4) the denominator summation of ECR produces a ratio and thus the optimization model is non-linear, since on the given node the total cost of component ECR is determined by whether or not the component executes on that node, which is associated with the decision variables in the optimization model. To formulate the model as a linear optimization equation, we have to represent the total CTn of a device in terms of individual component computational times (CTvn). This is also true for the network time cost of component.

However, the CTn of all components in terms of CTvn is not a straight forward summation of individual component computation costs of CTvn,. Rather, the reciprocal sum of individual component computation times is equal to the reciprocal or multiplicative inverse of the total computation time cost of a particular device, i.e.

$$\mathrm{CT_n} = \frac{\mathrm{ES_n}}{\sum_{\mathrm{v}\in\mathrm{V}} \mathrm{ECR_{vn}}} = \frac{1}{\sum_{\mathrm{v}\in\mathrm{V}} (\mathrm{ECR_{vn}}/\mathrm{ES_n})}, \forall \mathrm{n} \in \mathrm{N} \quad (5)$$

Due to the above case, instead of maximizing the time to failure of devices in the optimization model, we minimize the reciprocal of TTF and take the multiplicative inverse of the result. Namely, for some variable x, maximization of x is equivalent to the reciprocal of minimization of 1/x for all positive x.

Hence, for an individual device n, the objective is to minimize the reciprocal of TTF of a device, which is mathematically expressed as follows:

$$minimize \ (\sum_{v\in V} I_{vn} \times {}^{ECR_{vn}}/_{ES_n} + \sum_{(u,v)\in E} |I_{un} - I_{vn}| \times {}^{ETR_{u,v}}/_{ES_n}) \quad (6)$$

Where the indicator variable I is the decision variable, whereby when the model executes it will be 1(I_v=1), If the component v is allocated to a given node , otherwise it will be 0 (I_v=0)).

Therefore, for all Collaborative devices:

$$minimize \ (\sum_{\mathrm{n}\in\mathrm{N}} (\sum_{\mathrm{v}\in\mathrm{V}} \mathrm{I_{vn}} \times \mathrm{ECR_{vn}} / _{\mathrm{ES_n}} + \sum_{(u,v)\in E} |I_{un} - I_{vn}| \times {}^{ETR_{u,v}}/_{\mathrm{ES_n}})) \quad (7)$$

However, even though this model minimizes the total overall sum of the reciprocal of TTF, it could deplete some device's energy faster than others. In other words, it only maximizes the total TTF. To alleviate this limitation we model the objective function using the min-max [36] multi-objective optimization techniques so that it minimizes the reciprocal of TTF of all Iuv devices.

$\forall \mathrm{n} \in \mathrm{N}$ the above equation (7) is expressed as a min-max optimization as follows:

$$minimize\ (\ max_{n=1,\dots,N}\ (\ \sum_{v\in V} I_{vn} \times \mathrm{ECR}_{vn}\ /_{\mathrm{ES}_n} + \sum_{(u,v)\in E} |I_{un} - I_{vn}| \times {ETR_{u,v}}/_{\mathrm{ES}_n}\ ))\ (8)$$

Next, this Min-Max objective function has to be transformed to linear program form. To do so, let us represent the max expression of (8) that resides in the brackets with the variable Z.

$$Z \geq\ (\ \max_{n=1,\dots,N}\ (\ \sum_{v\in V} I_{vn} \times {\mathrm{ECR}_{vn}}/_{\mathrm{ES}_n} + \sum_{(u,v)\in E} |I_{un} - I_{vn}| \times {\mathrm{ETR}_{u,v}}/_{\mathrm{ES}_n}\ ))\quad (9)$$

Since some variable Z is greater than or equal to the maximum of any elements, it is therefore greater than or equal to all elements, and thus expression (9) is equivalent to (10).

$$Z \geq ((\ \sum_{v\in V} I_{vn} \times {\mathrm{ECR}_{vn}}/_{\mathrm{ES}_n} + \sum_{(u,v)\in E} |I_{un} - I_{vn}| \times {ETR_{u,v}}/_{\mathrm{ES}_n}\ ),\ \forall n \in N \quad (10)$$

As a result, the final objection function that we wish to minimize is summarized as:

minimize Z

Subject to the following constraints

1. Derivative constraint from objective function

$$Z \geq ((\ \sum_{v\in V} I_{vn} \times {\mathrm{ECR}_{vn}}/_{\mathrm{ES}_n} + \sum_{(u,v)\in E} |I_{un} - I_{vn}| \times {ETR_{u,v}}/_{\mathrm{ES}_n}\ ),\ \forall n \in N \quad (10)$$

2. The application must execute within the required performance constraint(PC) of the components: i.e

$$\sum_{n\in N}\sum_{v\in V}(I_{vn}) \times T_{vn} + \sum_{n\in N}\sum_{(u,v)\in E}|I_{un} - I_{vn}| \times {SSP_{u,v} \times NI_{u,v}}/{NA_n} \leq \mathrm{PC} \quad (11)$$

This constraint compels the model to execute within the performance requirement of the application, which is specified by the user or calculated by the system relative to the time taken to execute all components on the local device. The first term of the constraint is the summation of the execution time of the components, which is the total time taken to execute components residing on the local device. The second term is the summation of networking time cost, which is the total time it takes to transfer necessary data with remote components executing in other devices.

The above constraints only enforce the performance requirement from the perspective of the whole application, thus for an interactive application, where results are delivered to the user on a per operation basis, this may not be appropriate. Thus, we formulate alternative performance constraints that check with respect to operation invocations based on the original location of the component.

For all components in all devices, the component computation time and data transfer time for all invocations, must be less than the time taken to execute if the component executes in its original location plus some percentage of acceptable delay (e.g. + 10%).

$$(I_{vn}) \times T_{vn} + \sum_{(u,v)\in E}|I_{un} - I_{vn}| \times {SSP_{u,v} \times NI_{u,v}}/{NA_n} \leq PC_v, \forall v \in V\ and\ \forall n \in N \quad (12)$$

3. Memory usage of the components cannot exceed the available capacity on each device

$$\sum_{v\in V} I_{vn} \times MU_v \leq MA_n\ , \forall n \in N \quad (13)$$

This constraint ensures that the allocation of components to collaborative devices is within the total memory space shared to the collaboration (up to the maximum device memory capacity). Each device determines this shared memory size. Note that memory constraints could be ignored if we consider virtual memory with paging.

4. Un-offloadable components, e.g. those which interact with local resources such as UI, sensors, private or local data etc., cannot be allocated for remote execution.

$$I_{vn} \geq L_{vn}\ , \forall v \in V\ and\ \forall n \in \quad (14)$$

For un-offlodable (local) components L_vn = 1, otherwise L_vn = 0 meaning that the component can be executed on any device including the local device.

5. Each component is unique and can run only in one node:

$$\sum_{n\in N} I_{vn} = 1,\ \forall v \in V \quad (15)$$

Note that this constraint could be removed in future work if replication is considered, for example as proposed by Katmon and Ryan [8].

4. Evaluation

This section presents an empirical evaluation of the global adaptive offloading decision-making algorithm from section 3.

4.1 Experimental Setup

In order to address a range of application behaviours and environmental operating conditions, and thus simulate real world collaboration scenarios, we generate probabilistic undirected graphs based on the Erdos and Reyni graph G(V,P)model [37]. Specifically, vertices V of the graph G are application components and P is the connective probability that any two vertices will form an edge, i.e. any two components are coupled by an execution or data dependency.

The empirical evaluation involves either 2, 4, 8, 12, 16 collaborative devices, with application component graphs containing offloadable and unoffloadable components ranging from 8 to 104 incremented by 8. When the application graph is generated, connectedness is ensured in order to evaluate the performance of the proposed model with fully un-partitioned application graphs. A probability value of 0.1 was used for application graphs containing 8 to 48 components and 0.05 for 56 and above, so as to ensure graph connectivity without resulting in excessive density or at the extreme case, a fully connected mesh graph.

The evaluation involved a laptop PC and an Android based mobile device with the following specifications 1) an Intel Core i5 2.30GHZ laptop with 4GB RAM running Windows 7, and 2) a Samsung Galaxy s2 Smartphone with 1.2GHZ

processor and 1GB RAM, running Android OS version 4.1.2. Both devices were setup under controlled conditions with non-essential services and applications halted or removed. On the PC, the evaluation was done using a commercial optimization tool, IBM ilog CPLEX [38] to obtain the best possible results. Since this tool is not available on the Android platform, we used lp_solve [39], which is an open-source mixed integer linear programming (MILP) solver.

To emulate the heterogeneity of participating devices including computation capability, energy consumption, and wireless standards, we used ranges of parameters for the component and environmental metrics used in the optimization model experiments. For example, a range from 3,600 to 36,000 joules was used for the value of participating device energy share metrics which is 1 Wh (Watt-hours) to 10 Wh of energy (shared battery capacity), which represents low power mobile devices through to laptop PCs. For device network energy consumption (NEC) we used a range from 0.2 to 50 milli joules per 1 kilobyte of data transfer. To provide realistic values for NEC and to be inclusive of a wide range of network connectivity and sensors, we profiled data transfer energy consumption on the two previously specified devices using synthetic benchmarks and the IEEE 802.11n radio type. On average, consumption ranged from 0.82 to 2.37 milli joules per 1 kilobyte on the laptop and from 1.08 to 3.46 milli joules on the Android device. This supported the established NEC ranges (0.2 to 50 milli Joules) which represent different communication technologies and connectivity conditions. Table 2 shows all the environmental metrics ranges used. Note that in a real operational system, rather than being pre-specified, metrics would be collected online, while being potentially augmented with historical and predicative data.

Table 2. Metrics Value used in EVALUATION of proposed optimization model

Metrics	Value Range
ES	[3600 , 36000] J
ECR	[0.1, 10] J/s
NI	[1,10]
SSP	[0.01, 10] KB
RIF	[1,3] per second
NEC	[0.0002,0.05] J per 1 KB
T	[1,20] s
NA	[160kb/s to 4 Mb/s] ~ [20,512] KB/s
MU	[1,050,000] KB

4.2 Results

Since the formulated model is based on linear programming, it produces the global optimal distribution of components among collaborative devices in terms of maximizing the time to failure of participating devices within the specified performance and resource constraints. Nevertheless, in addition to evaluating performance and scalability we also evaluate its efficacy against a random distribution.

To evaluate scalability we measured the runtime performance and energy overhead of calculating the optimization for increasing numbers of collaborating devices and application components (as described in the Experimental Setup Section 4.1 above). The results are presented in the following four subsections. Section 4.2.1 compares the efficacy of proposed time to failure optimization with random component distribution. Section 4.2.2 evaluates the run time performance on the laptop PC with various levels of connectivity graphs. Section 4.2.3 shows the run time performance overhead of the optimisation on the Android-based device. Finally, Section 4.2.4 presents the energy overhead of the TTF optimisation on the Android-based mobile device.

4.2.1 Efficacy

The efficacy of the proposed time to failure model is measured by the degree to which component distributions satisfy the objective of maximizing all participating devices time to failure while attaining the performance and other constraints specified in the optimization model section 3.2. Results are compared to a random component distribution. As an example, Figure 2, shows the time to failure distribution of all participating devices in collaborations of up to 8 devices. As expected by the min max optimization, the time to failure of each participating device is relatively smooth and uniform, thereby meeting our goal for an opportunistic community cloud, which is to maximize the collaboration duration for all devices while meeting the performance constraints. In contrast, the random distributions are uneven with a high TTF for 1 or 2 devices (where either few components, or low power consuming components, are distributed to high capacity devices). However the rest of the participating devices have lower TTF due to many or high energy consuming components executing on less capacious devices. This would reduce performance and lead to subsequent adaptation as each device fails and would also mean that users of failed devices can no longer interact with the application (e.g. run a UI component). This is due to the lack of global view across all participating devices in the random model, whereas the proposed min-max model has a global focus not an individual device TTF. Furthermore, the standard deviation and mean in Figure 3 and Figure 4 respectively, clearly shows the achieved uniformity based on the optimised distribution of components in the proposed model.

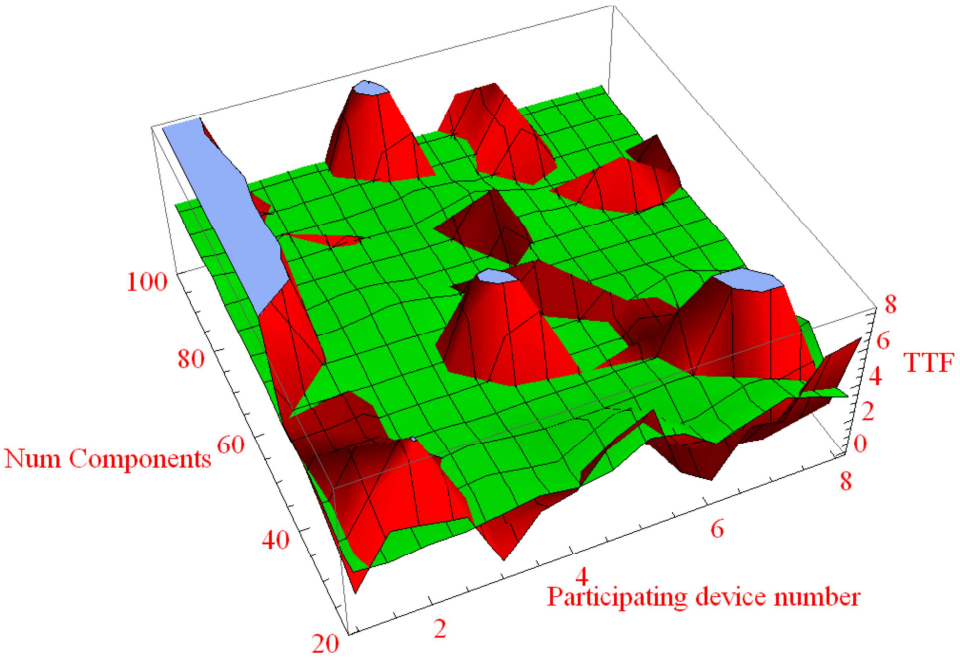


Figure 2. TTF of Participating devices of proposed model (top smooth surface) vs random component distibution (uneven surface with most parts underneath)

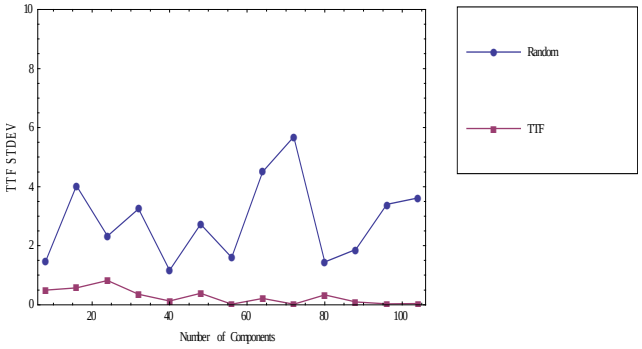


Figure 3. Standard devation of proposed vs random component distibution model.

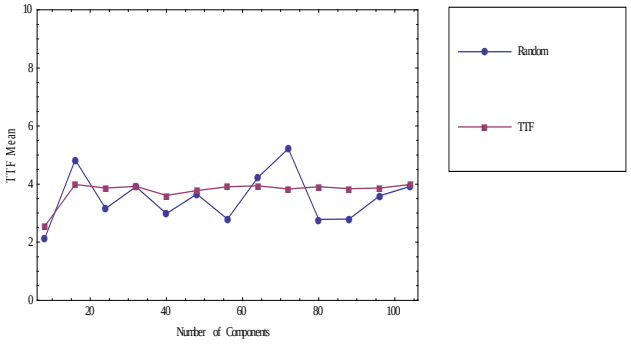


Figure 4. Mean of proposed vs random component distibution model.

4.2.2 Runtime Performance in PC

Figure 5, shows the runtime performance overhead of the formulated model when executing on the laptop. It can be observed that the runtime performance of the formulated model is consistent and fast for medium collaborations. For example, with a collaboration of 16 devices and 56 application components, the TTF model produced the optimal component distribution within a second. When the application size increases, the performance overhead will also increase and adaptation times become more inconsistent due to the variable nature of mixed linear integer programming solver performance [40]. Nevertheless, Figure 5 shows that adaptation was performed in consistent and affordable time for a range of application and collaboration sizes. Furthermore, by parallelizing the adaptation computation (solving the optimization model) and distributing it across multiple peers, it is possible to decrease the performance overhead and manage larger applications and collaborations. Note that time taken to execute an optimisation for a given application has a direct influence on the level of granularity that is used for offloading. i.e. for a large application with many application components, a higher granularity such as class or service level would need to be used. Whereas for a small application with only a few components, instance or method level could be used which gives greater offloading flexibility and thus efficacy. This is discussed further in the context of the mobile device experiment in section 3 below.

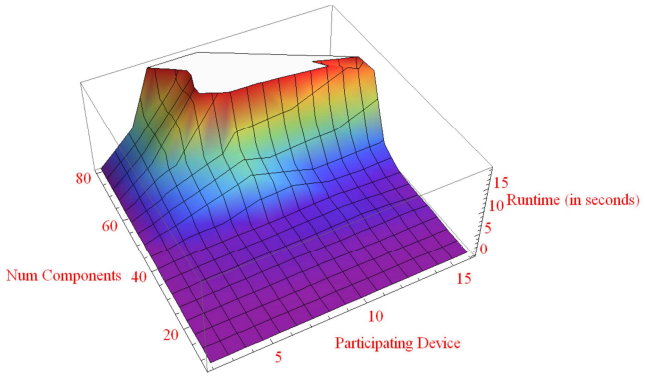


Figure 5. Performance of TTF model in laptop.

4.2.3 Runtime Performance Overhead in Mobile-device

Figure 6, shows the runtime performance of executing the optimisation model on the Android based mobile device. We evaluated 2, 4, 6 and 8 device collaborations with 8, 16, 24 and 32 application graph sizes. As expected, the runtime performance of TTF adaptation is considerably higher than on the PC but is still feasible for smaller collaborations or higher component granularity (e.g. service level).

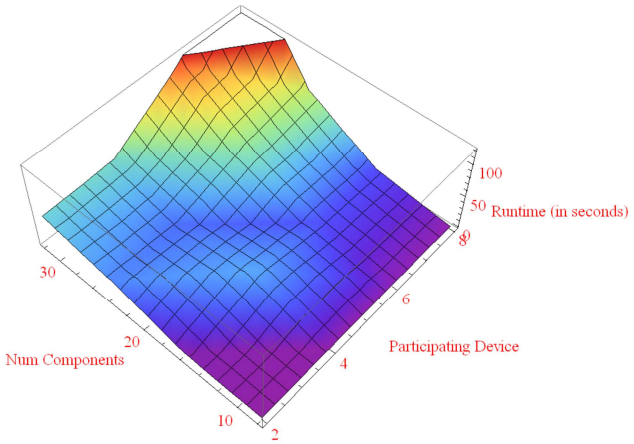


Figure 6. Performance of TTF adaptation in mobile device.

For example, with a collaboration of 6 devices with 24 application components, the TTF model takes 53 seconds to find the optimal component distribution solution. While 24 components is a small number when using fine granularity such as method individual component instance, it could be used to manage a substantial application if coarse-granularity such as class or service level is used. This is intuitive since 24 services could provide considerable application behaviour, although it would reduce the adaptation granularity and thereby reduce the efficacy of the solved adaptation placement topology [32]. In addition, the optimisation performance difference between the PC and Android-based device was exacerbated by the optimisation framework used. Namely, lp_solve is significantly slower [36], but was used since CPLEX was not available on the Android platform.

4.2.4 Energy overhead in Mobile Device

To measure the energy overhead of the TTF adaptation model in the Android device, we instrumented the solver model to record the energy utilization required to solve each instance of application adaptation. Figure 7, illustrates the energy utilization for the same application and collaboration size specified above in part 3. For instance, in the case of 24 application components with 6 collaborating devices, the adaptation computation consumes 25 Joules of energy which is 0.014% of the total battery capacity of the mobile device (which has 6.11Wh of battery capacity on a full charge). However, obviously this energy utilization overhead increases as the runtime to solve the TTF model increases. It should be noted that in a community cloud it would where possible be desirable to perform the application adaptation decision making (TTF optimisation process) on a more energy copious device such as a dedicated server or PC connected to mains AC power. Nevertheless, the runtime and energy overhead result shows the proposed TTF model can be solved in affordable time and energy on energy constrained devices for a range of application and collaboration sizes, especially where a higher level of offloadable component granularity is used.

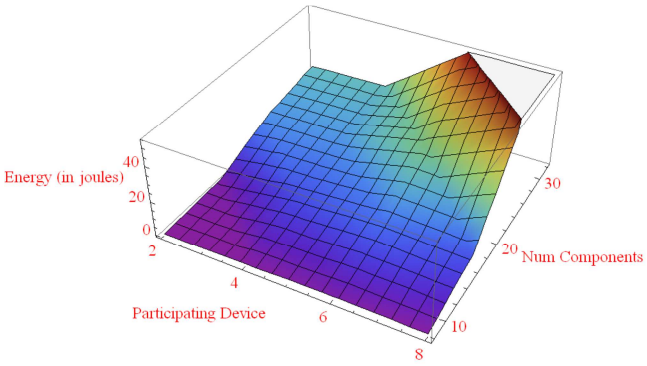


Figure 7. Energy over head of TTF adaptation in mobile device.

5. Conclusions And Future Works

This paper has formulated a global adaptive offloading decision-making model that extends the adaptation lifetime of participating devices in a community cloud (TTF due to energy depletion) by conserving individual device energy while attaining specified application performance requirements. The proposed model enables optimal distribution of context aware application components to create ad-hoc pervasive community clouds as an alternative architecture to proprietary cloud computing. Generally, the strategies proposed here will facilitate the development of a computing system that makes use of inexpensive networking distribution of context aware application components to create ad-hoc pervasive community clouds as an alternative architecture to proprietary cloud computing. Generally, the strategies proposed here will facilitate the development of a computing system that makes use of inexpensive networking sensors, communication, and computing devices distributed among pre-determined or opportunistic communities. As a proof of concept, we evaluated the efficacy, performance and scalability of the formulated with a range of application scenarios. The simulation results demonstrated that optimal component distribution decisions can be made within affordable time and energy for collaborations of useful size (within the constraint of choosing an acceptable level of offloading granularity) using contemporary pervasive devices.

Since the focus of this paper was formulating an efficient global offloading decision-making algorithm, other issues like candidate selection, context sharing model, and collaboration incentives were not addressed and thus are left to future work. Furthermore, future work will aim to implement and empirically evaluate the model with real test applications to complement the synthetic evaluation presented in this paper.

6. References

[1] D. J. Cook and S. K. Das, "Pervasive computing at scale: Transforming the state of the art," Pervasive and Mobile Computing, vol. 8, pp. 22-35, 2012.

[2] A. Marinos and G. Briscoe, "Community Cloud Computing," in Cloud Computing. vol. 5931, M. Jaatun, G. Zhao, and C. Rong, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 472-484.

[3] F. Zambonelli, "Pervasive urban crowdsourcing: Visions and challenges," presented at the IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops),, 2011.

[4] T. A. Team. (2011). Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. Available: http://aws.amazon.com/message/65648/

[5] J. Modine. (2008). Web startups crumble under amazon s3 outage Available: http://www.theregister.co.uk/2008/02/15/amazon_s3_outage_feb_2008/

[6] W. F. James M.Kaplan, Noah kindler. Revolutionizing data center energy efficiency [Online]. Available: http://www.ecobaun.com/images/Revolutionizing_Data_Center_Efficiency.pdf

[7] P. Jones, " INDUSTRY CENSUS 2012: EMERGING DATA CENTER MARKETS," ed, October 2012.

[8] J. Baliga, R. W. A. Ayre, K. Hinton, and R. Tucker, "Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport," Proceedings of the IEEE, vol. 99, pp. 149-167, 2011.

[9] L. Jie, Z. Feng, L. Xue, and H. Wenbo, "Challenges Towards Elastic Power Management in Internet Data Centers," in Distributed Computing Systems Workshops, 2009. ICDCS Workshops '09. 29th IEEE International Conference on, 2009, pp. 65-72.

[10] R. Courtland, "The high stakes of low power," Spectrum, IEEE, vol. 49, pp. 11-12, 2012.

[11] M. Jarus, S. Varrette, A. Oleksiak, and P. Bouvry, "Performance Evaluation and Energy Efficiency of High-Density HPC Platforms Based on Intel, AMD and ARM Processors," in Energy Efficiency in Large Scale Distributed Systems, J.-M. Pierson, G. Da Costa, and L. Dittmann, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 182-200.

[12] B. Smith, "ARM and Intel Battle over the Mobile Chip's Future," Computer, vol. 41, pp. 15-18, 2008.

[13] S. Citro, J. McGovern, and C. Ryan, "Extending Real Time Mobile Collaboration Algorithms to Handle Membership Events in an Ad-Hoc Mobile Network," in 2nd International Conference Collaborative Computing: Networking, Applications and Worksharing, Atlanta, Georgia, 2006, pp. 1-9.

[14] M. Conti, S. K. Das, C. Bisdikian, M. Kumar, L. M. Ni, A. Passarella, G. Roussos, G. Tröster, G. Tsudik, and F. Zambonelli, "Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence," Pervasive and Mobile Computing, vol. 8, pp. 2-21, 2012.

[15] M. Satyanarayanan, "Pervasive computing: vision and challenges," Personal Communications, IEEE, vol. 8, pp. 10-17, 2001.

[16] O. Holder, I. Ben-Shaul, and H. Gazit, "Dynamic layout of distributed applications in FarGo," presented at the Proceedings of the 1999 International Conference on Software Engineering, 1999.

[17] G. C. Hunt and M. L. Scott, "The Coign automatic distributed partitioning system," presented at the Proceedings of the third symposium on Operating systems design and implementation, New Orleans, Louisiana, United States, 1999.

[18] V. Krishnaswamy, I. B. Ganev, J. M. Dharap, and M. Ahamad, "Distributed object implementations for interactive applications," presented at the IFIP/ACM International Conference on Distributed systems platforms, New York, New York, United States, 2000.

[19] G. Xiaohui, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments," presented at the Proceedings of the First IEEE International Conference on Pervasive Computing and Communications. (PerCom 2003).

[20] D. S. Miloji, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," ACM Comput. Surv., vol. 32, pp. 241-299, 2000.

[21] H. Dong, W. Ping, and D. Niyato, "A Dynamic Offloading Algorithm for Mobile Computing," Wireless Communications, IEEE Transactions on, vol. 11, pp. 1991-1995, 2012.

[22] S. Ou, K. Yang, and A. `Liotta, "An Adaptive Multi-Constraint Partitioning Algorithm for Offloading in Pervasive Systems," in Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications, 2006, pp. 116-125.

[23] S. Ou, K. Yang, and J. Zhang, "An effective offloading middleware for pervasive services on mobile devices," Pervasive and Mobile Computing, vol. 3, pp. 362-385, 2007.

[24] H. Christian, "Runtime Locality Optimizations of Distributed Java Applications," presented at the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), 2008.

[25] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading for pervasive computing," Pervasive Computing, IEEE, vol. 3, pp. 66-73, 2004.

[26] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in Proceedings of the 8th international conference on Mobile systems, applications, and services, San Francisco, California, USA, 2010, pp. 49-62.

[27] E. Abebe and C. Ryan, "Adaptive application offloading using distributed abstract class graphs in mobile environments," Journal of Systems and Software, vol. 85, pp. 2755–2769, 2012.

[28] C. Ryan and P. Rossi, "Software, performance and resource utilisation metrics for context-aware mobile applications," presented at the Software Metrics, 2005. 11th IEEE International Symposium, 2005.

[29] X. Changj, L. Yung-Hsiang, and L. Zhiyuan, "Adaptive computation offloading for energy conservation on battery-powered systems," presented at the Parallel and Distributed Systems, 2007 International Conference on, 2007.

[30] [30] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing Performance, Energy, and Quality in Pervasive Computing," presented at the Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02), 2002.

[31] D. Kovachev, Y. Tian, and R. Klamma, "Adaptive Computation Offloading from Mobile Devices into the Cloud," presented at the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA),, 2012.

[32] E. Abebe and C. Ryan, "A Hybrid Granularity Graph for Improving Adaptive Application Partitioning Efficacy in Mobile Computing Environments," in 10th IEEE International Symposium on Network Computing and Applications (NCA), 2011, pp. 59-66.

[33] P. Rossi and C. Ryan, "Empirical Evaluation of Dynamic Local Adaptation for Distributed Mobile Applications," in On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE. vol. 3760, R. Meersman and Z. Tari, Eds., ed: Springer Berlin Heidelberg, 2005, pp. 828-845.

[34] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A Computation Offloading Framework for Smartphones," in Mobile Computing, Applications, and Services. vol. 76, M. Gris and G. Yang, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 59-79.

[35] A. Sagahyroon, "Battery and Power Consumption of Pocket PCs," Journal of Computers, vol. 7, 2012.

[36] H. Aissi, C. Bazgan, and D. Vanderpooten, "Min–max and min–max regret versions of combinatorial optimization problems: A survey," European Journal of Operational Research, vol. 197, pp. 427-438, 2009.

[37] P. E. a. A. Renyi, "On the evolution of random graphs," presented at the Publications of the Mathematical Institute of the Hungarian Academy of Sciences, 1960.

[38] Y. Xiao, Y. Cui, P. Savolainen, M. Siekkinen, A. Wang, L. Yang, Yl, x00E, J., x00E, x00E, A. ski, and S. Tarkoma, "Modeling Energy Consumption of Data Transmission over Wi-Fi," Mobile Computing, IEEE Transactions on, vol. PP, pp. 1-1, 2013.

[39] lp_solve. Available: http://lpsolve.sourceforge.net/5.5/

[40] S. Mahadev, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," Pervasive Computing, IEEE, vol. 8, pp. 14-23, 2009.

A Multi-Platform Workflow Management System optimized for Cloud Computing Platforms

A. Carrión[1], M. Caballer[1], I. Blanquer[1], N. Kotowski[2] and A.M.R. Dávila[2]
[1]Instituto de Instrumentación para Imagen Molecular (I3M),
Centro mixto CSIC - Universitat Politècnica de València - CIEMAT, Valencia, España
[2]Instituto Oswaldo Cruz (IOC), Fundação Oswaldo Cruz (FIOCRUZ),
Rio de Janeiro, Brazil

Abstract— *The scientific experimentation is facing a data deluge in which the amount of data generated is reaching the order of terabytes per day, and thus huge capacity is required to process this data. Computationally, these processes are modelled using Scientific Workflows. However, the execution of a Scientific Workflow can be a complex and resource-demanding task that must be managed by Workflow Management Systems (WMSs). As new computing paradigms emerge and infrastructures evolve, WMSs are extended to support these new computing back-ends. In fact, in the last years, Cloud Computing has appeared as another viable platform for running scientific applications. However, current WMSs are not optimized to exploit key cloud features. For that reason, this work details the design and implementation of a multi-platform WMS with a novel approach for efficiently supporting cloud resources. The engine has been successfully tested in the execution of a comparative genomics workflow called Orthosearch.*

Keywords: Workflow, Workflow Management Systems, Cloud Computing, Comparative-genomics.

1. Introduction

The relation between Science and computing goes back to the 1960s, when powerful computers (supercomputers) were introduced for performing scientific and engineering problems. At that time, a typical experimental scenario consisted in a repetitive cycle of moving data to a supercomputer for processing, submitting the executions and retrieving the outputs from the data storage [1]. Obviously, this process had to be automated for allowing scientists to focus on their research and not in the computational management. Fortunately, at the same time, the business community was addressing how to automate business processes and as a result the *Workflow* concept was born. In the business context, a *Workflow* can be defined as the orchestration of a set of activities in order to accomplish a larger and sophisticated goal. A specialization of this idea was adopted by the research community to model e-Science processes, the Scientific Workflows (SWFs). In this programming model, scientific applications are described as a set of tasks that have dependencies between them. It means that a task will start its execution only when the tasks it depends on have completed their execution.

The execution of workflow applications is a task with many details. A typical workflow is composed of hundreds of tasks that must be executed in a coordinated way. Moreover, all these tasks must be submitted to specific computing resources and the required inputs must be made available to the application. Software in charge of dealing with all these aspects are called Workflow Management Systems.

As new computing paradigms emerge and infrastructure evolve, so do the WMSs that support these computing back-ends. Traditionally, Scientific Workflow Applications have been extensively deployed in high-performance computing infrastructures, such as powerful clusters and supercomputers. Later, a highly distributed infrastructure, the Grid, appeared as an alternative to traditional approaches. In the last years, a new distributed computing paradigm, Cloud Computing, has appeared as another viable [2] platform for running scientific applications. In fact, some of their main features, such as rapid elasticity, resource pooling, and pay per use, are well suited to the nature of scientific applications that experience a variable demand during its execution.

Because the scientific experimentation is suffering from a data deluge phenomenon where the amount of data generated is reaching the order of terabytes per day, a huge amount of resources are needed to process this data and enable research. For that reason, it is crucial that WMSs have multi-platform support. Although current WMSs already support various platforms for the execution of workflow applications, many desirable features of cloud computing are not implemented, such as the dynamic provisioning of resources. For that reason, in this work we present a WMS with multi-platform support but also optimized to execute workflow applications in cloud computing platforms.

The remainder of the paper is structured as follows. Firstly, Section 2 gives an overview of the state-of-the-art WMSs. Afterwards, Section 3 explains in detail the architecture of a novel multi-platform WMS with actual support of cloud computing resources. Section 4 describes the comparative-genomics workflow, Orthosearch, selected as use case and Section 5 the experimentation and results obtained with it. Finally, conclusions and future working lines are exposed.

2. Related work

Due to the crucial role that workflow applications play in the scientific community, most current WMSs were developed to enable the execution of these applications in grid computing platforms. When Clouds became mainstream, WMSs were enhanced to support it. In this section, we present a brief description of the most prominent WMS found in the state-of-the-art which are related with our work.
Pegasus [3] is a mature Workflow Management System that combines features such as portability across a wide range of infrastructures, scalability, data management capabilities, exhaustive monitoring and complex workflow restructuring or transformations. It can be used with popular programming languages among the scientific community (such as Java, Python Perl) through its APIs (application programming interfaces) and also supports submission via web portals. Although it supports multiple cloud providers, it does not dynamic provision resources with different hardware and software requirements.
Taverna [4] is a WMS with a strong focus on bioinformatics where all computational workflow steps are Web Services. Workflows can be designed and executed on local desktop machines through the workbench or through other clients or web interfaces using the server mode. The server supports requests from many users to execute remote workflows with support of both grid and cloud platforms. It uses myExperiment [5] as repository for sharing and reusing workflows and the BioCatalogue and Biodiversity catalogue for Web Services discovery.
Kepler [6] is build upon the mature, dataflow-oriented Ptolemy system. From it, Kepler inherits several features such as the GUI and the Actor-Director model. In Kepler, workflows can be created connecting components called Actors which will process the data. An actor has several Ports for expressing input and output data and the director determines the model of computation used by the workflow. To the best of our knowledge, it only supports distributed execution via Web and Grid services, but not Cloud Computing platforms.
Galaxy [7] is an open, web-based approach that facilitates genomics research. It provides a collaborative environment for performing complex analyses, with automatic provenance tracking, allowing the transparent sharing of computational details, intent and context. Its objective is to offer accessible, reproducible and transparent computational research. A Galaxy instance supports running on compute clusters through Portable Batch System (PBS) and Sun Grid Engine (SGE).
The WMS described in this work distinguishes from the previous ones with features such as multi-cloud (public, private, hybrid) and multi-platform support (clusters and clouds). In addition, it presents a novel approach for the on-demand provisioning of cloud resources with ad-hoc hardware and software requirements. The solution will be released under GPL v3 license and it will be available for downloading at github (https://github.com/abel-carrion) very soon.

3. System architecture

The aim of this section is to describe the design and implementation of the architecture behind the WMS developed. The overall organization of the system is depicted in Figure 1. This schema is based on the one showed in [8], one of the most cited papers about the taxonomy of Grid WMSs. Our architecture is almost identical but extended to support a multi-platform scenario. In fact, our WMS currently supports execution on clusters, public clouds (Amazon EC2, Google Cloud Platform and Microsoft Azure) and private clouds (OpenNebula and OpenStack).

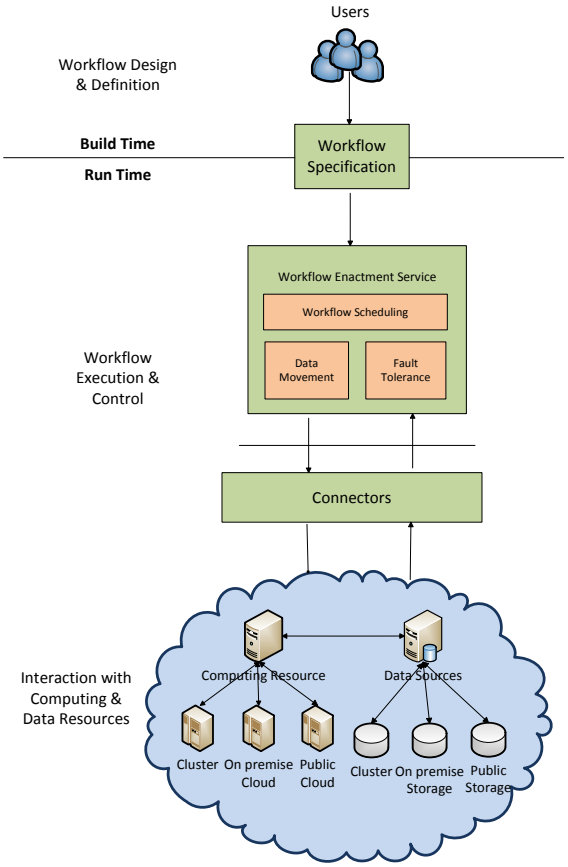


Fig. 1: WMS architecture.

3.1 Design principles

The key principles of our architecture are:

- **Platform-agnostic client**. The client program has been developed using a platform-agnostic programming lan-

guage and thus can be used in a wide spectrum of Operating Systems.
- **Generality**. It should be possible to execute any kind of workflow application that can be expressed as a Directed Acyclic Graph.
- **Extensibility**. The architecture can be extended to include new functionality such as support for a new computing and/or storage back-ends.
- **Modularity**. A change on a part of the system should not require changes on the rest of the system if the interfaces are preserved.
- **Multi-platform**. Each part of the workflow can be executed using different computing back-ends.
- **NIST Cloud Computing definition compliant**. When using cloud resources, the system follows the requirements expressed by the National Institute of Standards and Technology (NIST) cloud computing definition.

3.2 Workflow structure

In our system, workflows applications are composed of a number of tasks which have data dependencies (in the form of files) between them. A task depends on the output(file(s)) of one or more tasks to be used as its input. Only when the inputs of the task are available, it will start its execution.
In formal terms, these workflows can be represented by a Directed Acyclic Graph (DAG) where the nodes represent computational tasks and the directed edges the dependencies between them.

3.3 Workflow specification

A workflow specification (also called workflow model) defines a workflow including its task definition and structure (task connectivity). There are two types of workflow models: abstract and concrete (executable).

3.3.1 Abstract workflow

The abstract workflow specification is a template that describes the tasks that must be executed and for each of these tasks, the inputs, outputs, commands and arguments to be used when invoked. The resources-independent nature of these descriptions has two benefits: firstly, workflows can be ported to different computing infrastructures and secondly, these templates can be shared between users working on the same field of interest. Listing 1 shows a JSON template of a test workflow. According to this template, the workflow executes one stage named *process0* that must be executed in the front-end machine (*ramses*) using as Operating System the *64-bit* version of *Ubuntu*. The requirements of the process are deploying *4 single-core* machines with *4GB* of memory and one disk of *20 GB*. The execution of the stage requires invoking, for each node, the program *test* using as argument the filename of *one* file contained in *input0*. The output of the stage (and also of the workflow) are all .txt files generated by the program. Notice that all the values that start with # are references to JSON objects defined in the same file or the resource configuration file described below.

```
{
  "stages": [
    {
      "id": "process0",
      "hostId": "#ramses",
      "environmentId": "#ubuntu64bit",
      "nodes": [
        {
          "numNodes": "4",
          "coresPerNode": "1",
          "memorySize": "4096m",
          "disks": [
            {
              "nDisk": "0",
              "diskSize": "20g"
            }
          ]
        }
      ],
      "execution": [
        {
          "path": "./test",
          "arguments": "#input0(1)"
        }
      ],
      "stageIn": [
        {
          "id": "#input0"
        }
      ],
      "stageOut": [
        {
          "id": "output0",
          "type": "File",
          "filterIn": "*.txt",
          "replica": "none"
        }
      ]
    }
  ]
}
```

Listing 1: Workflow template example

3.3.2 Resource information file

To convert the abstract workflow into a concrete workflow or executable workflow, the WMS needs information about the hosts, the environments and the input files. This information can be found in a configuration file like the one showed in Listing 2 for the previous workflow. The array *hosts* contains a list with the data for connecting to the front-end hosts such as: the host name, port and different credentials depending on the platform to be used (for instance, a certificate for Windows Azure and a user/password pair for a cluster). *Environments* is an ad-hoc field for cloud platforms that defines the required features of the VMI (Virtual Machine Image) to use as a base to create the VMs and the software packages that should be installed on it. VMIs are obtained from the image repository associated to each deployment. Finally, *inputFiles* declares the input files of the workflow: the identifier, the type (File, Parameter, etc.) and the physical location (URI).

```
{
  "hosts": [
    {
      "hostId": "ramses",
      "type": "Cloud",
      "subType": "OpenNebula",
      "hostName": "ramses.i3m.upv.es",
      "port": "1111",
      "credentials": {
        "userName": "userName",
        "password": "passWord"
      }
    }
  ],
  "environments": [
    {
      "environmentId": "ubuntu64bit",
      "osName": "linux",
      "arch": "x86_64",
      "osFlavour": "ubuntu",
      "osVersion": "14.04",
      "packages": [
        "unzip"
      ]
    }
  ],
  "inputFiles": [
    {
      "id": "input0",
      "type": "File",
      "values": [
        "db.zip"
      ],
      "extract": "true"
    }
  ]
}
```

Listing 2: Configuration file

3.4 Workflow validation

The first step is parsing and validating the workflow template and the resource information file with a JSON processor. Once both documents have been analysed, the next step is to carry out the semantic validation by cross-validating the information provided in both files. The WMS implements a semantic validator which checks if the documents meet different rules. For instance, some fields only allow concrete types of values (memory is an integer followed by the characters 'm', 'g' or 't'). Moreover, every reference in the workflow template should exist in the resource configuration file. If any rule is violated, the system prompts to the user the erroneous file and line.

3.5 Workflow planning

The mapping or planning process distinguishes our WMS from other systems by providing a novel approach that dynamically provisions cloud computing resources. It uses the information of the resource configuration file for transforming the abstract workflow into an executable workflow. The mapper component of the system adds tasks for deploying cloud resources only when they are needed and tasks for undeploying them when their outputs have been staged-out. In addition, data management tasks are added for data staging in/out the required input by the tasks or output to the user selected location, respectively. Figure 2 shows the mapping process of a simple workflow with three tasks (A, B and C) to an executable workflow where A and C are deployed on cloud resources and B in a cluster. Although next subsection gives an extensive explanation of each task, the planner produces four types of nodes: deploy, copy, undeploy, cleanup and copyout.

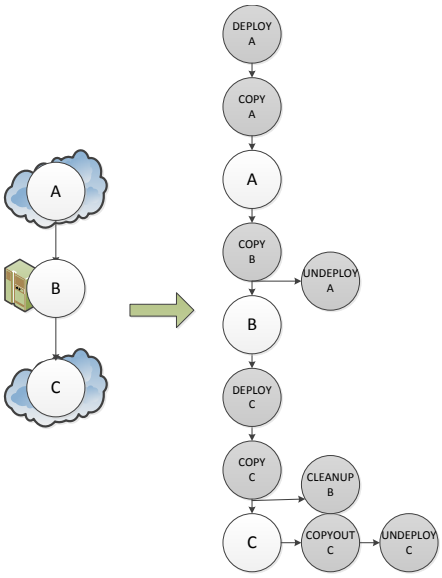


Fig. 2: Planner conversions.

3.6 Workflow execution

Once the mapper has produced the executable, it is submitted to the workflow execution engine. The execution of the workflow begins with the initialization of every element: the state of the tasks are set to *IDLE* and the state of the inputs/outputs to *DISABLED*. Next, due to the data-flow nature of the workflow system, the inputs provided by the user are *ENABLED*, allowing the execution of the first task(s). The workflow execution engine is controlled by two core functions: *runTask* and *getStatus*. The runtime checks if all the inputs of a task are enabled, calling *runTask* in that case. When a task is submitted, the engine periodically monitors its status through the *getStatus* function and if it has finished successfully, enables the outputs of the tasks (which in turn are normally inputs of the next tasks). Obviously, the behaviour of runTask and getStatus will vary according to infrastructure (cluster and cloud) and the task type (deploy, copy, user-defined, undeploy, cleanup or copyout).

3.6.1 Deploy task execution

The execution of a deploy task is required when the user desires to execute a task of the abstract workflow in a cloud platform. In order to dynamically deploy cloud

computing resources, the system makes a request to the IM (Infrastructure Manager) [9]. The main function of the IM is to deploy and automatically configure the virtual infrastructure required to execute and manage an application in a cloud computing environment, expressed in a RADL (Resource and Application Description Language) document.
On the one hand, the *runTask* function in a deploy task calls the deploy function of the IM API. Prior to it, the system needs to build a RADL document, using the hardware and software requirements of the task expressed in the JSON document. By means of a RADL document, the WMS calls the IM to configure the deployment as a Portable Batch System (PBS) cluster where all nodes share the same disk via NFS. In this manner, PBS acts as the scheduler of the jobs that the task should execute. On the other hand, the *getStatus* invokes the API function that queries the status of the infrastructure. The task is considered to be finished when the status returned by the API is *configured*. From this point on, the WMS interacts with the cloud infrastructure through SSH, using the information returned by the API (public IP and user credentials).

3.6.2 Copy task execution

The copy task is in charge of the data management during the execution, one of the most crucial parts of any WMS. Moreover, these tasks are executed regardless of the computing platform used (cluster or cloud). With respect to the input data, the system can download any file that can be retrieved with the protocols supported by the unix *wget* command (http, https and ftp). Another important feature is the possibility of explicitly indicating that the input files should be extracted on the destination resources. However, since there are tools that require compressed data as input, this extraction should be optional. In any case, the stage-in of an input file triggers the submission of a job to the physical or virtual cluster scheduler for downloading the file and next, if it is required, extracting the file.
The other type of stage-ins are the intermediate results produced by previous tasks in the DAG. To handle the transference of this kind of data, the WMS submits a basic job that invokes the scp (Secure Copy Protocol) program with the corresponding credentials and arguments.
The goal of *getStatus* in a copy task is to make sure that all the jobs submitted by *runTask* have finished successfully.

3.6.3 User-defined task execution

In contrast to the previous tasks, user-defined tasks are the same that appear in the abstract workflow specification but now they are executable. In our WMS, a user-defined task is said to be executable when two conditions are met: firstly, the target infrastructure is already available (the cluster is accessible or the cloud computing platform is deployed), and secondly, the input data needed by the tasks has been staged-in to these resources. As it can be appreciated, both conditions correspond to the actions performed by the deploy task and copy task, respectively.
According to the abstract workflow, a task can contain a block of executions or commands to execute. When the *runTask* function is invoked for this kind of tasks, the WMS analyses the commands to determine if there is parallelism in the submission of the job or not. The parallelism of a task is explicitly indicated by the user in the abstract workflow, appending the "*(x)*" expression to an argument where *x* is the granularity. The granularity refers to the number of files processed per computing node.

3.6.4 Undeploy task execution

As in the deployment task execution case, the *runTask* function calls the proper function of the IM API, *destroyInfrastructure*.
The aim of *getStatus* in this case is to make sure that the infrastructure removal operation is correctly carried out. This is especially important when public clouds are used to avoid incurring in unnecessary costs.

3.6.5 Cleanup task execution

The cleanup task is the equivalent of the undeploy task but for the case of clusters. Because a workflow task usually generates large of amounts of data and clusters are infrastructures shared with other users, a best practice consists on cleaning up the data once it has been staged-out. Thus, the function *runTask* simply deletes via SSH the whole execution directory created for the task and *getStatus* makes sure that the operation is actually done.

3.6.6 Copyout task execution

From the user's point of view, the purpose of the copyout tasks is to retrieve the data products of the computations. The mapper attaches these special tasks only to the final tasks of the abstract workflow specification (i.e tasks which don't have dependencies with other tasks).
The *runTask* function starts the stage-out of the output to one or more locations. The default action is to transfer the data to the user local space (where the submit host is being executed). If besides the field *replica* of the output contains references to another data storage sites, the data will be also copied to these locations. The other function, *getStatus*, will monitor the data transference until all of them are completed.

3.7 Fault tolerance

Although workflow execution failures in clusters and Cloud Computing platforms are not very common, fault tolerance is of most importance in distributed computing environments. It is necessary to provide the proper fault-tolerance mechanisms to handle failures and support the

reliable execution in the presence of software or hardware failures. Due to the difference in terms of requirements between the tasks that compose a workflow, the WMS defines different fault tolerance policies for each task. The policies simply define the number of retries in case of software failure or hardware failure. The user also has the possibility of indicating such values in the abstract workflow specification, using the object *retries* and its fields *OnWallTimeExceeded*, *OnSoftwareFailure* and *OnHardwareFailure* inside a stage object. If, for some reason, a task exceeds the maximum number of retries for any type of failure, the execution of the workflow is automatically aborted.

3.8 Provenance

Workflow provenance is critical to users to be able to follow the evolution of their executions and to determine the cause behind a failure. In that sense, the WMS implements a logging system that registers in a file the events that occur during the execution along with other important information (for example, the elapsed time of a task that has finished). This information is not only useful for monitoring the progress of the experiment but also for getting performance statistics as we have done in the use case presented in the next section.

4. Use case: Orthosearch

OrthoSearch (Orthologous Gene Searcher) [10] [11] is a genomics comparative workflow. Initially conceived as a Perl-based routine, it is a profile-protein, reciprocal best hits (RBH) based solution for homology inference among species. It comprises several stages and uses distinct bioinformatics tools, such as Mafft [12] and HMMER [13] which confront an orthologous database with an organism multifasta protein data. The workflow structure is depicted in Figure 3.

OrthoSearch was initially built in order to infer homology between Protozoa species, although there is no restriction to any specific genome. It has already been evaluated and proven to be effective when inferring orthology among five Protozoa organisms with two distinct orthologous databases [14], NCBI COG and KOG.

5. Experimentation

5.1 Data selection

We selected a subset of EggNOG database version 4 [15] which comprises eukaryotic ortholog groups only, EggNOG KOG. Its design comprises up-to-date techniques for ortholog groups construction, inparalogs recognition, robust automatic annotation, single-copy ortholog groups nesting and reconstruction of phylogenetic trees based on the generated clusters multiple alignments.

Three Protozoa species were selected to be confronted with EggNOG KOG database: Cryptosporidium hominis, Entamoeba histolytica and Leishmania infantum.

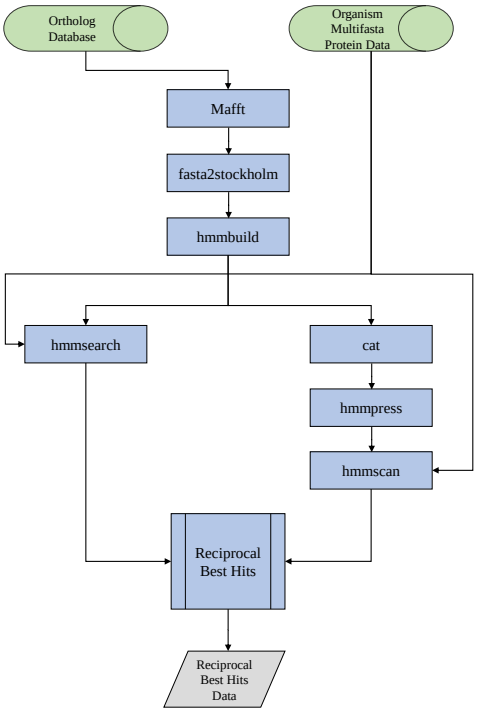


Fig. 3: Orthosearch abstract workflow.

5.2 Sequential execution

Figure 3 displays that the Orthosearch pipeline is composed of 8 stages or processes: Mafft, fasta2stockholm, hmmbuild, hmmsearch, cat, hmmpress, hmmscan and Reciprocal_Best_Hits. Initially, the serialized version of this pipeline was executed using a single Ubuntu 14.04 compute instance with 16 CPU cores, 24GB RAM and 100GB disk.

5.3 Workflow engine execution

After the sequential execution, the workflow was executed using the WMS presented in this work. The first action that the tool carries out is restructuring the workflow according to the parallelism expressed for each stage on the abstract specification. As a result, the original 8 stages of the pipeline were reduced to 4 stages: Mafft/fasta2stockholm/hmmbuild, hmmsearch, cat/hmmpress/hmmscan and Reciprocal_Best_Hits. On the one hand, Mafft/fasta2stockholm/hmmbuild and hmmsearch are stages that admit trivial parallelism and it fits very well the cloud computing execution model. The resources allocated for this task were 16 Ubuntu 14.04 contextualized VMs on a public cloud (supported by OpenNebula). On the other hand, cat/hmmpress/hmmscan requires the execution of a specific parallel version of hmmscan, implemented on MPI. For that reason, a cluster (kahan.dsic.upv.es) was selected for executing this part of the workflow with 6 concurrent processes. The last stage, Reciprocal_Best_Hits

cannot be parallelized and so was executed on a VM with hardware features similar to the one used on the sequential execution.

5.4 Performance results

Table 1 shows a comparison between the response time on the sequential case and the WMS execution for each protozoic organism used.

Table 1: Response time for each scenario.

	Cryptosporidium	Entamoeba	Leishmania
Total time	414 minutes	611 minutes	575 minutes
Speed-up	5X	3,84X	4,13X

In our best scenario, the WMS provided a 5.0 speedup ratio, with a total 6 hours and 54 minutes of execution time. The same experiment, in a sequential approach, required 34 hours and 10 minutes.

6. Conclusions and future directions

The advent of Cloud Computing and its core characteristics (rapid elasticity, resource pooling, and pay-per-use, among others) are well-suited to the nature of scientific applications that experience a variable demand during execution. As a consequence, many WMSs derived from projects in the area of grid computing were updated to support the execution on Cloud resources. However, many of their features are optimized for grids and thus are unable to obtain the most key aspects of clouds, such as dynamic provisioning of resources. For that reason, this work presents a novel multi-platform (clusters and clouds) WMS with support for on-demand provisioning of multi-cloud (public, private and hybrid) customized cloud computing resources.
The tool developed in this work has been tested using a comparative genomics pipeline, called Orthosearch. Promising results were obtained, with significant speedup ratio compared to the batch-oriented pipeline. The current working lines include adding support for Grid computing, using a more efficient transference protocol than SSH and implementing data privacy.

Acknowledgments

This paper wants to acknowledge the support of the EUBrazilCC project, funded by the European Commission (STREP 614048) and the Brazilian MCT/CNPq N° 13/2012, for the use of its infrastructure. The authors would like also to thank the Spanish "Ministerio de Economía y Competitividad" for the project "Clusters Virtuales Elásticos y Migrables sobre Infraestructuras Cloud Híbridas" with reference TIN2013-44390-R.

References

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the Use of Cloud Computing for Scientific Workflows," in *2008 IEEE Fourth International Conference on eScience*, pp. 640–645, IEEE, Dec. 2008.

[3] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, May 2015.

[4] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud.," *Nucleic acids research*, vol. 41, pp. W557–61, July 2013.

[5] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. De Roure, "myExperiment: a repository and social network for the sharing of bioinformatics workflows.," *Nucleic acids research*, vol. 38, pp. W677–82, July 2010.

[6] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pp. 423–424, IEEE, 2004.

[7] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.," *Genome biology*, vol. 11, p. R86, 2010.

[8] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, pp. 171–200, Jan. 2006.

[9] M. Caballer, I. Blanquer, G. Moltó, and C. de Alfonso, "Dynamic Management of Virtual Infrastructures," *Journal of Grid Computing*, Apr. 2014.

[10] S. M. S. da Cruz, M. Mattoso, V. Batista, A. M. R. Dávila, E. Silva, F. Tosta, C. Vilela, M. L. M. Campos, R. Cuadrat, and D. Tschoeke, "OrthoSearch," in *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08*, (New York, New York, USA), p. 1282, ACM Press, Mar. 2008.

[11] S. M. S. da Cruz, V. Batista, E. Silva, F. Tosta, C. Vilela, R. Cuadrat, D. Tschoeke, A. M. R. Dávila, M. L. M. Campos, and M. Mattoso, "Detecting distant homologies on protozoans metabolic pathways using scientific workflows.," *International journal of data mining and bioinformatics*, vol. 4, pp. 256–80, Jan. 2010.

[12] K. Katoh and D. M. Standley, "MAFFT multiple sequence alignment software version 7: improvements in performance and usability.," *Molecular biology and evolution*, vol. 30, pp. 772–80, Apr. 2013.

[13] R. D. Finn, J. Clements, and S. R. Eddy, "HMMER web server: interactive sequence similarity searching.," *Nucleic acids research*, vol. 39, pp. W29–37, July 2011.

[14] R. R. C. Cuadrat, S. M. da Serra Cruz, D. A. Tschoeke, E. Silva, F. Tosta, H. Jucá, R. Jardim, M. L. M. Campos, M. Mattoso, and A. M. R. Dávila, "An orthology-based analysis of pathogenic protozoa impacting global health: an improved comparative genomics approach with prokaryotes and model eukaryote orthologs.," *Omics : a journal of integrative biology*, vol. 18, pp. 524–38, Aug. 2014.

[15] S. Powell, K. Forslund, D. Szklarczyk, K. Trachana, A. Roth, J. Huerta-Cepas, T. Gabaldón, T. Rattei, C. Creevey, M. Kuhn, L. J. Jensen, C. von Mering, and P. Bork, "eggNOG v4.0: nested orthology inference across 3686 organisms.," *Nucleic acids research*, vol. 42, pp. D231–9, Jan. 2014.

An Environment-aware Anomaly Detection Framework of Cloud Platform for Improving Its Dependability

Guiping Wang[1], Shuyu Chen[2,3*], and Jun Liu[1]
[1]College of Computer Science, Chongqing University, Chongqing, CHINA
[2]School of Software Engineering, Chongqing University, Chongqing, CHINA
[3]Department of Electrical and Computer Engineering, McGill University, Montreal, CANADA
netmobilab@cqu.edu.cn

Abstract - *Virtualization technology is a core technology in Cloud Platform, which allows the hardware, the operating systems, and the applications running atop to be encapsulated into virtual machines (VMs). Along with the increasing scale and complexity of Cloud Platform, various faults cause the frequent downtime accidents of VMs, which has seriously lowered the dependability of Cloud Platform. Anomaly detection can detect anomalous status of VMs, while subsequent fault diagnosis can further discriminate the reasons of the detected anomalies. The former means is the foundation of the latter one. VMs are isolated one another in Cloud Platform. An anomalous VM usually does not affect other VMs. Aiming at detecting anomalous VMs in Cloud Platform, this paper proposes an environment-aware anomaly detection framework. 53 performance metrics of each VM are collected to characterize its current status. A series of processing steps are then conducted to judge whether the VMs in Cloud Platform are normal or abnormal. The experimental results show that the proposed framework can detect anomalous VMs in real time and with high accuracy rate, thus improving the dependability of Cloud Platform.*

Keywords: Cloud Platform; Virtual Machine (VM); Dependability; Anomaly Detection; Performance metrics.

1 Introduction

Cloud computing has the following essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service ([1]). Currently, it has become the mainstream of computing and service mode. Virtualization technology is a core technology in Cloud Platform, which allows the hardware, the operating systems, and the applications running atop to be encapsulated into virtual machines (VMs). Along with the increasing scale and complexity of Cloud Platform, various faults cause the frequent downtime accidents of VMs, which has seriously lowered the dependability ([2]) of Cloud Platform and restricted the development of cloud computing.

The frequent downtime accidents of VMs cause huge losses on not only Cloud service providers but also users. On Jan. 2, 2010, a Ruby-on-Rails application hosting company, Heroku, experienced the complete failure of their 22 VMs which were hosted on Amazon EC2 and ran 44,000 popular applications and development services. These specialized and high-capacity Amazon EC2 instances disappeared all of a sudden without any warning ([3]). On Jun 29, 2012, Amazon Web Services (AWS) broke down due to failed generators. A lengthy server reboot process exacerbated this outage. 7% of VM instances in Virginia availability region were affected by this incident ([4]). Fig. 1 illustrates the statistics of Amazon AWS downtime accidents from 2006 to 2013. The researchers from University of California at Berkeley summarize top 10 obstacles to the growth of Cloud, where the first one is availability (an important attribute of dependability) of Cloud service ([5]).

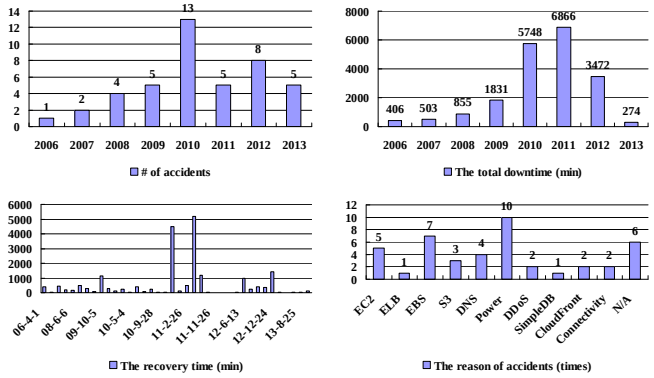


Fig. 1. The statistics of Amazon AWS downtime accidents from 2006 to 2013

VMs are the important carrier for Cloud services. They are isolated one another in Cloud Platform. An anomalous VM usually does not affect other VMs. Aiming at detecting anomalous VMs in Cloud Platform, this paper proposes an environment-aware anomaly detection framework, which first partitions all VMs into several monitoring domains according VMs' running environment attributes and then executes anomaly detection in each domain. This paper conducts experiments on the framework to verify its performance.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 presents the

definitions and preliminaries. Section 4 describes the proposed anomaly detection framework in detail. Section 5 introduces the collected performance metrics, and conducts experiments on the framework. Section 6 concludes this paper and looks into future work.

2 Related work

Since there is little research work in literature on understanding the dependability of Cloud environment, Guan et al. ([6]) first attempt to present a cloud dependability analysis (CDA) framework for characterizing system dependability in cloud computing. In order to analyze the correlation of various performance metrics with failure events in virtualized and non-virtualized environments, they design failure-metric DAGs (directed acyclic graph). By comparing the generated DAGs in these two environments, they gain insight into the impact of virtualization on the cloud dependability. Melo et al. ([7]) study high availability of Cloud environment from software rejuvenation point of view. They present a comprehensive availability model to evaluate the utilization of live migration mechanism to enable VMM rejuvenation with minimum service interruption.

Currently, anomaly detection under distributed environments (such as Cloud Platform) is a research focus in literature ([8]-[18]). During the operation of distributed systems, a large number of log data are produced in operating system, database, and applications, etc., which can reflect their operation state. Various detection data also can be collected by dedicated detecting tools or programs. These log data and detection data are referred to as *monitoring data* ([8]), which mainly record the operations of users on the observed system, reflect the performance or state of servers, physical hosts, storage systems, VMs, etc., or depict the behavior of the applications. Concretely, the main monitoring data adopted in literature to characterize the status of an observed distributed system include performance metric ([8]-[13]), system call sequence ([14][15]), network traffic ([16]), and system log ([17][18]).

Lan et al. ([9]) present an automated mechanism for node-level anomaly identification in large-scale systems. Health-related performance metric data (e.g., CPU utilization, available memory size, I/O, network traffic) are collected across the system for anomaly identification. Node grouping dynamically divides system resources into groups and the nodes in the same group are expected to exhibit similar behaviors. Data transformation, feature extraction, and outlier detection, are applied per group to find abnormal nodes (i.e., anomalies). The detected anomalies are manually be validated by system administrators. However, they do not clarify how to dynamically group nodes.

In order to improve dependability of Cloud platforms, Fu et al. ([10]) propose a hybrid adaptive anomaly detection framework using one-class and two-class support vector machines (SVM) detection models. However, once the framework switches from one model to the other, the model needs to be retrained, which costs much computing time.

Alarifi and Wolthusen ([14]) present a Hypervisor based anomaly detection system, which monitors system call sequences between a VM and its host kernel to detect abnormal VM's behaviors. The detection system uses hidden markov to construct a model about VM's normal behavior and obtains the classifier. The new system call sequences are then detected by the classifier. Xiong et al. ([16]) put forward two detection methods by analyzing dynamic characteristics of the network traffic in Cloud communications to detect anomalies.

System logs also can characterize system behaviors, and therefore be adopted in anomaly detection. Fu et al. ([17]) propose an unstructured log analysis method to detect anomaly in distributed systems. They first convert text messages in log files to log keys, and then learn a Finite State Automaton (FSA) to present the normal work flow for each system component. A performance measurement model is obtained to characterize the normal execution performance. New log files are then input to the model to detect anomalies.

The main problem in log or system call sequence based anomaly detection is that, under large-scale and high dynamic distributed environments (i.e., Cloud Platform), it is hard to construct a stable model to characterize system's normal behavior.

Chandola et al. ([19]) conduct a comprehensive survey on the researches in literature about anomaly detection. They classify existing techniques in anomaly detection into six different categories based on the underlying approaches: classification-based, nearest neighbor-based, clustering-based, statistical, information theoretic, spectral. They summarize the advantages and disadvantages of the techniques in each category. Using 121 UCI data sets, Delgado et al. ([20]) evaluate 179 classifiers from 17 families. Their research results show that random forest (RF) and support vector machine (SVM) are the best two classifiers.

3 Definitions and Preliminaries

This section first clarifies a set of general concepts related to dependability, and introduces new meanings into dependability. Then it describes the detection principles, and introduces preliminaries of the proposed framework.

3.1 Definitions Related to Dependability

Definition 1 (**Dependability**): The *dependability* of a system refers to the comprehensive abilities to deliver service that can justifiably be trusted ([2]).

Definition 2 (**Attributes of Dependability**):

Dependability is composed of five attributes: *reliability*, *availability*, *safety*, *integrity*, and *maintainability* ([2]).

Reliability refers to continuity of correct service, which can be measured by *mean time to failures* (MTTF). *Maintainability* refers to ability to undergo modifications and repairs, which can be measured by *mean time to repair* (MTTR). *Availability* refers to readiness for correct service, which can be measured by the ratio of MTTF to *mean time between failures* (MTBF), where MTBF = MTTR + MTTF. *Safety* refers to absence of catastrophic consequences on the user(s) and the environment. *Integrity* refers to absence of improper system alterations.

Definition 3 (**Threats to Dependability**): The threats to dependability include *failures*, *errors*, and *faults*. A *service failure* (or *failure* for short) is an event that occurs when the delivered service deviates from correct service. A service failure means that at least one (or more) external state of the system deviates from the correct service state. The deviation is called an *error*. The adjudged or hypothesized cause of an error is called a *fault* ([2]).

Despite the researches of anomaly detection in statistics can be traced back to the end of the 19th century ([21]), the researches of anomaly detection in distributed environments spring up until about 2010 ([8]-[18]). The underlying reason is that the past researches mainly focused on failure (e.g., downtime, outage) detection under distributed environment; however, along with the increasing scale and complexity of distributed systems, it may appear some anomalies before real failures occur. These anomalies include system performance degradation, system workload or performance fluctuation, and system response time slowing down.

The detecting system can detect the anomalies in real time, and remind human operators before failures. The detecting results can facilitate human operators adopting relevant measures and reducing the adverse effects of faults in a distributed system, thus improving its dependability. Therefore, this paper introduces new meanings, i.e., anomaly and anomaly detection, into dependability.

Definition 4 (**Anomaly**): From dependability point of view, an *anomaly* refers to the status of a detected system (or a node in the system) that deviates from the expected normal status, or deviates from the status of most of the time (or most of other nodes in the system).

Anomalies can be caused by faults, including hardware / software, malicious / non-malicious, and deliberate / non-deliberate faults ([2]). They can also be caused by non-fault factors, including normal workload fluctuation, the increasing of concurrent access, and changed environment. Anomaly detection can detect anomalous status of an observed system, while subsequent fault diagnosis can further discriminate the reasons of the detected anomalies. The former means is the foundation of the latter one. This paper only focuses on the former means, i.e., anomaly detection.

Definition 5 (**Anomaly Detection**): Anomaly Detection is a function of detecting the anomalous status of a detected system or anomalous nodes in a detected system.

The proposed framework can timely detect anomalous VMs and report them to human operators before failures, thus prolonging MTTF and improving the *reliability* of Cloud Platform. Anomaly detection and subsequent fault diagnosis can diagnose the types of faults and trace the sources of faults in real time before these faults cause catastrophic consequences, thus improving the *safety* of Cloud Platform. Meanwhile, these two means can reduce the difficulty level of recovery and reduce MTTR, thus improving the *maintainability* of Cloud Platform. Through the above improvements, these two means can improve the percentage time of providing correct service, thus improving the *availability* of Cloud Platform. Through improving reliability, safety, maintainability, and availability, the proposed framework can ultimately improve the Cloud Platform's comprehensive abilities (i.e., dependability) to deliver service that can justifiably be trusted.

3.2 Detection Principles & Preliminaries

In Cloud environment, the performance of VMs can be characterized by performance metrics, while the value of a performance metric is also affected by the factors including resource configuration of the VM, the workload of the VM, and the resource configuration of the underlying physical host. A same VM will exhibit different performance under different running environment. The detecting system should exclude the performance deviation between VMs caused by different running environment. Otherwise, it will result in a large number of false positives.

Therefore, in order to improve the detection accuracy, VM's running environment attributes and performance metrics are collected at the same time. The proposed framework adopts environment-aware detection. To be specific, the framework divides a large number of VMs in Cloud Platform into several monitoring domains based on VMs' running environment attributes, which makes VMs in a same monitoring domain have similar running environment. In each domain, the anomaly detection algorithms detect anomalous VMs based on their performance metrics.

Based on the above detection principles, two important definitions are given below.

Definition 6 (**VM's Running Environment Attribute Set**): This attribute set includes resource configuration of a VM, the workload of the VM, and the resource configuration of the underlying physical host. The attribute set can be formalized by the following vector:

$$\boldsymbol{R} = [R_1 \ R_2 \ \dots \ R_r]^T, \quad (1)$$

where R_i represents an environment attribute, r is the number of attributes.

Definition 7 (**VM's Performance Metric Set**): This set includes a set of metrics. Each metric is an individually measurable variable, which characterizes the performance or status of a VM from a certain point of view. The metric set can be formalized by the following vector:

$$\boldsymbol{X} = [X_1 \ X_2 \ \dots \ X_n]^T, \quad (2)$$

where X_i represents a performance metric, n is the number of metrics.

4 The Proposed Anomaly Detection Framework

Fig. 2 illustrates the proposed anomaly detection framework for Cloud Platform. The framework is composed of several modules, including *Partition* & *Deployment*, *Collection* & *Transmission*, *Data Processing*, *Environment-aware Detection*, and *Candidate VMs Detection*. Each module contains several function modules. The key modules are described as follows.

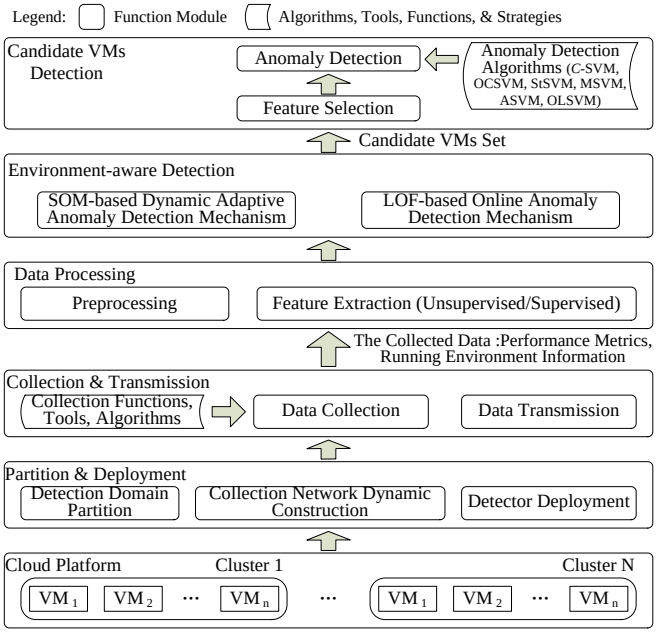


Fig. 2. The anomaly detection framework for Cloud Platform

Partition & *Deployment* module is responsible for partitioning the VMs into several monitoring domains according to VMs' running environment attribute set. Concretely, all VMs are clustered into several clusters based on their running environment attribute set. Each obtained cluster is a domain.

Collection & *Transmission* module is responsible for collecting the performance metric data and running environment attribute data of all VMs and transmitting to the upper module.

Data Processing module is responsible for indispensable processing on collected data before anomaly detection, including preprocessing (zero-mean, decorrelation, standardization, etc.) and feature extraction.

Environment-aware Detection module is responsible for preliminarily detecting anomalous VMs and submitting the set of candidate VMs to the upper module for further detection. The detection adopts SOM (Self Organizing Map) based or LOF (Local Outlier Factor) based anomaly detection mechanism (optionally).

Candidate VMs Detection module is responsible for further detecting anomalous VMs using more precise and powerful detection algorithms (i.e., SVM-based algorithms). The research of Delgado et al. ([20]) shows that SVM is one of the best classifiers. Moreover, the SVM-based algorithms can solve the challenges of anomaly detection under Cloud environment, such as multiple categories of anomalies, imbalance in datasets, online learning. Before anomaly detection, feature selection is executed on the performance metric data to reduce dimensionality and reserve some original performance metrics for future anomaly localization and fault diagnosis.

5 Experiments

5.1 The collected performance metrics

The collected performance metrics can be classified into two categories: *host performance metrics* and *network performance metrics*. Further, the metrics in the former category can be classified into four sub-categories: *computation*, *storage*, *disk I/O*, *process*. Totally, 53 performance metrics are collected, which are listed as follows.

1) *Computation resource performance metrics*: the 11 metrics listed in Table 1 reflect the current computation resource utilization in a VM and the underlying physical host.

2) *Storage resource performance metrics*: the 12 metrics listed in Table 2 reflect the current storage resource utilization in a VM and the underlying physical host.

3) *Disk I/O performance metrics*: the 9 metrics listed in Table 3 reflect the current disk I/O performance in a VM and the underlying physical host.

4) *Process performance metrics*: the 6 metrics listed in Table 4 reflect the current process performance in a VM and the underlying physical host.

5) *Network performance metrics*: the 15 metrics listed in Table 5 reflect the current network performance in a VM and the underlying physical host.

Table 1. The collected computation resource performance metrics

Metrics	Description
cpu_idle	Percentage of CPU idle time.
cpu_user	Percentage of CPU utilization that occurred while executing at the user level.
cpu_system	Percentage of CPU utilization that occurred while executing at the system level.
cpu_nice	Percentage of CPU utilization that occurred while executing at the user level with nice priority.
cpu_iowait	Percentage of CPU time waiting I/O operations.
cpu_irq	Percentage of CPU time spent to service hardware interrupts.
cpu_softirq	Percentage of CPU time spent to service software interrupts.
cpu_cs	Percentage of CPU time for process switch.
cpu_running	Number of running tasks in queues.
vcpu_run	Runtime of VCPU
vcpu_run_rate	Percentage of VCPU utilization

Table 2. The collected storage resource performance metrics

Metrics	Description
mem_swapd	Amount of occupied swap space.
mem_free	Amount of available memory.
mem_buf	Buffer size of block device.
mem_cache	Buffer size of character device.
mem_si	Amount of virtual memory per second read from physical disk.
mem_so	Amount of virtual memory per second written to physical disk.
mem_total	Total amount of physical memory.
mem_slab	Amount of memory for kernel.
vmem_cur	Amount of virtual memory in VM.
vmem_rate	Utilization rate of virtual memory in VM.
vmem_max	Maximum amount of occupied virtual memory in VM.
vmem_max_rate	Maximum utilization rate of virtual memory in VM.

Table 3. The collected disk I/O performance metrics

Metrics	Description
disk_await	The average wait time (in milliseconds) for I/O requests issued to the device to be served.
disk_svctm	The average service time (in milliseconds) for I/O requests issued to the device to be served.
disk_util	Percentage of I/O time per second.
disk_r	Number of operations of reading I/O devices per second.
disk_w	Number of operations of writing I/O devices per second.
disk_queue	Average length of I/O queues.
disk_rq	Average amount of data in an I/O operation.
vbd_rd	Number of operations of reading virtual block devices per second.
vbd_wr	Number of operations of writing virtual block devices per second.

Table 4. The collected process performance metrics

Metrics	Description
pro_size	Total amount of memory occupied by processes.
pro_share	Total amount of share memory occupied by processes.
pro_cpu	Percentage of CPU time occupied by processes.
pro_mem	Percentage of memory occupied by processes.
pro_time	Total CPU time occupied by processes.
pro_thread_count	Number of threads.

Table 5. The collected network performance metrics

Metrics	Description
net_rx_byte	Network data received per second.
net_tx_byte	Network data transmitted per second.
net_rx_packet	Number of received packets per second.
net_tx_packet	Number of transmitted packets per second.
net_rx_loss	Number of lost packets in receiving per second.
net_rx_loss	Number of lost packets in transmitting per second.
icmp_rx_ packet	Number of received ICMP packets per second.
icmp_tx_packet	Number of transmitted ICMP packets per second.
icmp_rx_loss	Number of unreachable packets in received ICMP packets per second.
icmp_tx_loss	Number of unreachable packets in transmitted ICMP packets per second.
net_load	Network load rate.
vnet_rx	Virtual network data received per second.
vnet_tx	Virtual network data transmitted per second.
vnet_rx_rate	Rate of virtual network data received per second.
vnet_tx_rate	Rate of virtual network data transmitted per second.

5.2 Experiments on performance metrics

In order to test the probability distribution of the collected performance metrics, all performance metrics of a random selected VM and the underlying physical host are sampled every 5 seconds lasting 2 hours. Therefore, 1440 sampled values are obtained.

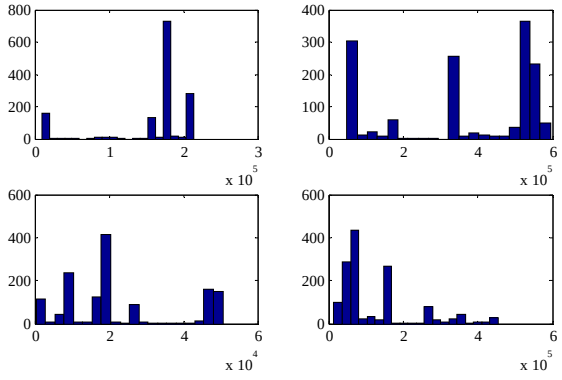


Fig. 5. The histograms of four performance metrics.

In Fig. 5, four performance metrics are randomly chosen, and their histograms are plotted. From this figure, it is concluded that typically the performance metrics do not follow the Gaussian distribution. Therefore, the feature extraction and anomaly detection algorithms based on the Gaussian distribution assumption are not considered in this paper.

5.3 Anomaly Injection

In order to evaluate the performance of the proposed framework, this paper constructs a Cloud Platform testbed, which is composed of several physical hosts. 0-4 VMs are deployed on each host.

In addition, to make the sample set include abnormal samples excluding normal ones, four types of anomalies are injected to randomly selected VMs in the Cloud Platform, which are listed as follows.

1) Anomaly in computation resource consumption: A computing-intensive program runs in a VM, and persistently over-consumes computation resources.

2) Anomaly in memory resource consumption: A running program in a VM continues to apply for dynamic memory through malloc() function without releasing the assigned memory, which causes memory leak and over-consumption of the VM's memory resources.

3) Anomaly in disk I/O operation: A running program in a VM persistently reads large files on disk, generating large amounts of disk I/O operations to simulate anomalous disk I/O.

4) Anomaly in network access: Several users run LoadRunner from their own physical personal computer to simultaneously access a Web application server deployed on a VM, generating a large number of HTTP connections to simulate anomalous network behavior.

During the experiments, these anomalies are injected solely (single-anomaly) or in the form of combination (combination-anomaly).

5.4 Experiments on the anomaly detection framework

This paper adopts the following two accuracy measures to evaluate the performance of the designed anomaly detection framework. (F_P, False Positive; F_N, False Negative; T_P, True Positive; T_N, True Negative)

1) *Sensitivity*: the proportion of True Positive (i.e., correctly detected anomalous VMs) to the number of actual anomalous VMs.

$$Sensitivity = \frac{T_P}{T_P + F_N} \tag{3}$$

2) *Specificity*: the proportion of True Negative (i.e., correctly detected normal VMs) to the number of actual normal VMs.

$$Specificity = \frac{T_N}{F_P + T_N} \tag{4}$$

The experimental results of four single-anomaly tests are listed in Table 6. (Each experiment is conducted 10 times, and the results are averaged. The results of all experiment are also averaged in the last row.)

Table 6. Performance measures of four single-anomaly tests

Anomaly	Sensitivity	Specificity
Computation	1.00	0.96
Memory	1.00	0.97
Disk I/O	1.00	0.95
Network	0.97	0.92
Average	0.99	0.95

This paper also conducts experiments on the combination of two types of anomalies. The experimental results of six combination-anomaly tests are listed in Table 7. The last row is the average result of all experiments.

Table 7. Performance measures of six combination-anomaly tests

Anomaly	Sensitivity	Specificity
Computation & Memeory	1.00	0.96
Computation & Disk I/O	1.00	0.93
Computation & Network	0.98	0.93
Memeory & Disk I/O	1.00	0.95
Memeory & Network	0.97	0.93
Disk I/O & Network	0.97	0.94
Average	0.987	0.940

From the above experimental results, it is concluded that the accuracy of the designed anomaly detection framework meets the practical requirements under Cloud environment. Averagely, the detection sensitivities are 0.99 and 0.987 under single-anomaly and combination-anomaly respectively, and the detection specificity are 0.95 and 0.94 respectively.

The real-time performance of the designed framework also meets the practical requirements. In the experiments including single-anomaly and combination-anomaly tests, the designed framework takes less than 0.35 second to produce a list of anomalous VMs.

6 Conclusions and Future Work

This paper proposes an anomaly detection framework for detecting anomalous VMs under Cloud environment. The main prominent feature of the framework is environment-aware detection, which makes the framework achieve high

detection accuracy. The future work of this paper will focus on anomaly localization and fault diagnosis. These subsequent means can further discriminate the reasons (fault or non-faults) of the detected anomalies and trace the sources of faults. The ultimate goal of these means is to improve the dependability of Cloud Platform.

7 Acknowledges

The work of this paper is supported by National Natural Science Foundation of China (Grant No. 61272399).

8 References

[1] The NIST Definition of Cloud Computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[2] A. Avizienis, J. C. Laprie, B. Randell, et al. Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, 2004.

[3] C. Books, Heroku learns from Amazon EC2 outage, http://searchcloudcomputing.techtarget.com/news/1378426/Heroku-learns-from-Amazon-EC2-outage, Jan. 8, 2010.

[4] J. Saarinen, Amazon explains recent outages, http://www.crn.com.au/News/307746,amazon-explains-recent-outages.aspx, July 6, 2012.

[5] M. Armbrust, A. Fox, R. Griffith, et al. A view of cloud computing. Communications of the ACM, vol. 53, no. 4, pp. 50-58, 2010.

[6] Q. Guan, C. C. Chiu, and S. Fu. CDA: A cloud dependability analysis framework for characterizing system dependability in cloud computing. Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 11-20, 2012.

[7] M. Melo, P. Maciel, J. Araujo, et al. Availability study on cloud computing environments: Live migration as a rejuvenation mechanism, Proceedings of IEEE/IFIP 43rd International Conference on Dependable Systems & Networks (DSN), 2013.

[8] H. S. Pannu, J. G. Liu, Q. Guan, et al. AFD: Adaptive Failure Detection System for Cloud Computing Infrastructures, Proceedings of 31st IEEE International Performance Computing and Communications Conference (IPCCC), pp. 71-80, 2012.

[9] Z. L. Lan, Z. M. Zheng, and Y. W. Li. Toward automated anomaly identification in large-scale systems. IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 2, pp. 174-187, 2010.

[10] S. Fu, J. G. Liu, and H. Pannu. A hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines, Proceedings of 8th International Conference on Advanced Data Mining and Applications (ADMA), pp. 726-738, 2012.

[11] K. Bhaduri, K. Das, and B. L. Matthews. Detecting Abnormal Machine Characteristics in Cloud Infrastructures, Proceedings of 11th IEEE International Conference on Data Mining Workshops, pp. 137-144, 2011.

[12] H. Nguyen, Z. M. Shen, Y. M. Tan, et al. FChain: Toward black-box online fault localization for cloud systems, Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems, pp. 21-30, 2013.

[13] G. P. Wang, S. Y. Chen, Z. Zhou, and M. W. Lin. A Dependable Monitoring Mechanism Combining Static and Dynamic Anomaly Detection for Network Systems, International Journal of Future Generation Communication and Networking, vol. 7, no. 1, pp. 1-18, 2014.

[14] S. Alarifi and S. Wolthusen. Anomaly detection for ephemeral cloud IaaS virtual machines, Proceedings of 7th International Conference on Network and System (NSS), pp. 321-335, 2013.

[15] A. Patel, M. Taghavi, K. Bakhtiyari, et al. An intrusion detection and prevention system in cloud computing: A systematic review, Journal of Network and Computer Applications, vol. 36, no. 1, pp. 25-41, 2013.

[16] W. Xiong, H. P. Hu, N. X. Xiong, et al. Anomaly secure detection methods by analyzing dynamic characteristics of the network traffic in cloud communication, Information Sciences, vol. 258, pp. 403-415, 2014.

[17] Q. Fu, J. G. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis, Proceedings of IEEE International Conference on Data Mining, pp. 149-158, 2009.

[18] H. B. Mi, H. M. Wang, Y. F. Zhou, et al. Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems, IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 6, pp. 1245-1255, 2013.

[19] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. ACM Computing Surveys, vol. 41, no. 3, Article 15, 2009.

[20] M. F. Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? Journal of Machine Learning Research, vol. 15, pp. 3133-3181, 2014.

[21] F. Y. Edgeworth. On discordant observations, Philosophical Magazine, vol. 23, no. 5, pp. 364-375, 1887.

SESSION

BIG DATA ANALYTICS, DATA WAREHOUSES, AND RELATED METHODS AND SYSTEMS

Chair(s)

TBA

Hadoop Scalability and Performance Testing in Heterogeneous Clusters

Fernando G. Tinetti[1], **Ignacio Real**[2], **Rodrigo Jaramillo**[2], **and Damián Barry**[2]
[1]III-LIDI, Facultad de Informática, UNLP,
Comisión de Inv. Científicas de la Prov. de Bs. As.
La Plata 1900, Argentina
[2]LINVI, Departamento de Informática, Facultad de Ingeniería, UNPSJB,
Puerto Madryn 9120, Argentina

Abstract— *This paper aims to evaluate cluster configurations using Hadoop in order to check parallelization performance and scalability in information retrieval. This evaluation will establish the necessary capabilities that should be taken into account specifically on a Distributed File System (HDFS: Hadoop Distributed File System), from the perspective of storage and indexing techniques, and queriy distribution, parallelization, scalability, and performance in heterogeneous environments. The software architecture will be designed and evaluated as either centralized or distributed, and the relevant experiments will be carried out establishing the performance improvement for each architecture.*

Keywords: Big Data, Information Retrieval, HDFS, MapReduce, Cluster, Parallelization, Scalability, Performance

1. Introduction

The amount of information is continuously growing: social networking, content management systems (CMS) and portals in general and as collaboration platforms in particular, data within organizations generated either by production systems or by digitizing existing information. Data usually measured in gigabytes a few years ago is now measured un terabytes and petabytes [5] [6]. Data as well as relevant applications typically require more resources than those available on a single computer. The challenge is therefore to produce and handle computing infrastructure that allows to take advantage (harnessing) of existing computing platforms, usually heterogeneous. Thus, several computers wroking collaboratively, would reach availability and scalability to cope with the currently needed information processing [7] [8]. Reusing low-cost equipment allows to address the aforementioned problem, and requires techniques of distributed systems, where each computing system has local storage and computation facilities so that processing and access can be distributed and balanced in a heterogeneous cluster [9]. A set of desirable properties for an information sharing and data reocovering system in a heterogeneous and scalable environment a could be defined [7] [10] [11]: high performance, fault tolerance and heterogeneous computing. Moreover, the NoSQL solutions for managing large volumes of data are typically based on the usage of a heterogeneous system.

There are several techniques for configuring heterogeneous computing environments. We have concentrated our work in the framework programmed in Java called Hadoop to store and process large amounts of data in clusters [1] [2]. HDFS besides being a distributed file system, scalable and portable, solves availability and reliability issues by replicating data in multiple computers [9].

1.1 Hypothesis

The amount of data that humans are capable of generating and storing hinders the analysis and information processing in general. Processing/analysis in this field is commonly referred to as *Big Data* applications [3]. Several problems are involved, two of the most complex ones could be *reduced* to the following questions:

- 1. How to store and protect the large volume of available data?
- 2. How to process and evaluate data in an acceptable period of time?

Specifically with regard to the latter question, the hypothesis from wich we work is the existence of a performance and scalability characterization of each heterogeneous cluster. This characterization is given in the context of the different techniques for handling large volumes of data while varying the number of nodes that comprise it.

1.2 Contribution

At least, we check a real usage of the infrastructure, Hadoop, proposing a design that facilitates scalability. This design could be especially used by organizations and agencies that need to handle large volumes of information, as in national or local governments or private sector companies. Another significant contribution lies in the utility that provide this type of architecture in the field of research and development.

1.3 Goals

We have defined a set of objectives guiding the work reported in this paper:

1) Design different scalable architectures for a Hadoop cluster varying the number of nodes in order to analyze processing time.
2) Select bibliographic material and generate a knowledge based on the techniques and methods used in partitioning, replication, and distribution in the Apache Hadoop infrastructure.
3) Set parameters and evaluate different architectures for optimizing Hadoop configuration.

2. Hadoop

Hadoop is a framework that allows to build a cluster architecture, providing parallel recovery of information and replication. Also, Hadoop implement a simple way to add and/or drop cluster nodes, improving scalability, minimizing the likelihood of failure nodes containing distributed data. Developed in the Java programming language, by the community of free software Apache, the Hadoop architecture is composed of three main components:

- The Hadoop Distributed File System*HDFS*, using a master/slave architecture, as shown in Fig. 1.
- The MapReduce framework, which allows the programmer to split and parallelize complex calculations in any number of computers.
- The Hadoop Common, a set of tools for integrating Hadoop subprojects.

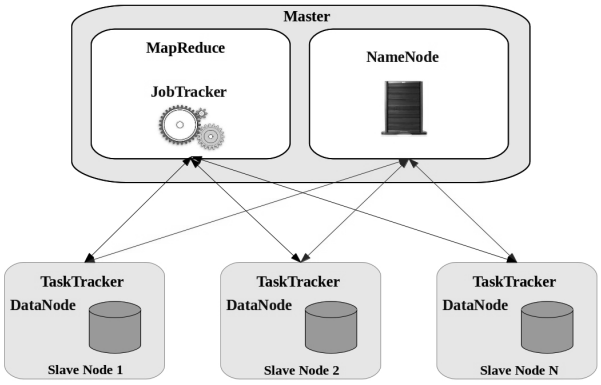


Fig. 1: Hadoop Architecture

The two main components, the HDFS and MapReduce, define a stable, robust, and flexible framework for distributed applications, making it possible to work with multiple nodes and process large amounts of information.

The HDFS is designed to provide high performance and data reliability on heterogeneous hardware. MapReduce allows the development of parallel processing applications, focused on scalability. Queries on distributed data could be distributed as well, thus enhancing performance via parallel/distributed processing. Both (HDFS-MapReduce) are designed to analyze large amounts of structured and unstructured data.

2.1 Hadoop DFS

HDFS implements a master/slave architecture as shown in Fig. 2, where: a) NameNode is the master process, b) DataNodes are the slave processes, and c) the master process is replicated in a Secondary NameNode. The HDFS keeps separately metadata (in the NameNode) and data (in the DataNodes). System (data) reliability is achieved by replicating files in multiple DataNodes, which also allows faster transfer rates and access. All files stored in HDFS system are divided into blocks, whose size usually is between 64 MB and 128 MB. A Hadoop cluster can consist of thou-

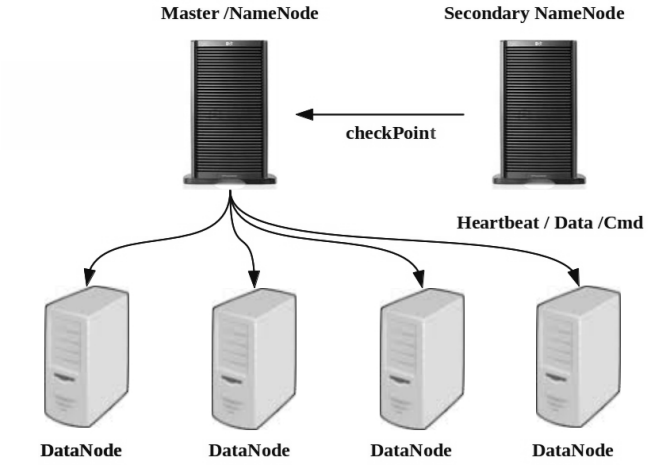


Fig. 2: HDFS Architecture

sands DataNodes wich respond to different read and write requests from clients, also maintaining block replication. The DataNodes regularly send information to the NameNode of its blocks to validate consistency with other DataNodes. A cluster may consist of thousands of DataNodes, each of which stores a portion of (possibly replicated) data. Each of the DataNodes is likely to fail. The HDFS provides rapid and automatic failover recovery via replication and the metadata contained in the NameNode.

During normal operation the DataNode sends signals (heartbeats) to NameNode every three seconds by default. If the NameNode does not receive a defined number of the heartbeats from a DataNode, the DataNode is considered out of service. Then the NameNode triggers the creation of new replicas of the DataNode out of service in other (not failing) DataNodes. Heartbeats also contain information about the total storage capacity, the fraction of storage that is in use, and the number of files or data transfer in progress.

2.2 Hadoop MapReduce

MapReduce allows Hadoop the parallel processing on large volumes of data through multiple nodes in a cluster, the data to be processed may be stored in the HDFS. The execution of a MapReduce process usually divides the input data into a set of independent chunks of information that are processed by the Map tasks in parallel. Then, the results of the map tasks are classified and will be the input to Reduce tasks. Typically both the input data and output data are stored in the file system.

MapReduce is based on the Master/Slave architecture, similar to that of the HDFS, as shown in Fig. 3. The Master runs the so called JobTracker and slaves run the TaskTrackers. The JobTracker is responsible for the management and control of all sumitted jobs. Also, it is responsible for task distribution and management of available TaskTrackers, trying to keep the job as close to the data as possible. The JobTracker takes into account the machines (nodes) that are close to and/or contain the data needed.

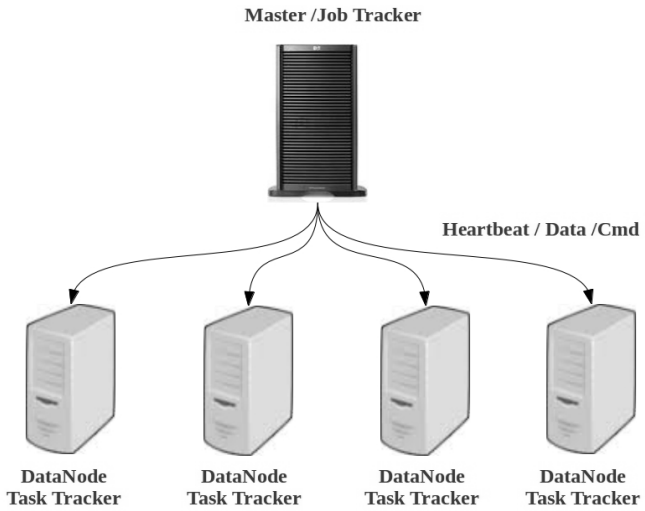


Fig. 3: MapReduce Architecture

MapReduce is based on key/value pairs, which are processed in (are the input to) Map tasks. Every Map task returns a list of pairs in a different domain data. All the pairs (generated by the Map tasks) with the same key are grouped and processed by a Reduce task.

MapReduce handles fault tolerance in a similar way to that described for the HDFS: each TaskTracker process reports regularly its status to the JobTracker process. If over a period of time the JobTracker process has not received any report from a TaskTracker process, the TaskTracker process is considered as not running. In case of failure, the task is reassigned to a different TaskTraker process.

3. Design and Implementation of Experiments

Different cluster configurations were evaluated from the point of view of scalability and (*raw*) performance. We also used two benchmarks, each used for measuring different performance metrics. Hardware and benchmarks are described in the following subsections.

3.1 Computers-Hardware

We specifically focused our work on heterogeneous computing cluster configurations, using the computers detailed below:

1) Name: **Master**
   Processor: Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
   Memory: 10 GB
   SATA Disk: 500 GB

2) Name: **Slave1**
   Processor: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
   Memory: 16 GB
   SATA Disk: 500 GB

3) Name: **Slave2**
   Processor: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
   Memory: 8 GB
   SDisk: 1 TB

4) Name: **Slave3**
   Processor: Intel(R) Core(TM) i3 CPU 540 @ 3.07GHz
   Memory: 8 GB
   SATA Disk: 1TB

The computers were used for different testing scenarios: a centralized one and three cluster-like installations. Initially, the Master was used as a standalone centralized Hadoop installation, including a Master and a DataNode. We will use this installation as the departure point for testing the Hadoop software as well as measuring a non-distributed environment. The different cluster installations (from *1* to 4 computers) were made up just taking advantage of the previous installation by adding one more computer including one more DataNode. The same number of Map and Reduce tasks per node is maintained in all the experiments.

The Hadoop client process (which is not part of the main Hadoop infrastucture shown in Fig. 1 before) in every experiment was run on

- Name: **Client**
  Processor:AMD Turion(tm) X2 Dual-Core Mobile
  Memory: 4 GB

SATA Disk: 320 GB

3.2 Benchmarks-Software

We used two well-known tests provided by the Hadoop software: **TestDFSIO** and **TeraSort**. TestDFSIO is aimed at assessing the performance of the cluster/Hadoop installation and TeraSort is focused on scalability and parallelization.

The Hadoop **TestDFSIO** benchmark is used for reading and writing files in the HDFS, indicating number and size of files. TestDFSIO provides timing information, performance, and the average I/O speed. Basically, TerstDFSIO is useful for:

- Measurement tasks such as stress tests on HDFS.
- Discovering bottlenecks in the network.
- Evaluating the hardware performance.
- Checking the operating system configuration and Hadoop cluster machines in general.

In short, TestDFSIO gives a first impression of how fast the cluster works in terms of I/O. This test runs MapReduce jobs it is a MapReduce program that reads/writes random data from large files. Each Map task performs the same operation in a separate file and informs speed to a Reduce task, which is programmed to collect and summarize all measurements, given in MB/seg.

The Hadoop **Terasort** benchmark is designed to assess the performance and scalability of a Hadoop installation. It is specifically designed to check the distrubution of processes in the cluster using Map Reduce. TeraSort execution actually involves the execution of three MapReduce programs:

TeraGen for data generation

TeraSort for sorting the generated data

TeraValidate for sorted data validation

The TeraGen program writes data to disk just like testDFSIO-write creates random data. TeraSort sorting performance is based on the way that divides data between mappers/reducers and how data is collected and written by the partitioner. A partitioner is implemented for achieving a balanced workload. The partitioner uses an ordered list of N-1 sample keys that define the range of keys for each Reduce. In particular, a key is sent to the i-th Reduce if it resides within a range such that sample[i-1] $<=$ key $<$ sample[i], this ensures that the ith Reduce output is less than the output of the (i+1)-th Reduce. The TeraValidate program ensures that the output is globally sorted out by controlling (in the output data) that each key is less than or equal to the previous one.

4. Results

A TestDFSIO preliminary test was carried out for assesing Hadoop I/O performance of different cluster configurations. This test was run increasing from 1 to 14 the number of files of size 1 GB each (i.e. from 1 to 14 GB). Cluster architecture was also increased from 1 to 4 nodes, as shown in Fig. 4. Read operation results were taken into account for this run, with default settings, which imply

- BufferSize: 1000000 bytes
- Replication factor: 3
- Number of tasks in parallel by node: 2
- Block size: 64 MB

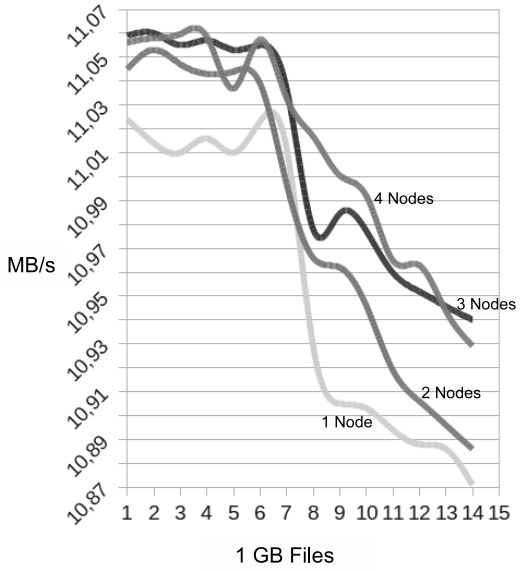


Fig. 4: TestDFSIO Read Performance

Fig. 4 shows how the performance decreases as the number of files (and involved data) increases, due to several factors among wich we can mention network traffic, local disk accesses, etc. Results are labelled as 1...4 Nodes from the standalone case to the complete cluster, with 4 nodes (computers) running the experiment. It is worth mentioning that heterogeneity does not play an important role and is almost negligible. However, when using the 4 computers (4 Nodes), performance is slightly penalized as compared to the case in which only 3 computers (3 Nodes) are used for the 14 files.

TeraSort results are the ones we are really interested in, because parallel computing is directly involved. Data to be sorted has to be generated, and we chose to follow the Fibonacci sequence $\times 10^7$. Therefore:

- In the first run 1×10^7 records or rows are generated, where each row is 100 bytes in size.
- 2×10^7, 3×10^7, 5×10^7, 8×10^7 and 13×10^7 records are then generated.

i.e. from 10 to 130 millions of records to be sorted. And given that each record is 100 bytes long, the total amount of data is among 1 GB to 13 GB. For each cluster configuration (from a standalone Master to the complete 4 nodes cluster), TeraGen was first executed to generate the data serie. Once

the data is generated, TeraSort is run in set of 10 identical experiments and the average runtime is taken as the result, just to avoid transient experiment "noise". Finally TeraValidate was run to confirm that the data were actually sorted. For each test configuration TeraGen was first executed to generate the first data series, and TeraSort was run in a sequence of 10 repetitions, thereby strengthen the statistical results and obtain representative values. TeraValidate was always used to confirm that the data were actually sorted. Table 1 shows measured runtime for each experiment, i.e. varying the number of computers mand the amount of data

Table 1: Summary of TeraSort Results

	1 Node	2 Nodes	3 Nodes	4 Nodes
1 GB	71	80	68	81
2 GB	114	148	113	102
3 GB	247	210	169	163
5 GB	577	373	258	210
8 GB	1042	885	504	316
13 GB	2386	1888	1098	574

to be sorted, where "1 Node" represents the standalone configuration (only the Master node is running), "2 Nodes" represents the configuration with the master and Slave1 running, and son on. There are several interesting details which can be quantified with the values shown in Table 1:

- A larger amount of data to be sorted implies increasing the runtime, as could be expected *a priori*.
- For small size data sets (e.g. 1 GB or 2 GB) using more computers does not imply a performance improvement. More specifically, the runtime using the Master and Slave1 increases the runtime for 1 GB and 2 GB data to be sorted.
- For large size data sets (e.g. 8 GB or 13 GB) using more computers always imply a performance improvement. The improvement depends on several factors such as centralized to distributed (1 Node and 2 Nodes), or where the added computer is relatively less powerfull than those already running in the cluster (3 Nodes and 4 Nodes, the 4th node is the least powerfull one in the cluster).
- For intermediate size data sets of those experimented with (e.g. 3 GB and 5 GB) gains are difficult to evaluate, and some more specific experiments should be carried out even with other benchmark/s.
- Given that Hadoop is expected to handle TB of information, all of the results could be considered highly encouraging, since even handling small size data sets it is possible to obtain performance gains using heterogenous computers.
- Some results regarding performance enhancements are linearly related, while others not. Relative computing power of each node has not been calculated, so the values cannot be strictically analyzed/evaluated from a numeric point of view. We have to continue our experiments (at least) in that line of work.

Fig. 5 shows the values of Table 1 graphically, focused on experiments runtime (on the vertical axis) depending on the amount of data to be sorted (on the horizontal axis). Some scalability details can be identified in Fig. 5, since

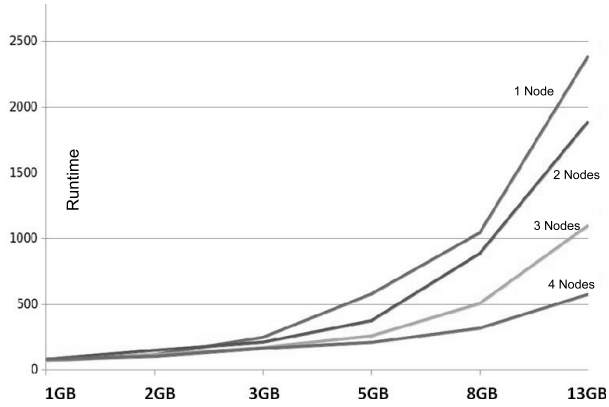


Fig. 5: TeraSort Scalability

scaling (increasing) data clearly implies increased runtime. At this time, it should be recalled that sorting is an $O(n^2)$ problem in general. Also, Fig. 5 clearly shows that increasing the number of nodes in a cluster the runtime is reduced using Hadoop with the proper configuration of HDFS and MapReduce.

5. Conclusions

Even when we have several problems in the Hadoop installation and configuration stages (mainly due to lack of documentation at the time we began this research some years ago) we have set a successful environment for experimentation with Hadoop in heterogeneous clusters. The installation and configuration could be replicated in every cluster (either homogeneous or heterogenous).

We have used TestDFSIO as a departure point: Hadoop I/O performance. Furthermore, we have found that TestDFSIO does not provide any information about parallel computing and/or scalability. At most, TestDFSIO could be used for node failure experimentation, varying the replication factor and injecting node failures in different scenarios.

We have used TeraSort for performance analysis in general, and performance scalability in particular. At this point, the MapReduce programming model and its conceptual basis are the most relevant. We have analyzed the processes involved in a Hadoop job so that we were able to determine the correct amount of Map and Reduce tasks per node and to properly configure MapReduce parameters. Our experiments show that processing times decreases as the cluster nodes are added. Clearly, MapReduce does not solve everything, is a solution for those problems that fit the model and can be parallelized. One of the important results of experimentation

is that having small size data sets to process (maybe up to 5 GB specifically for sorting) would suggest to avoid MapReduce at all (see Fig. 5). More specifically: adding computers to a cluster would not add real/proportional performance gains.

From using both, TestDFSIO and TeraSort it is possible to conclude that almost inexpensive infrastructure (basically: low-cost computers) enables the processing of large volumes of data. And the obtained performance in general terms, besides being linked to the implemented hardware, also depends on the correct configuration.

We have several lines of further work, taking advantage of the Hadoop experience we have acquired:

- Relative computing power evaluation, as aforementioned. We should design specific experiment scenarios and analysis.
- We should try gathering more computers, scalability in the tens, hundreds and thousands of nodes would give a better idea. Our results are exiting, but we know we have a limited number of available computers, and it is caused by our limited budget.
- We should try others benchmarks, by adapting current ones or developing new ones. An initial idea would be to identify different application areas, and use one or more benchmark per application area.

References

[1] S. Guo, Hadoop Operations and Cluster Management Cookbook, Packt Publishing, 2013.
[2] A. Holmes, Hadoop in Practice, 2nd. Ed., Manning Publications 2014.
[3] P. C. Zikopoulos, C. Eaton, D. deRoos, T. Deutsch, G. Laspis Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw-Hill Osborne Media, 2012.
[4] F. G. Tinetti, D. Barry, I. Aita, F. Páez, Distributed Search on Large NoSQL Databases, PDPTA2011, 2011.
[5] N. Scola, WhiteHouse.gov Goes Drupal [Updated], [Online]. Available: http://techpresident.com/blog-entry/whitehousegov-goes-drupal
[6] C. Henderson, Building scalable web sites, O'Reilly Media, Inc., 2006.
[7] O. M. Tamer, P. Valduriez, Principles of distributed database systems, Springer Science & Business Media, 2011.
[8] A. K. Elmagarmid, M. Rusinkiewicz, A. Sheth, Management of heterogeneous and autonomous database systems, Morgan Kaufmann, 1998.
[9] J. Venner, S. Wadkar, M. Siddalingaiah, Pro Apache Hadoop, Apress, 2nd ed., 2014.
[10] D. Taniar, C. H. C. Leung, W. Rahayu, S. Goel, High performance parallel database processing and grid databases, John Wiley & Sons, 2008.
[11] R. Ho, Scalable System Design Patterns, Pragmatic Programming Techniques, [Online]. Available: http://horicky.blogspot.com/2010/10/scalable-system-design-patterns.html 2010.

Towards adaptive execution strategies for large-scale and real-time data analytics

Martin Köhler[1], **Yuriy Kaniovskyi**[2], **and Siegfried Benkner**[2]
[1]AIT Austrian Institute of Technology GmbH, Vienna, Austria
[2]Research Group Scientific Computing, University of Vienna, Vienna, Austria

Abstract— *The ever increasing amount of data available to the scientific community poses new challenges for the design of large-scale data analytics systems. In particular, there is a need for bringing together recent developments from the fields of Big Data and High Performance Computing by integrating data-intensive programming paradigms, such as MapReduce, and compute-intensive paradigms for massively parallel architectures and accelerators. However, the development, configuration and execution of data analytics workflows on emerging compute- and data-intensive platforms are far from being easily manageable. In this work, we present the design and major aspects of an advanced data analytics framework that provides adaptive execution strategies for optimizing data analytics applications subject to user-defined QoS constraints on such platforms. Central to our approach is the design of a description model of the system environment. Based on the description model we aim to adapt a data pipelining framework to its environment considering multiple implementation variants of analytical modules based on different programming paradigms.*

Keywords: Big Data, High Performance Computing, framework design, adaptive, real time, optimization

1. Introduction

The amount of data available in various research fields is often enormous in size and available in different formats such as text files or binary data, or produced continuously through data streams. To meet ever increasing computational requirements there is a clear trend towards heterogeneous multi-core architectures, which combine conventional multi-core CPUs with different types of accelerators, like GPUs or co-processors. In both areas, Big Data and High Performance Computing, new disruptive technologies are emerging that are converging to meet the challenges exposed by large-scale data science in various domains.

These technologies lead to a shift in research towards data-driven approaches for exploring both existing and real-time generated data-sets, in addition to empiric, hypothesis-driven, and computational or simulation-driven methods. These new directions are often referred to as the fourth paradigm of science [1]. The realization of such a new paradigm poses many new challenges to the e-Science domain [2]. One of these challenges is to facilitate real-time access and interactive usage scenarios on top of complex data- and compute-intensive workflows.

Enabling real-time processing of large-scale compute- and data-intensive analytical workflows necessitates the integration of state-of-the-art technologies from the fields of Big Data and High Performance Computing. Data-intensive and massively parallel programming paradigms such as MapReduce and its open-source execution frameworks (e.g., Apache Hadoop) are at the centerpiece of many Big Data applications analyzing massive volumes of data. The current development focus aims on extending these systems for processing high velocity data and for real-time analytics. Massively parallel platforms, which combine conventional multi-core CPUs with different types of accelerators, like GPUs or co-processors have the potential to provide the required computational power, but their efficient utilization requires combing different programming models, such as MPI, OpenMP and CUDA, adding another level of complexity on top of established parallelization techniques.

However, even for experts, configuring and optimizing analysis workflows on such platforms is a complex and time-consuming task that requires detailed knowledge of the underlying hardware and software resources. Consequently, current approaches are often static (i.e. optimizations are performed at design time), restricted to a single application or the workflow layer only, and often assume a fixed execution environment. Therefore, there is an urgent need for adaptive mechanisms that autonomously configure and optimize workflows and their execution environment.

In this paper we present the design of a framework for real-time data analytics, motivated by new requirements in the domain of transportation [3]. Our framework is based on an existing modular data pipeline. For each module of the data pipeline, different implementation variants optimized for different types of execution units (e.g. multi-core CPUs, GPUs, accelerators) can be provided. The framework aims to adaptively optimize analytical workflows by selecting for each module the most efficient implementation variant, depending on the available hardware resources, such that execution times and data transfer costs are minimized. Central to our approach is a generic resource description model which captures major characteristics of storage and compute resources, the execution platform, the organization of the data to be analyzed, and the analytical workflow itself.

The remainder of the paper is structured as follows: Section 2 provides an overview of the design of the large-scale and real-time data analytics framework. Section 3 illustrates our environmental description model, while Section 4 discusses approaches towards adaptive execution strategies. Section 5 presents application scenarios from the transportation domain. Finally, we wrap up with concluding remarks, related work and future plans.

2. Design of the adaptive real-time data analytics framework

The adaptive real-time data analytics framework aims to transparently combine large-scale compute- and data-intensive workflows with stream processing to support real-time data analytics in the transportation domain. Our framework is based on a modular data pipeline platform and provides more than 40 modules for importing and exporting data from and to a variety of data sources, integrating and analyzing data, which can be flexibly arranged into data analytics workflows. In addition, a flexible plugin mechanism enables the development and integration of additional application-specific modules.

At present, the data pipeline platform can be deployed on shared memory systems only. Pipelines are executed by concurrent threads on data sources that are available locally. In case of remote data sources, data has to be transferred on demand to the execution system.

To overcome these restrictions and to enable real-time big data analytics on emerging heterogeneous parallel platforms, the pipeline platform is being extended through the integration of (1) stream processing technologies: processing sensor data in real-time requires the integration of a stream processing framework enabling the pre-processing of the data records and the adaptation of the pipelining platform towards the integration of live data into analytical pipelines; (2) data-intensive computing techniques: the amount of data necessitates a seamless integration of large-scale data storage systems, such as HDFS, and data-intensive computation approaches, such as MapReduce, in order to exploit data locality; and (3) high performance computing: parallel processing capabilities on multicore architectures as well as on GPUs and other types of accelerators. In order to support emerging processor architectures, the framework will allow the provisioning of new implementation variants for data analytics modules optimized for different parallel execution units using different programming models including MPI, OpenMP, OpenCL and CUDA.

To address the challenge of workflow orchestration, we design a generic resource description model that aims to systematically capture the main characteristics of the underlying hardware, the execution and storage platforms, the characteristics of the data sources and the application workflows. The model aims to support optimization and tuning of data analytics workflows for target execution platforms through selection of different implementation variants for each analytical module and devising a data placement and movement strategy in order to meet required quality of service specifications.

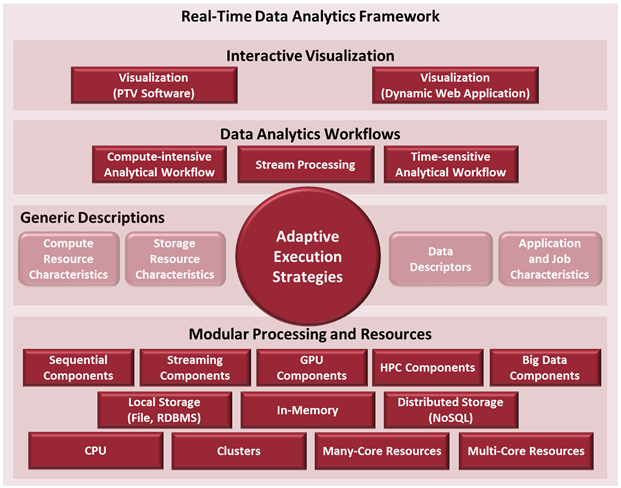


Fig. 1: High-level view of the adaptive real-time data analytics framework

Figure 1 provides a high-level view of the design of our adaptive real-time data analytics framework that takes into account all these considerations. The figure delineates the extended data pipelining framework, including storage resources, computing resources, and modules available for different programming paradigms. Implementation variants and available resources are characterized by means of generic descriptors (see Section 3), which form the basis for adaptive workflow tuning. On top of this, the framework supports the definition of analytical workflows, which can be separated into streaming-, time-sensitive-, and compute- and/or data-intensive workflows. Finally, results of the workflows will be visualized using state-of-the-art PTV software [4] and Web frontends.

2.1 Lambda architecture scheme

The framework adopts the lambda architecture scheme [5] to support real-time stream processing and batch processing of large data sets. The lambda architecture is a generic architectural design for scalable and fault-tolerant processing of large data volumes. It tackles the challenges of real-time stream processing and massively-parallel processing of large-scale data sources in a single architectural scheme by defining three system layers: the speed, the batch and the serving layers. The batch layer is utilized for storing the complete data set and supporting massively parallel execution of data-intensive jobs (e.g. MapReduce paradigm). The speed layer is introduced for handling new data and extracting knowledge from live data in real-time. Both layers

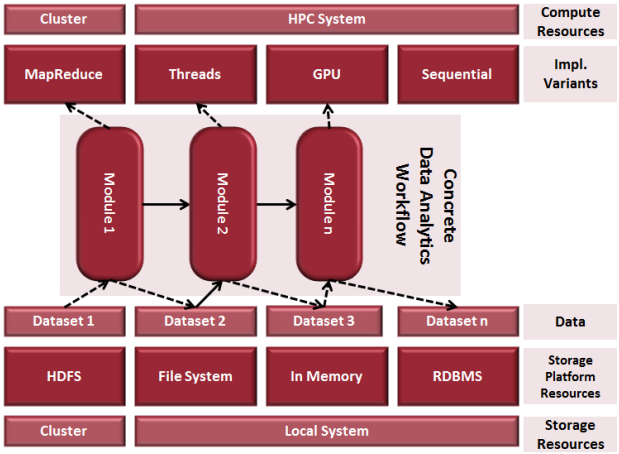


Fig. 2: Data Analytics workflow: flexible mapping of compute and storage resources, as well as implementation variants

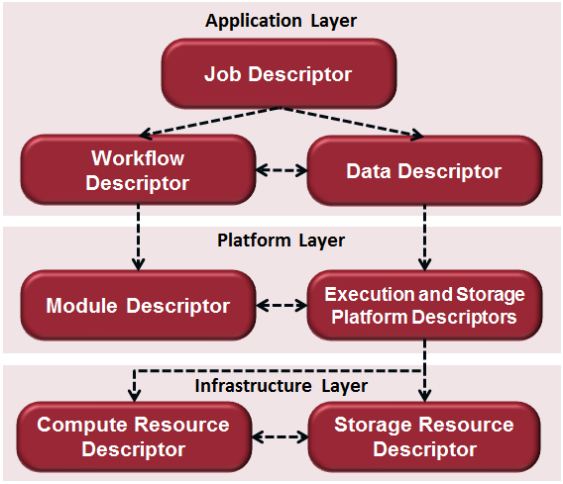


Fig. 3: Generic resource description model: categorization of descriptors for each resource type at different system layers. Connecting lines between descriptors indicate their dependencies.

store their results via the serving layer by incremental real-time and batch views. This type of architecture enables merging of queries by integrating batch views with incremental real-time views to allow incremental computation through incoming live data.

Following the lambda architecture, workflows are categorized into two types: stream (live data) and batch processing. In our case, time-sensitive and compute- and/or data-intensive workflows are considered batch processing workflows, which are explicitly triggered by the user. Contrary, streaming workflows are triggered by incoming data streams. Incoming data will be materialized in the storage system and directly used for incremental computations.

2.2 Data analytics workflows

A data analytics job is defined as an abstract workflow by means of a YAML file comprising module descriptions. Each module description includes the name, the type and the module specific parameters, including a list of input and output data pipelines. The execution ordering is determined through specifying and linking input and output pipelines.

The mapping of modules to specific implementation variants will be done by the adaptive execution strategies delineated in Section 4. An example mapping of a data analytics workflow is depicted in Figure 2. The figure illustrates a concrete workflow execution comprising multiple modules with assigned implementation variants and their respective execution platform. In addition, inputs and outputs of analytical modules are assigned to different data storage solutions in order to exploit data locality for a specific implementation variant.

In the presented example the first module (e.g. importer) is executed on the MapReduce execution platform since it gathers data from a scalable storage solution (HDFS). The data is forwarded to a local storage system (local file system) where the next module (filter) processes it. Assuming it fits, the result is stored in-memory, where the next module n, which is assumed to be compute-intensive, is executed on a GPU. The final results are then pushed to a RDBMS.

3. Generic resource description model

The generic resource description model is designed to support adaptive execution and data management strategies by systematically representing the main characteristics of all system layers and components. These include a description of the underlying hardware, the execution platforms, their respective programming paradigms, the characteristics of the data sources and the application. The organization of the resource descriptor model loosely follows the NIST model for Cloud computing [6]. We define different descriptors at the infrastructure, the platform and the application layers.

An overview of the generic resource descriptor model is depicted in Figure 3. The infrastructure layer (IaaS) comprises the systems' compute and storage resources. The platform layer (PaaS) comprises the execution and storage platforms, and the implementation variants. The application layer (SaaS) comprises descriptions of data analytics jobs (job specific execution context), the corresponding workflow descriptors, and the data sets involved. Through the subsequent dependencies of descriptors we are able to represent knowledge about the overall system, and navigate from a specific job to other environmental descriptors to determine what infrastructure and platform resources any particular job can be assigned to.

3.1 Infrastructure descriptors

The infrastructure layer comprises descriptors for compute and storage resources. Compute resource descriptors capture important characteristics of the available hardware resources ranging from single-node to multi-node parallel systems possibly equipped with GPUs or accelerators. Through a hierarchical format, the descriptor enables representation of available nodes and their respective properties, including the overall processing capacity, the operating system, and different I/O devices. Additionally, the compute resource descriptor holds information about various Machine PCI devices such as GPUs, Xeon Phi and other accelerators, as well as network and Infiniband interfaces. Compute resource descriptors are based on existing approaches such as PDL [7] and hwloc [8]. In addition, we explicitly represent storage resource descriptors complementing compute resource descriptors with information about the memory hierarchy detailing on cache, memory, disk and possible attached remote storage resources.

3.2 Platform descriptors

The platform layer captures relevant information about the execution platforms, the storage platforms, and the available modules (and their implementation variants) of the data pipeline framework. Execution platform descriptors comprise information about the available programming models, such as MapReduce, MPI or CUDA, and interlink compute resources that support them. The module resource descriptor is a registry of analytical modules and their available execution variants. Supported implementation variants for different execution platforms are mapped to the corresponding execution platform.

Storage platform descriptors represent various types of database management systems, ranging from in-memory systems to NoSQL databases. We differentiate distributed and local storage solutions and map them accordingly through the compute resource descriptor. The main goal of the storage platform descriptor is to represent performance-related properties of any data storage instance. Performance metrics are estimated through average throughput and latency for data pushing and fetching. Figure 4 illustrates the platform layer of the description model. Both, the storage and the execution platform descriptors are mapped to a specific node, or to a topology of nodes to determine which resources are available, and can be exploited to gain performance.

In addition, an invocation context provides a generic (key - value) configuration set of execution parameters for the target execution and storage platforms. These parameters can be set by either the adaptive mechanisms at run-time or the execution platform. Thus, an invocation context for an execution platform would include, among others, the number of task threads or, in case of Apache YARN for example, a specific task scheduler to be used for the execution.

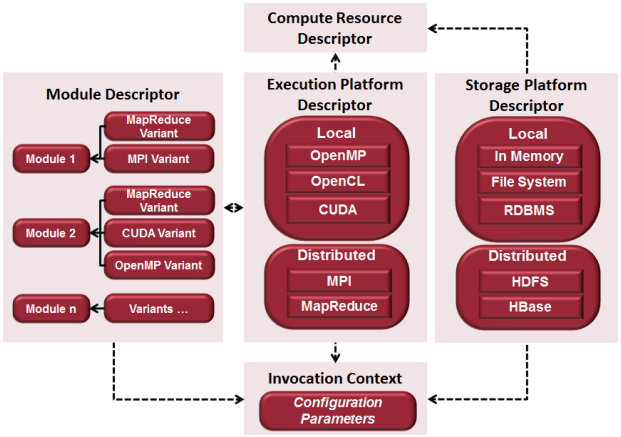


Fig. 4: Platform descriptors: the module descriptor comprises information about available implementation variants for analytical modules. The variants are mapped to the appropriate execution platforms that depend on compute resources. The invocation context provides a generic parameter set for execution and storage platform configuration.

3.3 Application descriptors

The descriptors at the application layer specify information about specific data analytics jobs together with the workflows and the data required for these jobs. The job descriptor represents a workflow execution request. In addition to quality of service policies, this descriptor is mapped to a workflow descriptor, which contains the abstract workflow of the analytical modules. Input and output data of the workflow are specified through dependencies to data descriptors.

Data descriptors hold the properties of specific data sets and are interlinked via dependencies to a storage platform. These dependencies can describe current storage of the data sets or possible future storage decisions. In addition, attributes describing data characteristics, such as cardinality, numerical distribution, time and spatial sparsity, as well as size are included.

Through subsequent dependencies, the job descriptor enables us to trace infrastructure and platform information of a particular job execution.

4. Adaptive execution strategies

In the following we describe our approaches towards adaptive execution strategies within our framework. An adaptive execution strategy is characterized by (1) a dynamic arrangement plan of concrete implementation variants of analytical modules on different execution platforms with respect to their computational and data requirements, and (2) a data placement plan that orchestrates data movement upon workflow execution. The latter is responsible for mapping input and output data of the complete workflow and the comprised modules, as well as deciding whether or not to

move data from the large-scale to local storage facilities for compute-intensive tasks. By taking both of these aspects into account, we aim to minimize bottlenecks with regard to computational performance and data transfer costs with the ultimate aim to minimize overall execution time such that real-time access and interactive usage scenarios become possible.

The implementation of the adaptive execution strategies will be based on a feedback control loop approach to select concrete implementation variants and data sets for a specific job to be executed. The loop adheres to the MAPE-K reference model [9], [10], [11], known from autonomic computing. The MAPE-K model formalizes a blueprint of an adaptive framework and its interactions with the managed element. In the following, the main components of the loop and their responsibilities in the context of the adaptive real-time analytics framework are described.

4.1 Control loop components

The control loop consists of two main elements: a managed resource and its autonomic manager. In our case, the managed resources are described by the generic resource description framework and involve the infrastructure and the platform. While the application layer is not considered a managed resource, its information regarding contextual workflow execution parameters (e.g. input data) is required for adaptive execution strategies. The objective of the autonomic manager is to minimize the runtime for a specific job.

The autonomic manager will be implemented transparently within the adaptive real-time analytics framework and consists of a knowledge component, a monitor, an analyzer, a planner, and an executor component, which together are responsible for devising the adaptive execution strategy for a specific job.

Upon receiving a workflow execution request, the control loop generates real-time information of the managed resources and utilizes the knowledge component for holding and sharing historic information to be utilized by all components during workflow analysis. Thus, knowledge is a database, holding not only the current state of the system, but also system training data in form of historical job executions.

The monitor filters and aggregates compute resource information within corresponding descriptors, while probing application metrics and performance parameters through the APIs of execution platform technologies upon workflow execution, with the purpose of supporting the systems performance models.

The analyzer determines the pool of available resources, including execution and storage platforms through analysis of the parameter scope within the descriptor information via the knowledge base. Together with the planner component, the loop establishes performance requirements of workflow modules regarding computational complexity and data management. Ranking of different configuration solutions is usually based on predictive models. Thus, some level of estimation or inaccuracy through a heuristic is inevitable.

Finally, the executer deploys the concrete workflow adapted by the planner on top of the execution platforms and starts the job.

4.2 Analyzing and planning

Throughout the analysis and planning stages of the loop, workflows are accompanied by performance models that enable the prediction of (relative) performance aspects of module implementation variants on different execution platforms. They are evaluated whenever a workflow is scheduled for execution and produce a performance description according to information contained in the knowledge component, including platform-independent information relevant at module invocation time (e.g. input data) and available computational resources (processing units, memory etc.) of the target platform. Performance models can be implemented in different ways including analytic methods, methods that rely on historical performance data, heuristics, or any combination of these approaches [12]. For instance, utility-based approaches, reinforced learning and probabilistic techniques can be utilized and are considered for learning which environmental parameters influence performance the most.

The utility function [13] designates a measure of usefulness of a concrete workflow. In our case utility represents a measure of system performance. In our previous works [14], [15] we applied the utility function to map a small set of environmental characteristics to determine the utility of a given job configuration in order to rank them appropriately. In the prototype we used an auxiliary function for the utility calculation that weighted environmental parameters using scalar weights. These however, had to be adapted manually if the environment or the application changed. Using the reinforced learning approach [16], the described adaptive mechanism will be able to scale the environmental characteristics according to the training based on historical data. The training would necessitate an extensive benchmarking of implementation variants on different representative data sets.

Recently, probabilistic techniques have been proposed throughout self-management literature [17] to provide a cost sensitive classification of feedback adjustments. By applying AI and Machine Learning approaches we can establish probabilistic performance models from historic executions. A Bayesian network model would describe domain variables from the environmental descriptors and establish probabilistic dependencies specified by conditional probability distributions. This model can provide a compact representation of multi-variate joint distributions and support efficient classification for inference tasks such as runtime prediction.

5. Application scenarios

The real-time data analytics framework is being developed in context of the Retida ("Real-Time Data Analytics in the Mobility Domain") project, funded by the Austrian Research Promotion Agency. Within the project, application scenarios are specified using a twofold approach: combining data-driven analytics capable of computing the current traffic flows on street networks and simulation-based traffic assignment on the street network. The project supports the utilization of the data analytics results in state-of-the-art transportation planning and management, traffic engineering and traffic simulation software PTV Visum [4]. A major goal is to facilitate interactive utilization of real-time analytics combined with simulations using the Visum software. As a result, we will tackle the challenge of implementing application scenarios for batch processing and for real-time calculation.

A real-time data analytics scenario demonstrates how various data sources can be combined and processed in an efficient manner. The application aims to calculate and visualize the current traffic state on the Austrian street network in real-time based on cell phone data [18] in order to detect and highlight untypical incidents such as car breakdowns.

In a second batch processing scenario we will exploit the capabilities of the framework and the diverse data sources for comprehensive comparisons of different traffic demand scenarios. This scenario will enable a detailed 'before and after' comparison on top of real-time data of this area and similar cases. The framework's ability to gather data for traffic demand modeling from one single interface reduces time consuming processes required before actual simulation steps and increases reliability.

6. Related work

An important contribution to automation and optimization of Big Data systems is the Stratosphere Project [19], now known as Apache Flink. Stratosphere features a programmable and scalable execution engine, automatic parallelization and query optimization techniques through a declarative query language. They offer a data analysis engine that acts through external heterogeneous data sources by connecting them ad-hoc to the framework. They use a query optimization engine that includes several different processing algorithms on top of the data source, thus exploiting data locality. In contrast, we focus on devising a framework that not only takes data placement strategies into account, but is also able to exploit and combine various programming paradigms from Big Data and High Performance Computing.

We also set a contrast to our previous works [14], [15], where we implemented a prototype for adaptive configuration of Apache Hadoop in the context of a molecular system biology application. Our current work builds upon this prototype and profoundly extends it. In our current design, we introduce a much larger set of environmental characteristics and performance metrics based on a generic description model. The strategies will also be extended through new approaches in the field of autonomic computing.

Other works that drive the autonomic incentive include the Automat toolkit [20], which is a community testbed architecture that targets research into mechanisms and policies for autonomic computing that are under closed-loop control. The toolkit enables researchers to instantiate self-management virtual data centers and to define the controllers that govern resource allocation, resource selection, and dynamic migration. Our adaptive strategies focus on resource selection mechanisms and self-configuration of the different execution environments.

In [21], utility functions in autonomic systems are used to continually optimize the utilization of managed computational resources in a dynamic, heterogeneous environment. The authors described a system that is able to self-optimize the allocation of resources to different application environments. We apply their approach to large-scale data processing systems.

The European PEPPHER project [22] proposed a component-based development approach for heterogeneous parallel systems, allowing to shield application developers from low-level implementation details while providing means for a seamless integration of different programming APIs, as well as for dynamic code adaptation and optimization. Parallel applications are composed at a high-level of abstraction from components, for which different implementation variants, optimized for different processing units, are provided. In our work, we extend this approach to a flexible data analytics pipeline, also taking into account data and storage characteristics.

7. Conclusion and future work

The ever increasing amount of data available to the scientific community raises new challenges on how to process and derive knowledge from this data. Many applications also require high performance systems in order to provide a level of viability in execution of scientific workflows. Data-intensive programming paradigms, such as Hadoop, and compute-intensive paradigms provide execution environments that are able to address these challenges. However, integration, configuration and orchestration of executions on these platforms is far from being manageable by humans.

In this work, we presented the design of a novel real-time data analytics framework that aims to combine the two worlds of Big Data and High Performance Computing. Our framework design, based on the recently proposed lambda architecture, supports integration and analysis of large volumes of data via batch and stream processing. The data analytics framework is implemented on the basis of a generic and modular data pipeline and incorporates different implementation variants of analytical modules, optimized for

different execution platforms such as Hadoop, MPI, OpenCL and CUDA.

Furthermore, we discussed adaptive mechanisms, based on concepts from autonomic computing [23], [12], that support autonomic mapping of analytical workflows to multiple platform execution environments, while taking into account its available resources and configuring them for resource exploitation. Central to our approach is a generic resource descriptor model, enabling to not only represent the underlying compute and storage resources available to the system, but also the platform execution environment and the application workflow characteristics in a generic manner. This information can then be used by an adaptation of the MAPE-K control loop to derive workflow mapping solutions that exploit the available resources according to QoS policies. We take into account state-of-the-art heuristic strategies enabling adaptive mapping of abstract to concrete execution plans. The adaptive mechanisms, including the generic resource description model and the execution strategies, will be implemented in the context of the real-time data analytics framework.

The presented data analysis framework is currently being implemented and will be evaluated based on application scenarios from the transportation domain. The application scenarios focus on on-demand calculation and analysis of the current traffic state and are developed in the context of the Austrian research project Retida.

Future work will include diverse deployment scenarios of the prototypical framework on emerging heterogeneous many-core platforms and the tight integration of the autonomic features according to the models presented in this work. We will present results in this context and answer important questions related to our system, namely (1) what resources and environmental characteristics pose a significance with regard to performance; (2) how to leverage available system resources in the best possible manner; and (3) how to optimize scientific workflows with regard to execution and data storage strategies.

Acknowledgment

The research leading to these results has received funding from the Austrian Research Promotion Agency (ICT of the Future/2014-2016) under grant agreement #845606 (Retida Project).

References

[1] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[2] T. Hey and A. E. Trefethen, "Cyberinfrastructure for e-science," *Science*, vol. 308, no. 5723, pp. 817–821, 2005.

[3] R. Kitchin, "The real-time city? big data and smart urbanism," *GeoJournal*, vol. 79, no. 1, pp. 1–14, 2014.

[4] PTV Group, "PTV Visum: http://vision-traffic.ptvgroup.com/de/produkte/ptv-visum/," 3 2015.

[5] N. Marz, *Big data : principles and best practices of scalable realtime data systems*. [S.l.]: O'Reilly Media, 2013.

[6] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," NIST - National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, United States, Tech. Rep., 2011.

[7] M. Sandrieser, S. Benkner, and S. Pllana, "Using explicit platform descriptions to support programming of heterogeneous many-core systems," *Parallel Computing*, vol. 38, no. 1-2, pp. 52 – 65, 2012, extensions for Next-Generation Parallel Programming Models.

[8] Inria, "Portable Hardware Locality (hwloc): http://www.open-mpi.org/projects/hwloc/," 3 2015.

[9] Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," *IBM Corporation (October 15, 2001)*, Oct. 2001.

[10] IBM, "An architectural blueprint for autonomic computing," IBM, Tech. Rep., June 2005.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00. New York, NY, USA: ACM, 2000, pp. 56–67.

[12] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing - degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, pp. 7:1–7:28, August 2008. [Online]. Available: http://doi.acm.org/10.1145/1380584.1380585

[13] WIKIBOOKS, "Principles of Economics: http://en.wikibooks.org/wiki/Principles_of_Economics/Utility," 7 2011.

[14] M. Köhler and S. Benkner, "Design of an adaptive framework for utility-based optimization of scientific applications in the cloud," in *The 2nd International Workshop on Intelligent Techniques and Architectures for Autonomic Clouds (ITAAC 2012), in conjunction with The 5th IEEE ACM International Conference on Utility and Cloud Computing (UCC 2012)*, USA, November 2012.

[15] M. Köhler, Y. Kaniovskyi, and S. Benkner, "An adaptive framework for the execution of data-intensive mapreduce applications in the cloud," in *The First International Workshop on Data Intensive Computing in the Clouds (DataCloud 2011)*. Anchorage, Alaska: IEEE, May 2011.

[16] G. Tesauro, "Reinforcement learning in autonomic computing: A manifesto and case studies," *Internet Computing, IEEE*, vol. 11, no. 1, pp. 22–30, Jan 2007.

[17] A. Bashar, G. Parr, S. Mcclean, B. Scotney, and D. Nauck, "Application of bayesian networks for autonomic network management," *J. Netw. Syst. Manage.*, vol. 22, no. 2, pp. 174–207, Apr. 2014.

[18] P. Widhalm, Y. Yang, M. Ulm, and M. Gonzales, "Discovering urban activity patterns in cell phone data," *Springer Transportation*, 2015.

[19] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke, "The stratosphere platform for big data analytics," *The VLDB Journal*, vol. 23, no. 6, pp. 939–964, Dec. 2014.

[20] A. Yumerefendi, P. Shivam, D. Irwin, P. Gunda, L. Grit, A. Demberel, J. Chase, and S. Babu, "Towards an autonomic computing testbed," in *In Proceedings of the Second Workshop on Hot Topics in Autonomic Computing*, 2007.

[21] W. Walsh, G. Tesauro, J. Kephart, and R. Das, "Utility functions in autonomic systems," in *Autonomic Computing, 2004. Proceedings. International Conference on*, May 2004, pp. 70 – 77.

[22] S. Benkner, S. Pllana, J. L. Träff, P. Tsigas, A. Richards, R. Namyst, H. Cornelius, C. Kessler, D. Moloney, and P. Sanders, "Peppher: A comprehensive framework for performance portability and programmability of heterogeneous many-core architectures," in *Programming Multi-Core and Many-Core Computing Systems*. Wiley Inc., December 2014.

[23] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41 – 50, Jan. 2003.

A Data Pre-partitioning and Distribution Optimization Approach for Distributed Data Warehouses

Billel ARRES
Universite Lumiere Lyon 2
69676 Bron, France
billel.arres@univ-lyon2.fr

Nadia KABACHI
Universite Lyon 1
69100 Villeurbanne, France
nadia.kabachi@univ-lyon1.f

Omar BOUSSAID
Universite Lumiere Lyon 2
69676 Bron, France
omar.boussaid@univ-lyon2.fr

***Abstract*—The increasing volumes of relational data let us find an alternative to cope with them. The Hadoop framework - an open source project based on the MapReduce paradigm - is a popular choice for distributed data warehouses and big data analytics. In this paper, we propose an original approach for partitioning and collocating data in distributed file systems, especially Hadoop-based systems, and this, to overcome the default data partitioning and placement policies which does not take any data characteristics into account. The goal is to reduce the amount of data transfer required during Mapreduce's shuffling phase. Indeed, the efficiency of many relational operations can be improved if a careful data fragmentation design and placement policy are applied, including indexing, grouping, aggregation and joins. Based on k-means clustering methode that allows to master the number of clusters through its k parameter, we investigate the performance gain for OLAP cube construction on a multi-nodes cluster with and without data organization. And this, by varying the number of clusters and data warehouse size. Our experiments suggest that defining a good data partitioning and placement schemes during the implementation of the data warehouse increase significantly the OLAP cube computation and querying performances.**

***Keywords*-Data warehouses; On-Line Analytical Processing (OLAP); Mapreduce; Data placement;**

I. Introduction

In the recent past, we have witnessed dramatic increases in the volume of data literally in every area: business, science, and daily life to name a few. Today, some claim that data (more specifically, data-intensive science) are the fourth paradigm in scientific research after experimentation, observation, theory, and computational simulation. The storage and processing of such an overwhelming amount of data is a challenging task in the current computing environments. To address the scalability requirements of today's data analytic, parallel shared-nothing architectures of commodity machines (often consisting of thousands of nodes) have been lately established as the de-facto solution.

MapReduce [9] is a well-known distributed framework for programming commodity computer clusters to perform large-scale data processing algorithm. Hadoop [17], an open-source MapReduce implementation, is able to process big data sets in a reliable, efficient and scalable way. Based on Hadoop, many cloud data warehouses (e.g., Hive [4], and HadoopDB [1]) are developed and widely used in various fields. Even though these data warehouses support, the performances are unsatisfactory. The reasons for this situation are: (1) these systems do not provide big data oriented OLAP optimizations; (2)the join operation, which is quite common operation in OLAP, is very inefficient when big data are involved [14]. In this paper, we studied the benefits of data partitioning and collocation within the context of data warehousing and OLAP querying. Based on query workload, we introduce a mechanism called reference table (RT) as an extension of Hadoop's distributed file system, to collocate dimensions tables predicates and attributes that are related or frequently used on the same or closest set of nodes, eliminating the network overhead by reducing data transfer in MapReduce's shuffle phase, since related set of files will be processed jointly. The rest of this paper is organized as follows. Following the introduction, Section 2 presents background and related work. Section 3 describes the problem we address in this paper. Section 4 details the proposed approach and system implementation. Section 5 evaluates the efficacy of the proposed approach. Finally, conclusions and future works are summarized in Section 6.

II. Related Work

OLAP (On-Line Analytical Processing) was introduced in the work done by [5]. They provided an overview of data warehousing and OLAP technologies, with an emphasis on their new requirements. Considering the past two decades have seen explosive growth, both in the number of products and services offered and in the adoption of these technologies in industry, their other work [6] gave the introductions of OLAP and data warehouse technologies based on the new challenges of massively parallel data architecture. There exist some optimization approaches of distributed data warehouses system, which are related to this paper. Data warehousing improvements can be classified as follows: (1) data layouts; (2) pre-computation for indexing; (3) intentional data placement. The latter two are close to our proposal and introduced briefly in this section. Data layouts aims to enhance MapReduce performance. In this case, Llama [19] proposes the use of a columnar file (called

CFile) for data storage in HDFS. This enables selective access only to the columns used in the query. However, this type of storage provides more efficient access to data than traditional row-wise storage only for queries that involve a small number of attributes. This is not always the case for data warehouses. For indexing, Hadoop++ [10] is a system that provides indexing functionality for data stored in HDFS by means of User Defined Functions (UDFs), i.e., without modifying the Hadoop framework at all. In fact, Hadoop++ can only collocate two files that are created by the same job, and requires reorganization of the data as new files are ingested into the system. In the case of data warehouses it is necessary to collocate all the data warehouse files (tables), not only two, and this can be a major inconvenience. Other research efforts tried to empower Hadoop by an intentional data organization. CoHadoop [13] is an extension of Hadoop to enable applications to control where data are stored. It retains the flexibility of Hadoop since it does not require users to convert their data to a specific format. However, it has not proved its efficiency on multidimensional data processing. In summary, at present the lack of effective support for multidimensional data processing model and OLAP analysis needs to be resolved urgently in big data era. At the same time, Hadoop as a cloud-computing framework is most widely used in the big data analysis platform, but the OLAP optimizations, based on Hadoop, are still blank. Based on our previous research [3], in which we presented a data collocation mechanism for big data environment, we designed the proposed approach as an extension of the Hadoop file system, which has been proven efficient on analysis of big data. This research has certain theoretical and practical values.

III. Problem Statement

The implementation of a data warehouse that incorporates the best features of the MapReduce parallel processing model - scalability and fault tolerance - is the goal of several research, eg [18] and [1]. For a program to be executed concurrently using several processing nodes in a parallel computer system (e.g. a cluster), the work has to be divided and assigned to each one of them. This usually requires also a given input dataset to be divided into blocks (chunks) and assigned to each of the nodes. This is carried out in a two-step process: data partitioning (or fragmentation) to divide the dataset, and data placement (or allocation) to assign the fragments to the system's nodes. By default, tables files are partitioned into a set of partitions (data blocks) using horizontal data partitioning (HDP) [15], then the Hadoop distributed file system (HDFS) tries to balance load by placing the blocks randomly on the Datanodes.

The default data placement policy of HDFS arbitrarily places partitions obtained across the cluster so that mappers often have to read the corresponding partitions from remote nodes. This causes a high data shuffling costs and network

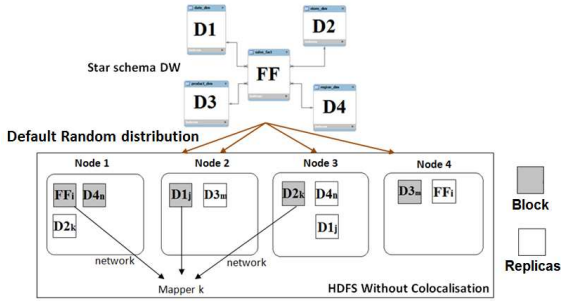


Figure 1: Default DW blocks processing.

overhead when querying step (Figure 1). The goal of our proposed approach is to minimize this cost instead of the network overhead by reducing the transferred data in the shuffle phase. In a context of data warehousing with Mapreduce, which is based on the Map and Reduce steps, transferring data between these two phases remind often more time consuming than processing data itself [2]. Let us formally define the transferred data which we want to minimize.

Definition 1: (Data transfer in shuffle phase)

The MapReduce framework operates exclusively on key-value pairs [17], that is, the framework views the input to the job as a set of key-value pairs and produces a set of key-value pairs as the output of the job. The output pairs can be different types than the input pairs.

Let job_α denote a MapReduce job. It is composed of $M_\alpha = \{m_1, ..., m_p\}$ map tasks and $R_\alpha = \{r_1, ..., r_q\}$ reduce tasks. We do not consider map or reduce tasks which fail or are the result of speculative execution and are not retained. We assume that each map task m_i processes chunk c_i, for $i = 1, ..., p$.

Let $I_\alpha = \{ip_1, .., ip_m\}$ be the set of intermediate key-value pairs produced by the map phase. $key(ip_j)$ represents the key of intermediate pair ip_j and $size(ip_j)$ represents its total size in bytes. K_α is defined as the set of intermediate keys produced in the execution of job_α, $K_\alpha = \cup_{ip \in I_\alpha} key(ip)$. We define $output(m_i) \subseteq I_\alpha$ as the set of intermediate pairs produced by map task m_i. We also define $input(r_i) \subseteq I_\alpha$ as the set of intermediate pairs assigned to reduce task r_i. This assignment is controlled by the reduce partitioning function:

$$part : K_\alpha \rightarrow R_\alpha$$

Let $N = n_1, .., n_s$ be the set of machines that compose the cluster; $node(t)$ represents the machine where task t is executed:

$$node : M_\alpha \cup R_\alpha \rightarrow N$$

The way in which this assignment is done depends on the scheduling algorithm, the properties of the job and the characteristics and behaviour of the cluster where it is executed.

Now we distinguish between local and remote transfers of intermediate tuples. Let ip_j be an intermediate key-value pair, produced in map task m, i.e., $ip_j \in output(m)$ and consumed by reduce task r, i.e., $ip_j \in input(r)$. We define $P_\alpha(ip_j) \in 0, 1$ as a variable that indicates whether ip_j is transferred or not through the network:

$$P_\alpha(ip_j) = \begin{cases} 0 & \text{if } node(m) = node(r) \\ 1 & \text{if } node(m) \neq node(r) \end{cases}$$

From function P_α we can derive the total amount of data that is transferred through the network in the execution of job_α.

$$transfer(job_\alpha) = \sum_{ip_j \in \alpha_i} size(ip_j) P_\alpha(ip_j)$$

Definition 2: Let $W = \{job_1, ..., job_w\}$ be the set of jobs in the workload sample. Our goal is to minimize the total amount of data transferred over the network in the shuffle phase of jobs involved in W:

$$minimize \left(\sum_{job_\alpha \in W} transfer(job_\alpha) \right)$$

by optimizing: (1) the default data partitioning with a vertical pre-partitioning of data warehouse schema before its implementation (2) data placement by collocating related partition's chunks obtained. In both cases, we propose a data mining based approach, which is performed transparently to the users, in order to free them from the burden of complex partitioning and collocating optimisations. Our approach is summarized in Figure 2.

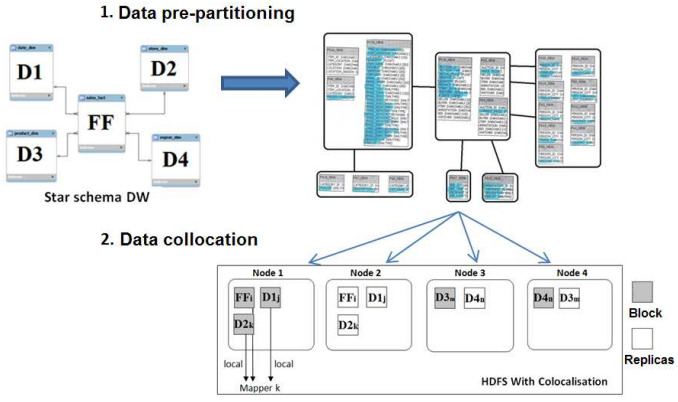


Figure 2: DW blocks processing with data pre-partitioning and collocation approach.

IV. The Proposed Approach

Data partitioning and data collocation are a well known and widely used optimisation methods (OM) in data base community. In the context of parallel processing, taking into account the existing interaction between these two OM's can increase significantly the optimizations results.

A. Data Pre-partitioning

Basically the original input data warehouse files (tables) are stored at a single node or outside the parallel system, then it has to be divided and transferred to each of the participating nodes. By default, Hadoop distributed file system partitions these input data files into large horizontal partitions (at least 64MB up to 1GB)[10] without taking any data warehouse schema characteristics into account. Our first optimisation approach consist on applying a vertical partitioning where each data warehouse table file is split in two or more tables having fewer columns but keeping the same number of rows. Based on a set of query workload, the process takes as input data warehouse tables files and outputs fragments by applying the partitioning algorithm described below. Note that all the process is done off-line.

1) Principle: The data warehouse schema consists of a set of fact and dimension tables along with their attributes. Vertical partitioning consists on splitting up a table by columns: one set of columns goes into one data store, and another set of columns goes into a different data store, it is necessary to select the appropriate dimension's attributes such that the query cost is minimized.

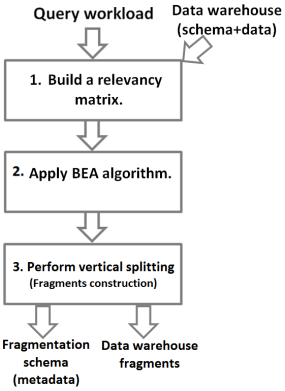


Figure 3: Data pre-partitioning principle.

2) Building the Relevancy Matrix: The process of building the Relevancy matrix has two main steps. The first one consists on generating a query attributes matrix QA from the query workload.

Let $T = \{t_1, ..., t_n\}$ be the set of the warehouse tables (facts and dimensions tables). The workload consists of a set of queries $W = \{q_1, ..., q_m\}$ and $\alpha_{ti} = \{a_e | a_e \in t_i\}$ be the set of attributes in T. The matrix QA represents couples (q_i, a_j), where general term QA_{ij} equal to 1 if a_j appears in query q_i and to 0 otherwise.

In the second step we build, from the obtained query attribute matrix, a Relevancy matrix M which is a n $\times$ n symmetric matrix, n is the number of different attributes in

the tenant's data, and the elements are the association degree between two attributes, it is defined as the number of queries which involves two attributes a_i and a_j, that is the frequency of occurrence of two attributes, it can be defined as follows:

$$M_{ij} = Relevancy(a_i, a_j) = \sum T(a_i, a_j)(a_i, a_j \in \alpha_{ti})$$

Among which, α_{ti} is the tenant's data attribute sets, a_i, a_j are the arbitrary two different attributes in α_{ti}, $T(a_i, a_j)$ is the operation (query) which involves a_i and a_j jointly, $\sum T(a_i, a_j)$ is the total number of operations which involves a_i and a_j.

3) Partitional Clustering Based on BEA: Bond Energy Algorithm [12], short for BEA, proposed by McCormick et al., is a rearrangement clustering technique which is widely used in vertical partitioning big tables in distributed database system. BEA takes the relevancy matrix as input, and keep replacing arbitrary two rows or columns so that the elements which has high relevancy could get together, and having a maximum ME. ME is short for Measure of Effectiveness, it represents the similarity between each element with the up and down and left and right the four neighbours, defined as follows:

$$ME(AA) = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N} a_{i,j}[a_{i,j+1} + a_{i,j-1} + a_{i+1,j} + a_{i-1,j}]$$

AA represents a M×N non negative matrix with boundary conditions: $a_{0,j} = a_{M+1,j} = a_{i,0} = a_{i,N+1} = 0$. If element in the matrix on the left side of the left or the right of the right matrix (upper and lower boundary similarly), the element value is 0, namely boundary elements and the elements outside the matrix have no relevancy.

According to the definition of BEA, clustering the relevancy matrix consists on making attributes which have higher relevancy get together so that ME have a maximum value. The objective function is defined as follows:

$$Max\{ME(AA) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} Relevancy(a_{i,j}[a_{i,j+1} + a_{i,j-1}])\}$$

The clustering process has two steps:

1) Reorganize the init relevancy matrix by first placing one of the columns arbitrarily, $i = 1$.
2) Try to place individually each of the remaining $N - i$ columns in each of the $i + 1$ possible positions (to the left and right of the i columns already placed), and compute each column's contribution to the ME. Place the column in the position that gives the largest incremental ME. Increment i by 1 and repeat this step until $i = N$.

The details of the above algorithm can be found in [12].

4) Fragments Construction: The last step takes the clustered relevancy matrix (BEA's output). Elements with similar values are grouped together to identify attributes clusters, making sure that subsets obtained after partition have the highest relevancy and are not breaking the privacy constraints, which are:

Completeness: Guaranteed by the partitioning algorithm, which assigns each attribute to one partition.

Reconstruction: A relation R decomposed into fragments R_1,R_2,...,R_n is reconstructed by the join operation: $R = \bowtie_K R_i$, for all R_i.

Disjointness: Attributes have to be disjoint in vertical fragmentation. Two cases are distinguished: If tuple IDs are used, the fragments are really disjoint. Otherwise, key attributes are replicated automatically by the system.

At the end of the process, we obtain a data warehouse fragments which will be transferred to the distributed file system. These fragments, initially obtained by a vertical fragmentation of the initial data warehouse, will be then horizontally fragmented by the distributed file system and distributed to data nodes automatically. At this point we can affirm that this first optimisation, as it combines two different methods of slicing tables, improves performance of queries as we will confirm in the evaluation section. The main advantage is that each of the resulting tables will have a lot less rows and also less attributes. Thus the method spatially expands a very large dimension, both vertically and horizontally, into tables of far smaller sizes.

B. Data Collocation

The main idea of our work suggests that to improve data warehouse query response time, particularly OLAP queries, we must first define a good strategy for data partitioning and distribution. As data partitioning was explained above, we detail in this section the second part of our approach which consist on a blocks placement strategy as an extension of the distributed file system.

1) Principle: Our collocation approach exploit selection dimension's attributes found in workload's queries to derive suitable blocks placement. It outputs a data warehouse distribution schema (metadata) and is subdivided into three steps that are:

1) dimension's attributes extraction from the workload;
2) dimension's clustering (using k-means method);
3) reference table construction for data collocation.

2) Dimension's Attributes Extraction: Selection attributes set is simply parsed from workload. Let $D = \{d_1, ..., d_n\}$ be the set of the warehouse tables. The workload consists of a set of queries $W = \{q_1, ..., q_m\}$.

Let $\alpha_{qi} = \{a_e | a_e \in q_i\}$ be the set of attributes related to a query q_i and $\beta_{dj} = \{b_f | b_f \in d_j\}$ the set of attributes of a dimension d_j. To make it simple, let consider a query that involves two attributes a_1 and a_2 belonging to d_1 and

d_2 respectively, our strategy consists on collocating the d_{1j} and d_{2k} blocks (chunks) on the same or closest nodes, while remaining tables blocks will be placed via Hadoop's default strategy over the remaining nodes. For that, as a first step, parsed dimensions are coded in a query-dimension matrix QM whose general term QM_{ij} equals to 1 if $\exists b_f$ in β_{dj} which is also in α_{qi}, and to 0 otherwise. For example, the QM matrix corresponding to W and D is featured in Table 1.

	d_1	d_2	d_3	d_4	...
q_1	1	1	0	0	
q_2	0	0	1	0	
...					
q_{20}	0	0	0	1	

Table I: Sample query-dimensions matrix QM

3) Dimensions Clustering: Our objective is to derive blocks placement schema that optimize data access for queries, especially OLAP queries. Since the HDFS default block placement policy does not take any data characteristics into account, clustering data warehouse blocks files with respect to queries achieves our goal. Intuitively, we ideally seek to build rectangles (clusters) of 1's in matrix QM. For study purpose, we chose the widely-used k-means algorithm [11] for clustering. But other clustering algorithms can be used. This algorithm inputs vectors of object attributes (columns of QM in our case). It attempts to find the centers of natural clusters in source data by minimizing total intra-cluster variance:

$$\sum_{i=1}^{k} \sum_{x_j \in C_i} (x_j - \mu_i)^2$$

where C_i, i = 1, ..., k are the k output clusters and μ_i is the centroid (mean point) of points $x_j \in C_i$. Let C be the set of all clusters C_i. Usually, having k as an input parameter is viewed as a drawback in clustering. In our case, this turns out to be an advantage, since we want to limit the number of clusters, typically to the number of nodes or sets of nodes (Racks) the data warehouse will be distributed on. In practice, we used the Weka [8] SimpleKMeans implementation of k-means. SimpleKMeans uses the Euclidean distance to compute distances between points and clusters. It directly inputs matrix QM (acually, the d_j vectors) and k, and outputs set of dimensions clusters C. For example, on matrix QM (Table 1) with k = 3, SimpleKMeans outputs: C = ((d_1,d_2),(d_3),(d_4)).

4) Reference Table Construction: The reference table (denoted RT) construction step consists on assigning each dimensions clustering output (C_i) to a reference (Ref(i)), each reference Ref(i) is represented by a random integer value, but other data types may also be used. There is an N:1 relationship between clusters C_i and references.

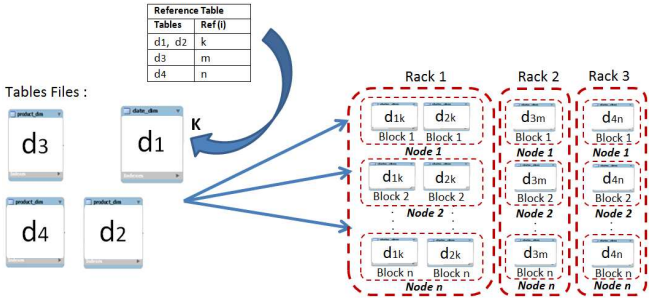


Figure 4: Example of four tables blocks collocated using the Reference Table on a multi-nodes cluster.

During the loading phase, each data warehouse table file is assigned to at most one reference and many tables files can be assigned to the same reference. Tables file's chunks with the same reference are then placed on the same (or closest) set of Datanodes, whereas others with no reference are placed via Hadoop's default strategy. The reference table is set with default values according to the policy location initially defined. Figure 4 shows the RT corresponding to four tables files collocation on a cluster.

In our work, we extended the Hadoop file system HDFS to support the customized data warehouse placement policy. We did not address other aspects such as the variation of the blocks size, which remains equal to default (64MB). Nor the replication policy which remains to default (Replication factor to 3). Our goal is the study of the plain query performance gains due to careful data warehouse organization in the context of parallelization with MapReduce. We used on top of our extended Hadoop version the Apache Hive [4] which is a data warehouse software that allows querying and managing large datasets residing in distributed storage. Hive provides an SQL-like language called HiveQL and has also support for creating data cubes[4].

V. Experimental Results

In the experiments, we aim at comparing our data pre-partitioning and distribution approach, which is the output of this project work, to default strategy used on Hadoop based clusters. We compare the performances (Execution time) of queries presented in section 5.1, first, without optimization (Default) using the original Hadoop version, then with optimization (Optimized) using our HDFS extension.

A. Experimental Conditions

In order to assess the effectiveness of our approach, we designed an adapted benchmark targeted to multidimensional data, called TPC-OH benchmark, which is inspired from the well-known TPC-H benchmark [16], the most prominent decision support benchmark. TPC-H benchmark consists of a suite of business oriented complex ad-hoc queries and concurrent data modifications. The workload and the data

Table II: DW Vertical Pre-partitioning results

Schema	Initial (nbr)	Fragmented (nbr)
LineItem	01	07
Orders	01	06
Customer	01	05
PartSupp, Supplier, Part	01	03
Region, Nation, Time	01	01

populating the database have been chosen to have broad industry-wide relevance. The workload is composed of 22 SQL queries with a high degree of complexity. Existing TPC-H implementation allows the generation of raw data stored into eight TBL files, namely Region, Nation, Customer, Supplier, Part, PartSupp, Orders, LineItem, by using a specific scale factor SF. The latter determines the final TPC-H data size. Basically, TPC-OH is a suitable transformation of the TPC-H benchmark into a multi-dimensional OLAP benchmark. Indeed, each business question of TPC-H workload is mapped into an OLAP star-join query, and a temporal dimension (Time table) is added to the data warehouse. Such star-join query use "`with cube`" operator to group the results. Also, we translated the TPC-H SQL workload into an HQL workload, since we used the Apache Hive as data warehousing software for querying and managing the data residing in Hadoop distributed storage system.

B. Cluster Setup

To achieve evaluation objectives, we used 10 PCs in a cluster (1 NameNode, 9 DataNodes). The NameNode is equipped with Intel Xeon E5-2630 (Six Core HT,2.6GHz), 16GB (4x4GB) RAM, and a 1TB SATA Hard Drive. The DataNodes are equipped with Intel Core i5-2400M, 4GB RAM, and a 300GB HDD. The OS is Ubuntu 12.04 LTS, and the Mapreduce framework is Hadoop 1.0.3. The network speed is 1G bps. We implement our data warehouse blocks placement approach by using the HDFS-385 API (Version 1.2.0) which is an expert-level interface for developers who want to try out custom placement algorithms to override the HDFS default placement policy [7]. Our code is written in Java and is available on demand.

C. Data loading

Table 2 shows the pre-partitioning schema (number of fragments) of TPC-OH relational data warehouse obtained, with respect to the workload as explained in Section 4.1. For data collocation, we arbitrarily fixed k-means parameter k = 10 to process the workload, which could correspond to cluster's size. In this first experiment, we investigate the effect of the proposed pre-partitioning and collocation approach on time needed to load data warehouse tables by the modified (Optimized) file system and the original one (Default). The data warehouse size is equal to 920GB.

From Figure 5 it can be seen that loading data time decreases significantly as the cluster size increases. The most noticeable difference between the two systems is on a 16 nodes cluster size, where about 3493,11 seconds elapsed for the newly (optimized) file system to execute successfully, however, the original (default) Hadoop file system takes less than 3247,2 seconds. A difference of 7 percent. In fact, to load data files the two systems performed a full MapReduce job. However, the optimized file system is slightly slower, as shown in Figure 5. This is due to the increased network utilization by collocating different files partitions (blocks). Note that with these initial tests performed in this work, loading cost encountered by the proposed approach is very small compared to the savings at query time as shown in the next section.

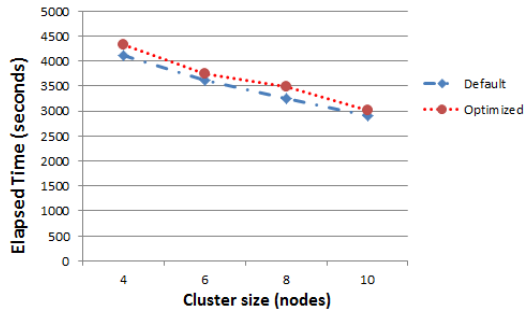


Figure 5: Loading data warehouse time (920 GB)

D. Query Execution

In this part of experiments, we used typical business queries of TPC-H [16] for which the shuffle phase has a significant impact in the response time. With respect to the partitioning scheme shown in Table 2, we used the following queries: Q5 and Q9 that are examples of hash joins on different columns, Q7 that executes a replicated join and Q17 that executes a co-group. Figure 6 shows that the proposed approach improves clearly query response time, especially for Q17 performing co-group operation, and this was expected since collocating related data blocks leads to map only tasks job operations rather than map and reduce tasks.

In contrast, Figure 7 shows the elapsed time for data cube computation by varying the data warehouse size on a 10 nodes cluster (N). For data collocation, we fixed k = 10 to process the workload as in the previous tests. As shown in the figure, cube computation execution time increases significantly as the data size increases and the benefits of the proposed approach are appreciable with the increasing size compared to default (HDFS) data distribution. The Figure shows that tables files collocation improves the query performance from 10% for 160GB to 25% for a 920GB data warehouse size. This behaviour is expected since collocation of data, in the context of data warehousing, avoids network

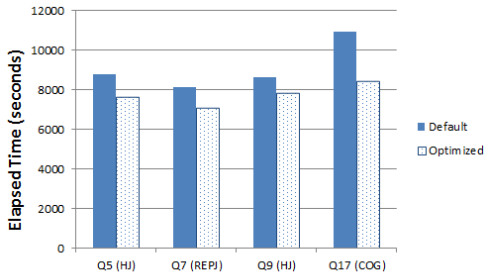


Figure 6: Queries execution time (920GB)

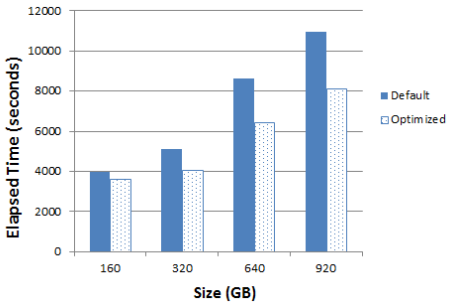


Figure 7: Building data cube

overhead, besides, it reduces the expensive data shuffling operation, which is the most time consuming phase for Mapreduce compared to default data placement policy.

E. Influence of Number of Clusters

In this experiment, we studied the effect of data collocation on the query response time by varying the clustering parameter k. As previous tests, we fixed the cluster size (10 nodes), data warehouse size (to 640 and 920 GB) and varied K-means parameter k to observe its influence on a sample query response time. Figure 8 confirms that performance improves quickly when collocation is applied, but tends to degrade when the number of clusters increases. Furthermore, it hints that an optimal number of clusters for tested data warehouse and workload benchmark lies between 5 and 6, making us conclude that over-collocation can be harmful and must be detected and avoided. Note that, on Figure 8, k = 1 corresponds to performance records when no collocation is applied (this one collocation is the HDFS default warehouse distribution for all DW's tables).

It is important to note that the distribution and collocation (of data and processing) using parallel programming models like Mapreduce are two opposing optimization techniques. In fact, collocation helps to minimize the data flow between the map and reduce phases (the shuffle phase) in order to avoid network overhead, applying the principle that says "Moving calculation is cheaper than moving data", allowing the intermediate results produced by the map phase to be executed on the same node. However, by seeking to collocate

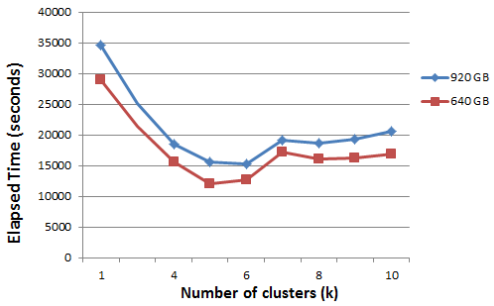


Figure 8: Influence of number of clusters

data, treatments will be centralized, making the task of each node heavier by creating a big load imbalance and therefore slower processing results. In this case, the objective is to find the right balance between distribution and collocation data, ie finding the right "k" which in our case is equal to 5 as shown in Figure 8.

VI. Conclusions And Future Work

In this paper, we present a data warehouse pre-partitioning and placement policy as an extension of the distributed file system to override the default policy and improve query performances. Our approach is simple yet flexible; it can be exploited in different ways by distributed and Hadoop-based data warehousing solutions. We studied the performance of our approach under different settings and compared it with default plain Hadoop solutions. Our experimental results show that data pre-partitioning and collocation optimization approach, outperforms default placement strategy in terms of performance gain by reducing the overhead of data shuffling and network. In next step, we will extend the experiments to study the effects of other configuration parameters on collocation data in the context of parallel data warehousing such as partitions size, replication factor and OLAP query complexity. We are also studying an intentional data placement strategy for large data warehouses with the integration of Multi-Agent Systems (MAS) and Intelligent Agents to the process, making clusters self-organized and autonomous dealing with new data and queries which are not included in the system's workload and are appended continuously.

References

[1] A.Abouzeid, K.B.Pawlikowski, D.Abadi, A.Silberschatz, and A.Rasin. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[2] A.Elsayed, O.Ismail, and M.E.El-Sharkawin. Mapreduce: State-of-the-art and research directions. *International Journal of Computer and Electrical Engineering*, 6(1):34–39, 2014.

[3] B. Arres, N. Kabachi, and O. Boussaid. Optimizing olap cubes construction by improving data placement on multi-nodes clusters. *23rd International Conference on Parallel, Distributed, and Network-Based Processing*, 45:520–528, 2015.

[4] A.Thusoo, J.S.Sarma, N.Jain, Z.Shao, P.Chakka, S.Anthony, H.Liu, P.Wyckoff, and R.Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[5] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[6] S. Chaudhuri, U. Dayal, and V. Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98, 2011.

[7] A. S. Foundation. Design a pluggable interface to place replicas of blocks in hdfs. *https://issues.apache.org/jira/browse/HDFS-385*, 2014.

[8] G.Holmes, A.Donkin, and H.Witten. Weka: a machine learning workbench. *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems*, 14:357–361, 1994.

[9] J.Dean and S.Ghemawat. Mapreduce: simplified data processing on large clusters. *Communictions of the ACM*, 51:107–113, 2008.

[10] J.Dittrich, J.A.Quiane-Ruiz, A.Jindal, Y.Kargin, V.Setty, and J.Schad. Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). *Proceedings of the VLDB Endowment*, 3(1):515–529, 2010.

[11] J.MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1:281–297, 1967.

[12] W. T. McCormick Jr, P. J. Schweitzer, and T. W. White. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, 20(5):993–1009, 1972.

[13] M.Eltabakh, Y.Tian, F.Gemulla, A.Krettek, and J.McPherson. Cohadoop: Flexible data placement and its exploitation in hadoop. *PVLDB*, 4(9):575–585, 2011.

[14] J. Song, T. Li, X. Liu, and Z. Zhu. Comparing and analyzing the energy efficiency of cloud database and parallel database. *Advances in Computer Science, Engineering and Applications*, 167:989–997, 2012.

[15] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: friends or foes? *Communications of the ACM*, 53(1):64–71, 2010.

[16] TPC-H. Transaction processing performance council. *http://www.tpc.org/tpch*, 2012.

[17] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[18] W.Huiju, Q.Xiongpai, Z.Yansong, W.Shan, and W.Zhanwei. Lineardb: A relational approach to make data warehouse scale like mapreduce. In *Database Systems for Advanced Applications*, volume 6588, pages 306–320. 2011.

[19] Y.Lin, D.Agrawal, C.Chen, and S.Wu. Llama: leveraging columnar storage for scalable join processing in the mapreduce framework. *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 961–972, 2011.

Big-ETL: Extracting-Transforming-Loading Approach for Big Data

M. Bala[1], **O. Boussaid**[2], **and Z. Alimazighi**[3]
[1]Department of informatics, Saad Dahleb University, Blida 1, Blida, Algeria
[2]Department of informatics and Statistics, University of Lyon 2, Lyon, France
[3]Department of informatics, USTHB, Algiers, Algeria

Abstract— *ETL process (Extracting-Transforming-Loading) is responsible for (E)xtracting data from heterogeneous sources, (T)ransforming and finally (L)oading them into a data warehouse (DW). Nowadays, Internet and Web 2.0 are generating data at an increasing rate, and therefore put the information systems (IS) face to the challenge of big data. Data integration systems and ETL, in particular, should be revisited and adapted and the well-known solution is based on the data distribution and the parallel/distributed processing. Among all the dimensions defining the complexity of the big data, we focus in this paper on its excessive "volume" in order to ensure good performance for ETL processes. In this context, we propose an original approach called Big-ETL (ETL Approach for Big Data) in which we define ETL functionalities that can be run easily on a cluster of computers with MapReduce (MR) paradigm. Big-ETL allows, thereby, parallelizing/distributing ETL at two levels: (i) the ETL process level (coarse granularity level), and (ii) the functionality level (fine level); this allows improving further the ETL performance.*

Keywords: Data Warehousing, Extracting-Transforming-Loading, Parallel/distributed processing, Big Data, MapReduce.

1. Introduction

The widespread use of internet, web 2.0, social networks, and digital sensors produce non-traditional data volumes. Indeed, MapReduce (MR) jobs run continuously on Google clusters and deal over twenty Petabytes of data per day [1]. This data explosion is an opportunity for the emergence of new business applications such as Big Data Analytics (BDA); but it is, at the same time, a problem given the limited capabilities of machines and traditional applications. These large data are called now "big data" and are characterized by the four "V" [2]: *Volume* that implies the amount of data going beyond the usual units, the *Velocity* means the speed with which this data is generated and should be processed, *Variety* is defined as the diversity of formats and structures, and *Veracity* relates to data accuracy and reliability. Furthermore, new paradigms emerged such as *Cloud Computing* [3] and *MapReduce* (MR) [4]. In addition, novel data models are proposed for very large data storage such as *NoSQL* (Not Only SQL) [5]. This paper aims to provide solutions to the problems caused by the big data in a decision-support environment. We are particularly interested in the very large data integration in a data warehouse. We propose a parallel/distributed ETL approach, called Big-ETL (ETL Approach for Big Data), consisting of a set of MR-based ETL functionalities. The solution offered by the research community, in this context, is to distribute the ETL process on a cluster of computers. Each ETL process instance handles a partition of data source in parallel way to improve the performance of the ETL. This solution is defined only at a process level (coarse granularity level) and does not consider the ETL functionalities (fine granularity level) which allows understanding deeply the ETL complexity and improve, therefore, significantly the ETL process. To the best of our knowledge, Big-ETL is a different and original approach in the data integration field. We first define an ETL process at a very fine level by parallelizing/distributing its core functionalities according to the MR paradigm. Big-ETL allows, thereby, parallelization/distribution of the ETL at two levels: (i) ETL functionality level, and (ii) ETL process level; this will improve further the ETL performance facing the big data. To validate our Big-ETL approach, we developed a prototype and conducted some experiments.

The rest of this paper is structured as follows. Section 2 presents a state of the art in the ETL field followed by a classification of ETL approaches proposed in the literature according to the parallelization criteria. Section 3 is devoted to our Big-ETL approach. We present in Section 4 our prototypical implementation and the conducted experiments. We conclude and present our future work in Section 5.

2. Related work

One of the first contributions on the ETL field is [6]. It is a modeling approach based on a non-standard graphical formalism where ARKTOS II is the implemented framework. It is the first contribution that allows modeling an ETL process with all its details at a very fine level, i.e. the *attribute*. In [7], authors proposed a more holistic modeling approach based on UML (Unified Modeling Language) but with less details on ETL process compared to [6]. Authors in [8] adopted BPMN notation (Business Process Model and Notation), a standard notation dedicated to the business process modeling. This work was followed by [9], a modeling framework based on a metamodel in MDD (Model Driven Development) architecture. [7] and [8] are top-down approaches and allow, therefore, modeling

sub-processes in their collapsed/expanded form for more readability. Authors in [10] proposed a modeling approach which consists of a summary view of the ETL process and adopt the Reo model [11]. We consider that this contribution could be interesting but is not mature enough and deserves Reo customization model to support the ETL specifics.

Following the big data emergence, some works tackled interesting issues. [12] is an approach which focuses on the performance of ETL processes dealing with large data and adopts the MapReduce paradigm. This approach is implemented in a prototype called ETLMR which is a MapReduce version of the PygramETL prototype [13]. The ETLMR platform is demonstrated in [14]. [15] shows that ETL solutions based on MapReduce frameworks, such as Apache Hadoop, are very efficient and less costly compared to ETL tools market. Recently, authors in [16] proposed CloudETL framework. CloudETL uses Apache Hadoop to parallelize ETL processes and Apache Hive to process data. Overall, experiments in [16] shows that CloudETL is faster than ETLMR and Hive for large data sets processing. [17] demonstrates the P-ETL platform. P-ETL (Parallel-ETL) is implemented under the Apache Hadoop framework and provides a simple GUI to set an ETL process and the parallel/distributed environment. In the batch version, P-ETL runs thanks to an XML file (*config.xml*) in which the same parameters should be set. In the P-ETL approach, the mappers (Map step) are in charge of standardizing the data (cleasing, filtering, converting, ...) and the reducers (Reduce step) are dedicated for merging and aggregating them. To the best of our knowledge, there are no works having tackled the ETL modeling issue intended to the big data environment and more precisely to the parallel/distributed ETL processing. We focus in this paper on the parallelization/distribution issue to improve the performance of the ETL. The classification proposed in Tab.1 is based on the parallelization criteria.

Table 1: Classification of ETL works

Approach	Purpose	Classification
[6]	Modeling	Centralized approach
[7]	Modeling	Centralized approach
[8]	Modeling	Centralized approach
[13]	Performance	Centralized approach
[12]	Performance	Distributed approach
[10]	Modeling	Centralized approach
[15]	Performance	Distributed approach
[16]	Performance	Distributed approach
[17]	Performance	Distributed approach
Big-ETL	Performance	Distributed approach

a) Centralized ETL process approach: In this paper, the ETL process approach is defined as centralized (or classical) when (i) the ETL process runs on an ETL server (one machine), (ii) in one instance (one execution at the same time), and (iii) the data are with moderate size.

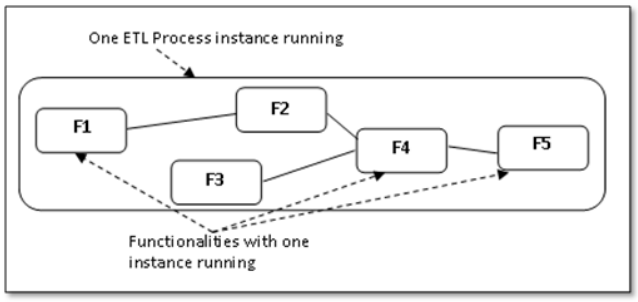


Fig. 1: Centralized ETL Process approach.

In this context, only the independent functionalities can be run in parallel way (both the ETL functions and the machine, on which it will be run, should be multithreaded). An ETL functionality, such as Changing Data Capture (CDC), Surrogate Key (SK), Slowly Changing Dimension (SCD), Surrogate Key Pipeline (SKP), is a basic function that supports a particular aspect of an ETL process. In Fig. 1, we can see that (*F1* and *F3*) or (*F2* and *F3*) can be run in parallel way.

b) Distributed ETL process approach: The well-known solution to cope with big data is the "parallelization/distribution" of the data and the ETL process on a cluster of computers. The MR paradigm, for instance, allows splitting large amounts of data sets where each partition will be subject to an instance of the ETL process.

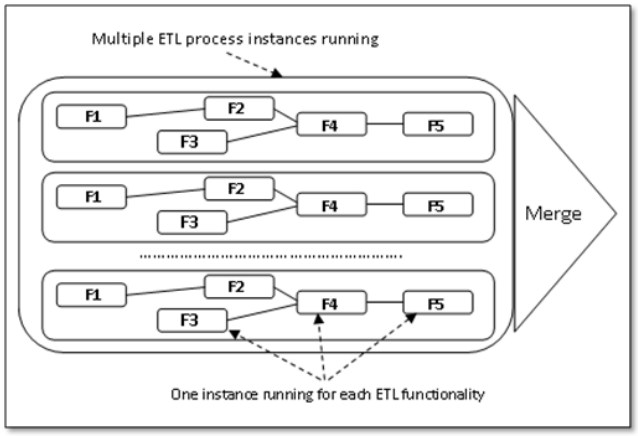


Fig. 2: Distributed ETL Process approach.

As depicted in Fig. 2, multiple ETL process instances run in parallel way where each one deals with its data partition in the Map step. The partial results produced by the mappers are merged/aggregated in the Reduce step and then loaded into the DW. All approaches proposed with MR paradigm, [12] and [15] for instance, apply the distribution

only at the process level. Big-ETL applies MR at two levels: (i) Process level (coarse granularity level), and (ii) functionality level (fine granularity level). We believe that the ETL, in the context of technological change having affected both data and processes, still presents some scientific problems such as big data modeling considering its different characteristics (volume, variety, velocity, veracity,...), data partitioning, parallel processing in its various forms (processes parallelization, process components parallelization, pipeline parallelization,...), etc. Functionalities as the core ETL functions deserve a most in-depth study to ensure, at a very fine level, robustness, reliability and optimization of the ETL process. Our Big-ETL is a parallel/distributed ETL approach based on two distribution levels (Process and functionalities) and two distribution directions (Vertical and horizontal).

3. ETL Approach for Big Data

We present in this section our Big-ETL approach. We deployed it on many ETL functionalities such as Changing Data Capture (CDC), Data Quality Validation (DVQ), Surrogate Key (SK), Slowly Changing Dimension (SCD), Surrogate Key Pipeline (SKP). Among all these ETL functionalities, we chose to present, in this paper, CDC to illustrate our Big-ETL approach.

3.1 Big-ETL principle

Our Big-ETL process is functionalities-based approach which exploits the MR paradigm. For each of these functionalities, we apply the same principle adopted at a process level in the "distributed ETL process approach" as depicted in Fig. 2.

3.1.1 Key Concepts

a) ETL functionality: In order to control the complexity of the ETL process, we define it thanks to a set of core functionalities. The ETL functionality is a basic function that supports a particular ETL aspect such as Changing Data Capture (CDC), Data Quality Validation (DQV), Surrogate Key (SK), Slowly Changing Dimension (SCD), Surrogate Key Pipeline (SKP), etc. The ETL task, however, is an instance of an ETL functionality. Let *SK1* and *SK2* be two ETL tasks that generate a surrogate key for inserting tuples in *PRODUCT* and *CUSTOMER* dimensions respectively. *SK1* and *SK2* are two different tasks but both are based on SK. Thus, the SK is the ETL functionality where SK1 and SK2 are its instances. In the follows, we describe an ETL process in terms of its functionalities.

b) Elementary process/function: When an ETL functionality is not atomic (aggregate functionality) in terms of processing, we consider that it is in charge of several separated elementary processes where each one is affected for an elementary function. An elementary process is an atomic unit of processing which is synchronized with other elementary processes to ensure the ETL functionality. Each one of the elementary processes is implemented as an elementary function. Thus, we consider that the aggregate functionality is a set of synchronized elementary functions. For example, the functionality CDC which is responsible to identify the changes (INSERT, UPDATE, DELETE) having affected the data in a particular source, can be decomposed in three elementary functions where each one is in charge of identifying INSERT, UPDATE, DELETE respectively.

3.1.2 Vertical Distribution of Functionaltilies (VDF)

As shown in Fig. 3, the ETL process runs in one instance, while each of its functionalities runs in multiple instances. For example, the functionality *F4* (oval) that runs in three instances (fragments separated by dashes), received its input data from *F2* and *F3*. These inputs are partitioned and each of the three partitions is subject to an instance of *F4* (mapper). Partial results produced by the three mappers are merged by reducers to provide the final *F4* outputs. This is a novelty in the parallel/distributed ETL approaches based on MR paradigm as all other approaches does not consider the parallelization/distribution at an ETL functionality.

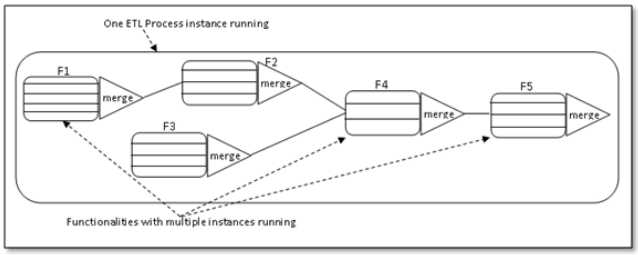


Fig. 3: VDF Approach.

3.1.3 Vertical Distribution of Functionaltilies and Process (VDFP)

In case where VDF presents low performance (particularly if the ETL process contains much sequential functionalities), the designer should set the ETL process to be run in several instances. This is an hybrid approach that takes the principles of the "distributed ETL process" and VDF approaches at the same time as shown in Fig. 4.

It should be noted that the VDFP approach requires more resources (*cluster nodes, HDD space, RAM, Cache, LAN bandwith* ...). When the ETL process runs in the VDF approach and reaches *F4*, it will require three tasks as *F4* runs in three instances. The same process executed in the VDFP approach will require thirty parallel tasks if it runs in ten instances in addition to the three instances of *F4*.

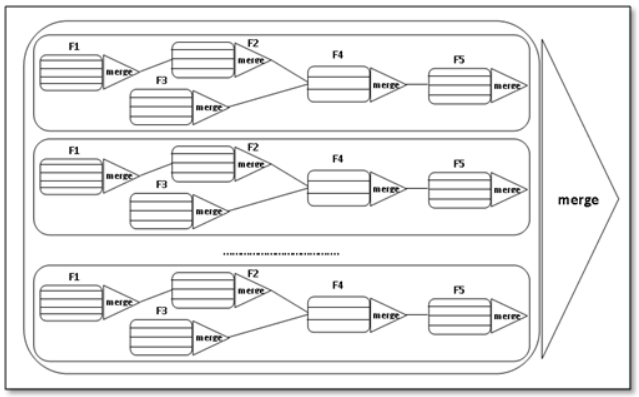


Fig. 4: VDFP Approach.

3.1.4 Horizontal Distribution of Functionaltilies (HDF)

Some ETL functionalities operate several elementary processes on source data. In this case, these functionalities are not atomic and can thereby be decomposed into elementary functions where each one is in charge of a particular process unit. Let F be a functionality in an ETL process which operates some elementary processes units $T_1, T_2, ..., T_n$ on the source data.

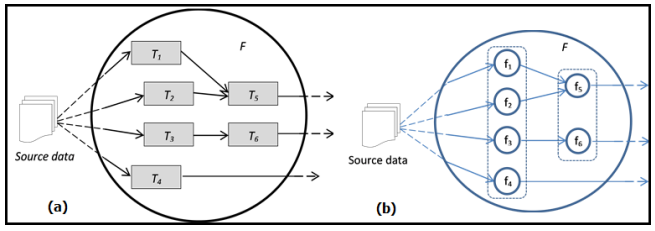


Fig. 5: Elementary processes (a) and functionalities (b).

We can decompose F into elementary functions noted $f_1, f_2, ..., f_n$ where each f_i is in charge of T_i. FIG. 5 (a) shows an ETL functionaly F which operates six elementary processes $T_1, T_2, ..., T_6$. We note that T_1, T_2, T_3, T_4 can be run in parallel way since no dependencies exist between them. In the same way, T_5 and T_6 can be, also, run in parallel. Thus, we can decompose F into six elementary functions noted $f_1, f_2, ..., f_6$ which are in charge of $T_1, T_2, ..., T_6$ respectively (FIG. 5 (b)). Unlike the VDF approach which distributes the ETL functionality by instanciation, the HDF approach distributes the ETL functionality by fragmentation. In a distributed environment, the schema depicted in FIG. 5 (b) allows, in a first phase, distributing F in four fragments f_1, f_2, f_3 and f_4 which run in parallel way. In the second phase, F is distributed into f_4 and f_5 that can be run in parallel way and allow providing the final output of F.

3.1.5 Pipeline Processing Distribution (PPD)

Some ETL functionalities process the source data tuple-by-tuple in a sequential way called pipeline processing. Since all the tuples pass by the pipeline, we propose a synchronization schema in order to process a subset of tuples in parallel way. The number of tuples present in the pipeline should be equal to the number of functionalities. Indeed, when a particular tuple is being processed by the last functionality in the pipe, its successors should be also processed according to the order of the functionalities as defined in the pipe. Let P be a pipe in which is defined a set of functionalities F_1, F_2, ..., F_n.

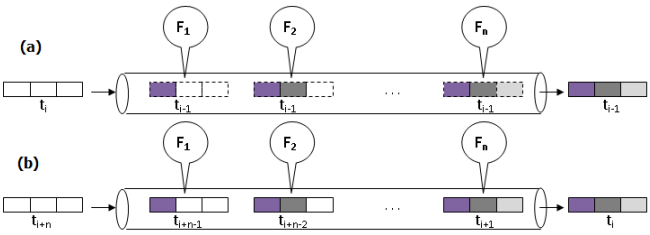


Fig. 6: Sequential (a) and parallel (b) pipeline.

When the tuples of data $t_1, t_2, ..., t_m$ should pass by a sequential pipe P, the tuple t_i is moved in the pipe P when the tuple t_{i-1} is completely processed by all the functionalities F_1, F_2, ..., F_n (FIG. 6 (a)). Thus, only one tuple can be present in the pipe at the same time. In order to improve further the performance of the ETL process, we propose to parallelize the pipe. In this way, several tuples can be processed simultaneously in the pipe where each one is handled by a functionality according to the order defined in the pipe. Thus, when the tuple t_i is being processed by F_n, the tuple t_{i+1} is processed, in the same time, by F_{n-1}, the tuple t_{i+2} is processed by F_{n-2} and so on. In this way, the number of tuples being processed in the pipe is equal to the number of functionalities defined in the pipe (equal to n). Indeed, when a tuple is moved out the pipe after a complete process, another tuple (first in the queue) is moved in and so on until a complete process of the data partition. FIG. 6 (b) depicts the pipe P in the parallel approach.

3.2 Changing Data Capture (CDC) in Big-ETL approach

Our Big-ETL approach is applied on many ETL functionalities such as CDC, SCD, SKP, etc. Seeing the paper space constraint, we illustrate Big-ETL with the CDC functionality. The ETL functionality CDC is considered as the main functionality in the E step of ETL. It identifies the data affected by changes (INSERT, UPDATE, DELETE) in the source systems. These latter is then extracted and processed for the DW refresh [18]. The rest of data (unaffected by changes) is rejected since it is already loaded in the DW. The most common technique used in this field is based on snapshots [18]. In the classical algorithms of CDC, the changes between two corresponding tuples are detected by comparing them attribute-by-attribute. Furthermore, tuples

contain hundreds of attributes in data warehousing systems. In order to improve the CDC performance and make its cost lower, we adapted the well-known hash function CRC (Cyclic Redundancy Check) which is widely used in digital data transmission field [19] and internet applications [20]. We adapted CRC function in the CDC context as follows. Let *tuple1* and *tuple2* be two tuples stored in *ST* and *STpv* respectively. If *tuple1* and *tuple2* satisfy the two equations 1 and 2, it means that they are similar. In this case, the tuple1 will be rejected by the CDC process as no changes have occurred. However, if only the equation 1 is satisfied, it means that *tuple1* has been affected by changes and will be extracted by the CDC process as UPDATE.

$$tuple1.KEY = tuple2.KEY \quad (1)$$

$$CRC(tuple1) = CRC(tuple2) \quad (2)$$

To propose a CDC schema in the Big-ETL environment, we consider that both *ST* and *STpv* tables contain large data, the CDC functionality will run on a cluster of computers, and we adopt the MR paradigm. The classical scheme of CDC will be supplemented by new aspects namely (i) data partitioning, (ii) lookup tables, (iii) insert and update data capture process and (iv) delete data capture process.

3.2.1 Data partitioning

To deal with large data, we adopt the rule of "divide and conquer". In the context of CDC, the system should, firstly, sort *ST* and *STpv* on the column KEY, and then split them to obtain usual volumes of data. The partitioning of *ST* allows processing the generated partitions in parallel way. *STpv* is partitioned to avoid searching *ST* tuples in a large volume of data.

3.2.2 Lookup tables

To avoid searching a tuple in all *ST* and *STpv* partitions, we use Lookup tables denoted *LookupST* and *LookupSTpv* respectively. They identify the partition that will contain a given tuple. Here, are some details on the use of lookup tables:

- *LookupST* and *LookupSTpv* contain the min and max values of keys (*#KEY*) for each *ST* and *STpv* partitions respectively;
- For a T_i tuple in *ST*, it consists of searching the $Pstpv_k$ partition of *STpv* that satisfies the expression 3 in *LookupSTpv*;
- For a T_j tuple in *STpv*, it consists of searching the Pst_k partition of *ST* that satisfies the expression 4 in *LookupST*.

$$LookupSTpv.KEYmin \leq T_i.KEY \leq LookupSTpv.KEYmax \quad (3)$$

$$LookupST.KEYmin \leq T_j.KEY \leq LookupST.KEYmax \quad (4)$$

3.2.3 INSERT-UPDATE data capture (IUDCP) and DELETE data capture (DDCP) Processes

We propose two parallel processes in the new CDC scheme that support (i) INSERT and UPDATE data capture (*IUDCP*), and (ii) DELETE data capture (*DDCP*).

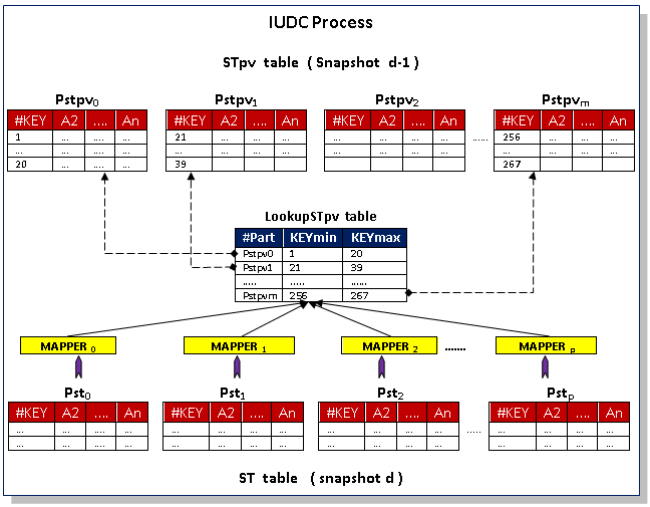


Fig. 7: IUDC process architecture.

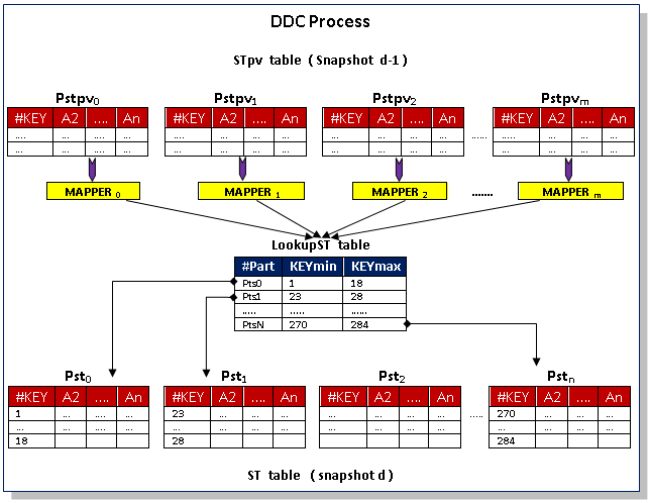


Fig. 8: DDC process architecture.

Each process runs into multiple parallel instances. Each instance of *IUDCP* and *DDCP* handles *ST* and *STpv* partition respectively. Fig. 7 depicts *IUDCP*. Each partition Pst_i is assigned to a Map_i task which is responsible for checking the existence of each its partition tuples in *STpv*. To this end, the mapper looks up in *LookupSTpv* the partition $Pstpv_k$ that may contain the tuple. Once the $Pstpv_k$ partition is identified, three cases can arise: (1) #KEY value is nonexistent in $Pstpv_k$; this means an insertion (INSERT), (2) #KEY value exists and identifies a similar copy of the tuple in $Pstpv_k$; the tuple is rejected (3) #KEY value exists in $Pstpv_k$ with

change in at least one attribute between the two tuples; this is a modification (UPDATE).

Algorithm 1 IU_MAP(Pst)

Input: Pst, LookupSTpv, tuple1: ST record, tuple2: STpv record
Output: CHANGES

```
1:  while not eof (Pst) do
2:    read(Pst, tuple1)
3:    Pstpv ← lookup(LookupSTpv, tuple1.KEY);
4:    if found() then
5:      tuple2 ← lookup(Pstpv, tuple1.KEY);
6:      if found() then
7:        if CRC(tuple1) ≠ CRC(tuple2) then
8:          extracting tuple1 as UPDATE;
9:        end if
10:     else
11:       extracting tuple1 as INSERT;
12:     end if
13:   else
14:     extracting tuple1 as INSERT;
15:   end if
16: end while
17: return (CHANGES);
```

As shown in Fig. 8, DDCP operates on the same principle as IUDCP but in the opposite direction. In DDCP, we focus exclusively on the case where the tuple does not exist (DELETE). In order to have an approach about how to process the mixing of these multiple operations, we propose a main program of CDC called *CDC_BigData*. At this level, the *ST* and *STpv* tables are sorted and then partitioned, the *LookupST* and *LookupSTpv* tables are generated respectively from *ST* and *STpv*, and finally the parallel *IUDCP* and *DDCP* processes are invoked. Algorithm 1 is responsible for capturing insertions and updates in *ST* table. A Pst_i partition will be processed by an instance of iu_map () function. Line 3 looks up in $LookupSTpv$ for a $Pstpv$ partition which may contain the tuple readed in line 2. Lines 4-12 describe the case where the $Pstpv_k$ is located. Line 5 looks-up in Pstpv the tuple. Lines 6-9 treat the case of tuple affected by changes (UPDATE) by invoking CRC hash function. Lines 10-12 treat the case where the tuple does not exist in the partition $Pstpv_k$ and is thereby captured as an insert. Lines 13-15 treat the case where the tuple does not match with any partition in the lookup table $LookupSTpv$ and thereby it is captured as an insert.

4. Implementation and experiment

We developed an ETL platform called P-ETL (Parallel-ETL) which provides: (i) data distribution, and (ii) parallel and distributed ETL processing. P-ETL is implemented in Apache Hadoop environment and uses mainly two modules (1) HDFS for distributed store and high-throughput access to application data, and (2) MapReduce for parallel processing. We defined two levels for our experiments: (i) ETL process level (coarse granularity level) and (ii) ETL Functionality level (fine granularity level). We present in this section the results for the first scenario. To evaluate our P-ETL platform, we proposed an ETL process example applied on students' data gathered at the Education Ministry. The data source contains student identifier (*St_id*), his enrollment date (*Enr_Date*), his cycle (*Bachelor, Master or Ph.D.*), his specialty (*medicine, biology, computer , ...*) and finally we find information about *scholarship* (if the student received *scholarship* or not) and about *sport* (if he practices *sport* or not). We developed a program to generate csv source data. In this experiment, we generated 7 samples of source data that vary between $244 * 10^6$ and $2,44 * 10^9$ tuples where each one has 44 bytes of size. The ETL process configured to process the data is as follows. The first task is *projection* which restricts the source tuples to an attributes subset by excluding *Scholarship* and *Sport*. The process presents in the second task a *restriction* which filters tuples and rejects those having *Null* value in *Enr_Date*, *Cycle*, and *Specialty*. The third task in the process is *GetDate()* which retrieves year from *Enr_Date*. The last task is the aggregation function *COUNT()* which computes the number of students grouped by *enrolment year*, *Cycle* and *Specialty*.

We considered the P-ETL scalability by varying the "data source size" and the "number of tasks". The test environment is a cluster made up of 10 machines (nodes). Each machine has an intel-Core i5-2500 CPU@3.30 GHZ x 4 processor with 4GB RAM, 20 GB of free HDD space. These machines operate with Ubuntu-12.10 and are interconnected by a switched Ethernet 100 Mbps in a LAN. The framework *Apache Hadoop 1.2.0* is installed on all the machines. One of these 10 machines is configured to perform the role of Namenode in the HDFS system and JobTracker in the MapReduce system. However, the other machines are configured to be HDFS DataNodes and TaskTrackers. Overall, we can see, in FIG. 9, that the increasing of tasks improves the processing time. Indeed, we further analyzed the results and discovered some interesting aspects. Seeing the paper length constraint, we can not present all the experiment results.

FIG. 10 shows the "time saving" by increasing tasks. The "time saving" is calculated as the difference between "processing time" corresponding to different "number of tasks". We can see that the time saving to handle $2,2 * 10^9$ tuples (FIG. 10 (a)) decreases when we configure more than "5 tasks". Also, to handle $2,44 * 10^9$ tuples (FIG. 10 (b)), the time saving after "8 tasks" becomes not significant. To sum up our experiment, we note that the "number of tasks" is not the only parameter to be set in order to speed-up the process. Our cluster must be extended in terms of nodes, memory

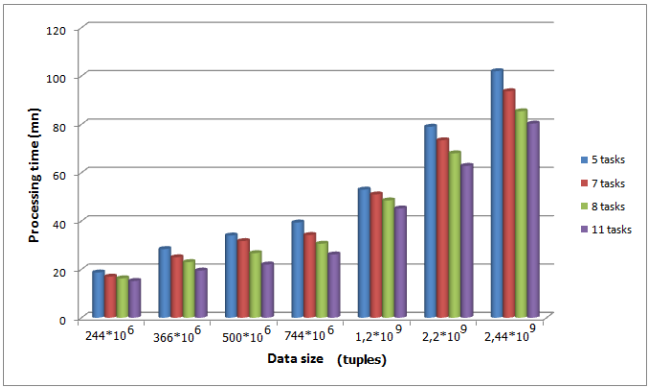


Fig. 9: Proc. time (min.) by scaling up data (tuples) and increasing tasks.

space (RAM, cache), LAN bandwidth, etc. The cluster used for this experiment is a small-sized infrastructure. The HDD space is very low (20 GB per node). Thus, trying to increase tasks, for example, to more than eight while staying on the same resources in terms of HDD, RAM ..., will not make Handoop able to improve performance of the process if HDD space or memory is, already, completely consumed by the eight tasks.

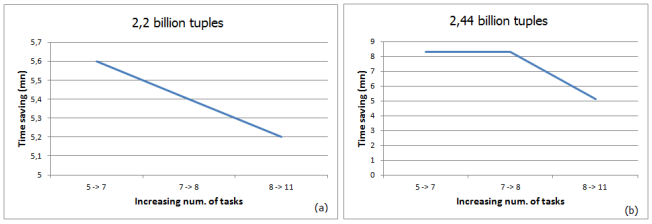


Fig. 10: Time saving (min.) by increasing tasks.

5. Conclusion

The ETL is the core component of decision-support system since all the data dedicated for analysis pass through this process. It should be adapted following the new approaches and paradigms to cope with big data. In this context, we proposed a parallel/distributed approach for ETL process where its functionalities run in parallel way with MR paradigm. In the near future, we plan to finish our experiments on a larger scale both in ETL process level and ETL functionality level. A complete benchmark in which we compare the four approaches (centralized ETL process approach, distributed ETL process approach, Big-ETL approach, Hybrid approach) is an interesting perspective.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] S. Mohanty, M. Jagadeesh, and H. Srivatsa, *Big Data Imperatives: Enterprise Big Data Warehouse,BI Implementations and Analytics*. Apress, 2013.

[3] B. Sosinsky, *Cloud computing bible*. John Wiley & Sons, 2010, vol. 762.

[4] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[5] J. Han, E. Haihong, G. Le, and J. Du, "Survey on nosql database," in *6th international conference on pervasive computing and applications (ICPCA), 2011*. IEEE, 2011, pp. 363–366.

[6] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual modeling for etl processes," in *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*. ACM, 2002, pp. 14–21.

[7] J. Trujillo and S. Luján-Mora, "A uml based approach for modeling etl processes in data warehouses," in *Conceptual Modeling-ER 2003*. Springer, 2003, pp. 307–320.

[8] Z. El Akkaoui and E. Zimányi, "Defining etl worfklows using bpmn and bpel," in *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*. ACM, 2009, pp. 41–48.

[9] Z. El Akkaoui, E. Zimànyi, J.-N. Mazón, and J. Trujillo, "A model-driven framework for etl process development," in *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*. ACM, 2011, pp. 45–52.

[10] B. Oliveira and O. Belo, "Using reo on etl conceptual modelling: a first approach," in *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*. ACM, 2013, pp. 55–60.

[11] F. Arbab, "Reo: a channel-based coordination model for component composition," *Mathematical Structures in Computer Science*, vol. 14, no. 3, pp. 329–366, 2004.

[12] X. Liu, C. Thomsen, and T. B. Pedersen, "Etlmr: a highly scalable dimensional etl framework based on mapreduce," in *Data Warehousing and Knowledge Discovery*. Springer, 2011, pp. 96–111.

[13] C. Thomsen and T. Bach Pedersen, "pygrametl: A powerful programming framework for extract-transform-load programmers," in *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*. ACM, 2009, pp. 49–56.

[14] X. Liu, C. Thomsen, and T. B. Pedersen, "Mapreduce-based dimensional etl made easy," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1882–1885, 2012.

[15] S. Misra, S. K. Saha, and C. Mazumdar, "Performance comparison of hadoop based tools with commercial etl tools–a case study," in *Big Data Analytics*. Springer, 2013, pp. 176–184.

[16] X. Liu, C. Thomsen, and T. B. Pedersen, "Cloudetl: scalable dimensional etl for hive," in *Proceedings of the 18th International Database Engineering & Applications Symposium*. ACM, 2014, pp. 195–206.

[17] M. Bala, O. Mokeddem, O. Boussaid, and Z. Alimazighi, "Une plateforme etl parallèle et distribuée pour l´intégration de données massives," *Revue des Nouvelles Technologies de l'Information*, vol. Extraction et Gestion des Connaissances, RNTI-E-28, pp. 455–460, 2015.

[18] R. Kimball and J. Caserta, *The data warehouse ETL toolkit*. John Wiley & Sons, 2004.

[19] D. V. Sarwate, "Computation of cyclic redundancy checks via table look-up," *Communications of the ACM*, vol. 31, no. 8, pp. 1008–1013, 1988.

[20] M. P. Freivald, A. C. Noble, and M. S. Richards, "Change-detection tool indicating degree and location of change of internet documents by comparison of cyclic-redundancy-check (crc) signatures," Apr. 27 1999, uS Patent 5,898,836.

Using the column oriented NoSQL model for implementing big data warehouses

Khaled. Dehdouh[1], **Fadila. Bentayeb**[1], **Omar. Boussaid**[1] , **and Nadia Kabachi**[1]
[1]ERIC Laboratory/ University of Lyon 2, Bron, France

Abstract - *The column-oriented NoSQL (Not Only SQL) model provides for big data the most suitable model to the data warehouse and the structure of multidimensional data as the OLAP cube and allows it to be deployed in the cloud and a high scalability whilst delivering high performance. In the absence of a clear approach which allows the implementation of data warehouses using this model, we propose in this paper, three approaches which allow big data warehouses to be implemented under the column oriented NoSQL DBMS. Each one differs in terms of structure and the attribute types used when mapping the conceptual model into logical model is performed. We use these approaches to instantiate the conceptual model of the star schema benchmark (SSB) data warehouse into columnar logical models, and show the differences between them when decisional queries are performed.*

Keywords: Big data warehouses, columnar NoSQL model, logical modeling.

1 Introduction

A data warehouse is a database for online analytical processing (OLAP) to aid decision-making. It is designed according to a dimensional modelling which has for objective to observe facts through measures, also called indicators, according to the dimensions that represent the analysis axes [1]. Thus, at the conceptual level, the multidimensional modelling gave birth to the concepts of fact and dimension. The most popular models used to design data warehouses are the star, snowflake, and constellation schemas [2]. These models are then converted to the logical models which depend on the storage mode that will be adapted at the physical level [3]. Classically, the mapping from the conceptual to the logical model is made according to three approaches; ROLAP (Relational-OLAP), MOLAP (Multidimensional-OLAP) and HOLAP (Hybrid-OLAP) [4].

However, with the advent of the big data, the logical modelling adopted by these approaches does not adapt itself to an environment characterized by such amount of data. To solve a part of this issue, other models have appeared such as the column oriented NoSQL. This latter gives a data structure more adequate to the massive data warehouses. Yet, the data warehouse implementation process requires to take into account the recent data structures and should adapt itself to the new technological constraints.

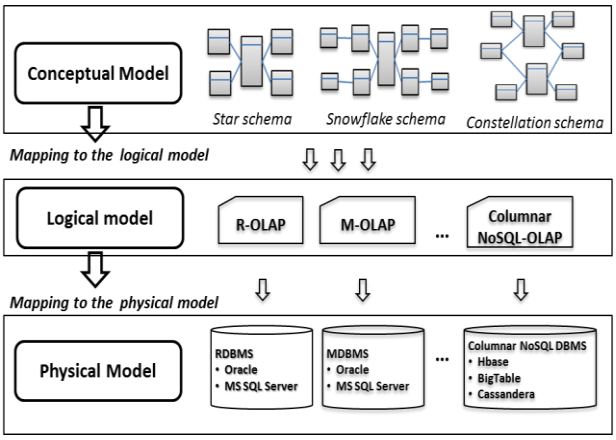


Figure 1: Implementation process for data warehouses

As depicted in the figure 1, the logical model aims at reorganizing the data according to the most appropriate storage architecture for a better taking in charge by the DBMS. It is situated between the conceptual and the physical models of data. In other words , it not only gives more details than the conceptual model on the structuring of data and their relations, but it prepares the transition to the physical level as well; this makes it the most decisive model in the modeling process.

In order to fully benefit from the columnar NoSQL model advantages, and in the absence of a clear approach allowing for implementing the columnar NoSQL data warehouses, we propose in this paper, three types of the conceptual model translations at logical columnar model level namely NLA (Normalized Logical Approach), DLA (Denormalized Logical Approach), and DLA-CF (Denormalized Logical Approach by using Column Family). Each approach leads to a different logical model. We describe each one, and show how we can use these models for implementing data warehouses.

To compare between the three translations, we have implemented the SSB (star schema benchmark) data warehouse [5] according to our propositions. This implementation was achieved under HBase which is column-oriented NoSQL DBMS. We have noticed that the execution of decisional queries using the SSB data warehouse with the

denormalized approaches (DLA and DLA-CF) takes three times less compared with an implementation with SSB using the normalized approach (NLA). Besides, the use of the family column structure for gathering the attributes belonging to the same dimension leads to the improvement until 10% of the queries' execution time which involve several attributes of the same dimension to perform aggregations.

The rest of this paper is organized as follows. Section two gives the related works of column-oriented data warehouse implementation. Section three introduces the column-oriented NoSQL model. Section four presents the three approaches that we propose for data warehouse implementation under the columnar NoSQL DBMS. Section five describes the conversion rules from a dimensional model towards the logical models according to the three approaches NLA, DLA and DLA-CF. In section six, we conducted experiments to evaluate the star schema benchmark implementation according to our three approaches that we propose. Finally, in section 7, we conclude this paper and give some perspectives.

2 Related work

Although the columnar NoSQL model is widely used for storing and analyzing massive data, it does not have, to our knowledge, any methods or defined rules which allow us to know whether a column-oriented NoSQL data warehouse is well performing or not.

However, some works are aimed at developing data warehouses in columnar NoSQL DBMS. In [6] [7], the author has proposed an approach for transforming a relational database into a column oriented NoSQL database using HBase. However, this approach is limited to the logical level, and does not consider the conceptual model of data warehouses; i.e.: mapping a relational logical model into a column oriented logical model. In recent work [8], we have developed a new benchmark for the columnar NoSQL data warehouse, but without giving the formalization for the modeling process. However, this work is considered as the first work which proposes implemented star data warehouse under column oriented NoSQL DBMS directly from dimensional model.

Another recent work, based on our benchmark, has tried to define a logical model for NoSQL data stores (oriented columns and oriented documents) [9]. However, its column oriented logical modeling has been only limited to the use the columns family concept without considering the attributes which are not necessarily belonging to a column family. Indeed, the columns oriented NoSQL model proposes two kinds of attributes: a simple attribute and a composite attribute (nested attribute). This latter is represented by the concept of column family (see section 3). Thus, columnar NoSQL DBMS rather favors the denormalization of the dimensional model, without necessarily using the column families (composite attribute).

To complete these works by taking the simple attributes case into a count, we propose three candidate approaches which summarize the mapping of the multidimensional conceptual data model into a logical modeling adapted to the column-oriented NoSQL data warehouses.

3 Column oriented NoSQL model

In this section, we present the columnar NoSQL model which is characterized by non-relational logical representation of data, and storing the data of a table column-by-column [10]. It allows data warehouse architecture to be deployed in the cloud and a high scalability whilst delivering high performance [11].

Indeed, the non-relational aspect of this model allows the massive data warehouses to be deployed in a distributed environment when scaling up [12], and the column oriented aspect for storing data is beneficial to the data warehouse when aggregation is performed with value belonging to the same column [13]. However, columnar NoSQL model does not have a mechanism which takes in charge the links between tables; thus, it is assigned to the customer applications level [14].

Consequently, the columnar NoSQL DBMS as HBase favors gathering columns in a single table when storing data. Each column stores data in the form of a "key/value" pair which can be stored in a distributed file system. The combination <row key, column name, timestamp> represents the coordinates of the value as depicted in the figure 2.

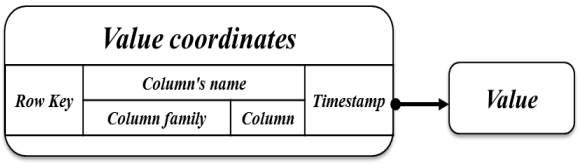


Figure 2: Data structure of the columnar NoSQL model

The row key serves for identifying the column values belonging to the same tuple. The column name allows identifying the attribute of a value; it can be composed by column family name and column name. Indeed, the column may be composite or simple. If the column name is prefixed, this means that the column name consists of the name of the column family (prefix) and the name of the nested column. In this case, it is called composite attribute (belong to a column family), otherwise it is considered as simple attribute. The figure 3 represents the corresponding UML class diagram of a column-oriented NoSQL data model (tables, rows, column families and columns).

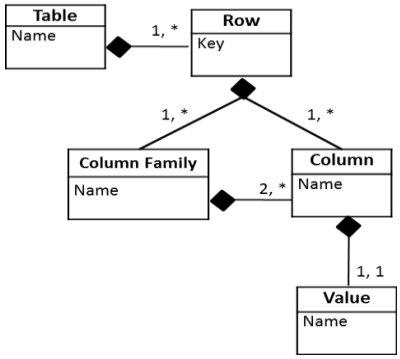


Figure 3: UML class diagram of the concepts of a column-oriented NoSQL data model.

Finally, the timestamp allows checking data coherence. Each value is allocated a timestamp by the system for the purpose of data consistency. It is noteworthy that data replication across different machines (nodes) required by the data management in a distributed environment (Eventually consistent) sometimes leads to different versions of the same data during updates. The timestamp associated with each value means that it is the most recent version which will be taken when a query is entered into the database [15].

Moreover, HBase and Cassandra from the Apache Foundation and BigTable from Google are three examples of column oriented NoSQL DBMS. In the next section, we present a logical model which allows implementing data warehouses under a column-oriented NoSQL DBMS.

4 The logical model for the columnar NoSQL warehouses

In order to implement the big data warehouses within the column-oriented NoSQL model, we propose three approaches namely NLA (Normalized Logical Approach), DLA (Denormalized Logical Approach), and DLA-CF (Denormalized Logical Approach by using Column Family). Each one differs in terms of the structure and the attribute types used when mapping is performed.

The first one uses different tables for storing fact and dimension, and use only the simple attribute for representing measure and dimension attributes. The second approach proposes storing the fact and dimensions into one table, and uses only the simple attribute for representing measure and dimension attributes. The third approach proposes storing the fact and dimensions into one table, and uses only the composite attribute for representing measure and dimension attributes. These approaches are described below.

4.1 Normalized Logical Approach (NLA)

This approach proposes to map the dimensional model of data warehouse by the normalized approach as the relational does. In order to achieve this, the classic dimensional models are converted towards relational logical models. The dimensions and the facts are stored separately on different tables. To ensure the links between these two entities (dimension-fact), the dimension table identifier is duplicated in the fact table. However, without the referential integrity constraints in the columnar NoSQL DBMS, it is the responsibility of the customer application level to check this scheduler.

4.2 Denormalized Logical Approach (DLA)

This approach proposes transforming the data conceptual model into a model based on a large structure (table) called *BigFactTable*, which keeps the facts and the dimensions joined. On the opposite of the normalized approach which separates the facts from their dimensions, this approach favors the denormalization of the schema by integrating, in the same table, the fact and the dimension. This process is very frequent in the modeling of the data warehousing, particularly in the management of the dimensional hierarchies; as explained by [16]. To map measures and dimensions into logical model, this approach uses the simple attribute proposed by the columnar NoSQL model. Thus, the fact and dimension values are now identified by the same identifier (row key) of the table, and we have no longer to achieve joining between tables when aggregation is performed.

4.3 Denormalized Logical Approach by using Column Family (DLA-CF)

This approach proposes transforming the data conceptual model into columnar NoSQL logical model as Denormalized Logical Approach does. However, this approach uses the composite attributes to map the measures and dimensions instead the simple ones. Indeed, each dimension is mapped into a column family and the attributes belonging to the same dimension are gathered in one column family. This allows attributes belonging to a given dimension to be shared in the same disk space which improves the column access time especially when decisional query involves several attributes of the same dimension (hierarchy: year, trimester and month) to perform aggregations.

5 Mapping from the dimensional model to the columnar NoSQL logical models

In order to define the rules that cover the mapping process from the dimensional model to the candidates' columnar NoSQL logical models defined above (section 4), we first formalize the different instantiations that lead to these logical models.

5.1 Formalization

Given data warehouse dimensional model DW composed by the couple (F, D). F represents the set of measures $F = \{M_1, M_2, \dots, M_q\}$, and D represents a set of dimensions $D = \{D_1, D_2, \dots, D_j\}$. Each dimension D grouped a set of attributes represents the axe of analysis used for observing the measure attributes M_q, it is defined by $D_j = \{At_1^j, \dots, At_k^j\}$

Definition 1 (NLA)

According to NLA, the instantiation of DW leads to map the fact *F* and the dimensions *D* to separate tables called respectively *FT* (fact table) and *DT* (dimension table). The simple attribute is used for representing both the dimension and measure attributes; such as, $\exists\ E \subset DT_j : E \to DT_j.At_k^j$, and $\exists\ E' \subset FT : E' \to M_q \wedge E$. It means that for each dimension table, there is at least an attribute which uniquely identify all the attributes of dimension, and there is another one which identify both all measure attributes and the identifiers of dimensions.

Definition 2 (DLA)

The instantiation of DW according to DLA leads to map the fact *F* and the dimensions *D* to the same table called *BigFactTable* (*BFT*). This approach uses the simple attribute for representing both the dimension and measure attributes. Thus, we consider that the logical modeling is performed according to DLA, if and only if there is a sub set of attributes *E* included in *BFT*, such as: $\exists\ E \subset BFT : E \to BFT.At_k^j \wedge BFT.M_q$. It means there is at least an attribute which uniquely identify all the attributes of dimension and all measure attributes in the *BigFactTable*.

Definition 3 (DLA-CF)

According to DLA-CF, the instantiation of DW leads to map the fact *F* and the dimensions *D* to the same table called *BigFactTable* (*BFT*) by using the column family structure *CF*. Indeed, all measure attributes are gathered into a column family, and each dimension is converted to a column family, too. Thus, the dimension attributes which belong to the same dimensions are gathered into a column family. We consider that the logical modeling is performed according to DLA, if and only if there is a sub set of attributes *E* included in *BFT*, such as: $\exists\ E \subset BFT : E \to BFT.CF_j.At_k^j \wedge BFT.CF.M_q$. It means there is at least an attribute which uniquely identify all the attributes of dimension and all measure attributes in the *BigFactTable*.

5.2 Mapping rules from the dimensional model

At the conceptual modeling level, the dimensional model is independent from the details related to data structuring and the environment implementation; hence, we adopt the dimensional model as presented by [14] without any expansion or modification. However, we expose, in this section, the rules which allow to map a dimensional model already established towards a logical model according to the three approaches that we propose in this work.

Conceptual model to NLA: in order to instantiate from the conceptual model by using this approach, the following rules must be applied:

(R1) Each fact becomes a table called fact table *FT*, and each dimension becomes a table called *DT*.

(R2) Each measure $M \in F$ is translated within *FT* as a simple attribute ($i.e\ FT.M$).

(R3) Each dimension *D* and each attribute $At_k \in D$ is mapped into *DT* as a simple attribute (i.e. $DT_j.At_k^j$), and the *FT* is completed by simple attribute $DT_j.At_k^j$ (the value reference of the linked dimension).

Conceptual model to DLA: in order to instantiate from the conceptual model by using this approach, the following rules must be verified:

(R1) Each fact and dimension is converted in one large table called *BigFactTable BFT*.

(R2) Each measure $M \in F$ is translated within *BFT* as a simple attribute ($i.e.: BFT.M$).

(R3) For all dimensions *D*, each attribute $At_k \in D$ is translated into a simple attribute (i.e. $BFT.At_k$).

Conceptual model to DLA-CF: in order to instantiate from the conceptual model by using this approach, the following rules must be checked:

(R1) Each fact and dimension is converted in one table called *BigFactTable BFT* as composite attributes (column families).

(R2) Each measure $M \in F$ is mapped as a simple attribute and included in a column family into a *BigFactTable* ($i.e.: BFT.CF.M$).

(R3) Each dimension D is translated into a composite attribute (i.e. $BFT.CF_j$), and each attribute $At_k^j \in D_j$ is translated as a simple attribute included in the CF_j (i.e. $BFT.CF_j.At_k^j$).

The matching between entities from the conceptual model with those from the logical models that we propose is shown in the following table:

Conceptual model	NLA	DLA	DLA-CF
Fact F	FT	BFT	BFT
Measure M	FT.M	BFT.M	BFT.CF.M
Dimension D	DT	BFT	BFT.CF
Dimension attribute At	DT.At	BFT.At	BFT.CF.At

Table 1: Matching between the conceptual model and the logical models.

6 Experiments

In this section, we have evaluated the performances of the star data warehouse under the columnar NoSQL DBMS. For this reason, we have implemented a decisional benchmark SSB within HBase columnar NoSQL DBMS according to three (3) approaches. The first one implements the SSB following the normalized logical approach NLA; we called this data warehouse NLA-SSB. The second approach denormalizes the schema of data warehouse and implements the SSB according to the denormalized logical model without using the family columns DLA. We called it DLA-SSB. The third and last approach implements the SSB according to the denormalized logical model by using column family DLA-CF. We called it DLA-CF-SSB. To achieve this evaluation, we conducted two experiments to study the impact that the choice of approach used to implement a data warehouse under the column oriented NoSQL DBMS may have on the execution time of the decisional queries.

6.1 Test environment

In order to perform our experiments within a column oriented NoSQL and distribute environment, we have put in place a non-relational and distributed storage and processing environment [17]. This environment is based on a private Cloud Computing architecture produced using the Hadoop-2.6.0 and a HBase-0.98.8 DBMS, for managing data in a distributed environment. In order to simplify data handling and boost the performance of the HBase DBMS, we strengthened this configuration with an SQL interface for HBase, called Phoenix-4.1.0. This latter is an open source and allows the data handling at the HBase level (scan, put and get) to be combined to express a selection of data and to apply filters [18].

The test environment is a cluster made up of 25 machines (nodes). Each machine has an intel-Core TMi5-3220M CPU@3.30 GHZ processor with 8GB RAM. These machines operate with the operating system Ubuntu-14.04 and are interconnected by a switched Ethernet 100 Mbps in a local area network. One of these machines is configured to perform the role of Namenode in the HDFS system, the master and the Zookeper of HBase [19]. However, the other machines are configured to be HDFS DataNodes and the HBase RegionServers. Although the private Cloud architecture we used is limited in terms of capacity (number of nodes composing the cluster), it is sufficient to allow us to deploy a non-relational data warehouse with scaling-up and to apply a queries set in a distributed environment.

6.2 Data set

In order to perform our experiments, we used data generators of SSB which are available according to normalized[1] and denomalized[2] approaches [8], and we populated NLA-SSB, DLA-SSB, and DLA-CF-SSB data warehouses according to SF = 1000, this allows to generate fact table with 6×10^9 tuples of data sample.

6.3 Queries set

For our experiments, we used a queries set composed of eight (8) queries which are divided into two categories as depicted in table 2. The first category is composed of four (4) queries; they gradually increase in the number of dimensions involved when aggregation is performed. Each query in this category uses one attribute per dimension. The second category is composed of four (4) queries in which they involve only one dimension and gradual increase in the number of dimensions attributes when aggregation is performed.

Queries set	Query	Dimension	Attributes	Measure
Category-1	Query 1.1	Date	year,	Sum (revenue)
	Query 1.2	Date, Part	year, category	
	Query 1.3	Date, Part, Supplier	year, category, region	
	Query 1.4	Date, Part, Supplier, Customer	year, category, region (Supplier), region (Customer)	
Category-2	Query 2.1	Part	color	
	Query 2.2		color, type	
	Query 2.3		color, type, size	
	Query 2.4		color, type, size, container	

Table 2**:** Descriptive table of queries set

6.4 Experiment 1

In this experiment, the aim is to study the execution time impact of the normalized and denormalized approaches by

[1] https ://github.com/electrum/ssb-dbgen
[2] https://github.com/Dehdouh/DBGEN-CNSSB

using queries which involve attributes from different dimensions when aggregations are performed. To do this, we applied the first queries set category to NLA-SSB, DLA-SSB, and DLA-CF-SSB data warehouses. The results we obtained are shown in the following figure:

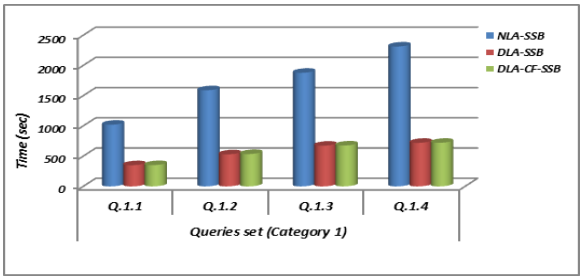


Figure 7: Execution time of the category 1 of queries set

We observed that the denormalized data warehouses represented by DLA-SSB and DLA-CF-SSB show better performance than normalized data warehouse represented by NLA-SSB. Indeed, the query execution times obtained from the data warehouses DLA-SSB and DLA-CF-SSB are better until three times than those executed by the data warehouse NLA-SSB. This is because implementing data warehouse according to normalized approach entails higher costs for materializing the link between dimension and fact especially when the queries involve more joins between the tables for performing aggregations.

However, for denormalized warehouses (DLA-SSB and DLA-CF-SSB), we found that gathering the dimension attributes in a column family does not impact the warehouse performance when the query handles attributes belonging to different dimensions.

6.5 Experiment 2

In this experiment, the aim this time is to study the execution time impact of the normalized and denormalized approaches by using queries which involve only one dimension and gradual increase in the number of dimensions attributes when aggregation is performed. To do this, we applied the second queries set category to DLA-SSB, DLA-CF-SSB, and DLA-CF-SSB data warehouses. The results we obtained are shown in the following figure:

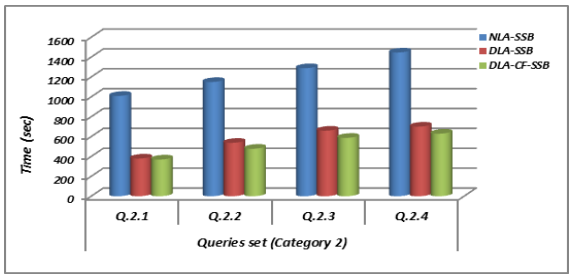


Figure 8: Execution time of the category 2 of queries set

We observed that the query execution times is different for each data warehouse and gives advantage to DLA-CF-SSB. Indeed, the queries used in this experiment (category 2) involve only one dimension when performing aggregations. In the case of NLA-SSB data warehouse, the join between facts table and dimension (Part) is performed. Thus, involving another attribute belonging to the same dimension (Part) entails only additional time related to its scan. This time is lower than the time of joining additional dimension.

On the other hand, we found that DLA-CF-SSB data warehouse performs execution times until 10 % better than DLA-SSB data warehouse. In the context of big data warehouse, this is very important especially in the case of data warehouses characterized by a large number of attributes which compose the dimensions. Indeed, HBase DBMS stores columns by lexicographical order which may sometimes store the columns of the same dimension separately in different disk spaces. Thus, using the column family allows having the attributes belonging to the same dimension stored in the same disk space.

Based on these results, we found that the use of the column family for implementing columnar NoSQL data warehouses gives benefits only with decisional queries handling attributes belonging to the same dimension (i.e.: the dimension hierarchy is involved).

7 Conclusion

Facing the emergence of large and unusual volumes of data (big data), we have proposed three approaches which allow mapping the multidimensional conceptual data model into a logical modeling adapted to the column-oriented NoSQL data warehouses. We have called these approaches; NLA, DLA, and DLA-CF. Each one differs in terms of the structure and the attribute types used when mapping is performed. We have described each one and showed the rules governing the instantiation of the conceptual model. Each approach has its weaknesses and strengths, and the choice depends of the use case.

We have used these approaches for evaluating the performance of SSB data warehouse under distributed environment when applied on decisional queries set. Then, we have observed that the denormalized data warehouses represented by DLA and DLA-CF show better performance than NLA which represents normalized approach. Indeed, the NLA uses less disk memory, but it is quite inefficient when queries with joins are performed.

Morover, we have found that the DLA-CF is more efficient than DLA, but only when query handling attributes belong to the same dimension. Thus, the use of the column family depends of the type of the queries which are applied to the columnar NoSQL data warehouse.

As a perspective, we tend to explore in the next work, the instantiation of data warehouse across other different NoSQL systems namely: key/value, documents-oriented, and graph-oriented to analyze the big data warehouses. These systems give efficient managing of big data corresponding to different contexts.

8 References

[1] Inmon, W. "Building the data warehouse". QED Information Sciences, Inc, 1992.

[2] Kimball, R. "Kimball Dimensional Modeling Techniques", Kimball Group University, 2013.

[3] Coronel, C., Morris, S., Rob, P.: "Database Systems: Design, Implementation, and Management", Cengage Learning, 2012.

[4] Chaudhuri, S., Dayal, U., Ganti, V. "Database technology for decision support systems", IEEE Computer Society, 48--55, 2002.

[5] O'Neil P., O'Neil B., Chen X.: The Star Schema Benchmark (SSB), http://www.cs.umb.edu/\~poneil/ StarSchemaB.PDF, (2009).

[6] Li, C. "Transforming relational database into HBase: A case study", International Conference on Software Engineering and Service Sciences (ICSESS), 683--687, 2010.

[7] Han, D., Stroulia, E. "A three-dimensional data model in hbase for large time-series dataset analysis", IEEE MESOCA, 47--56, 2012.

[8] Dehdouh, K., Boussaid, O., Bentayeb, F. "Columnar NoSQL Star Schema Benchmark", Model and Data Engineering MEDI, 281--288, 2014.

[9] Chevalier, R., El Malki, M., Kopliku, A., Teste, O., Tournier, T. "Implementing Multidimensional Data Warehouses into NoSQL". International Conference on Enterprise Information Systems (ICEIS 2015), 172--183, 2015.

[10] Jing, H., Haihong, E., Guan, L., Jian, D. "Survey on NoSQL database", International Conference on Pervasive Computing and Applications (ICPCA), 363--366, 2011.

[11] Pokorny, J. "Nosql databases: A step to database scalability in web environment", Association for Computing Machinery ACM, 278--283, 2011.

[12] Jerzy, D. "Business Intelligence and NoSQL Databases", Information Systems in Management 1, 25--37, 2012.

[13] Matei, G. "Column-oriented databases, an alternative for analytical environment". Database Systems Journal, 3--16, 2010.

[14] Apache Software Foundation. "The Apache HBase Reference Guide", http://hbase.apache.org/book/joins.html, 2014.

[15] Cattell, R. "Scalable SQL and NoSQL Data Stores", Association for Computing Machinery ACM SIGMOD Record, 12--27, 2011.

[16] Kimball, R., Ross, M. "The data warehouse toolkit: The complete guide to dimensional modeling", Second Edition, Inc, 2002.

[17] Taylor, R. "An overview of the Hadoop-MapReduce-Hbase framework and its current applications in bioinformatics". BMC Bioinformatics Journal. 2010.

[18] James, T. https://github.com/forcedotcom/phoenix/wiki /Performance, 2013.

[19] Hunt, P., Konar, M., Junqueira, F. P., Reed, B. "Zookeeper: Wait-free Coordination for Internet-scale Systems", Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, 11--24, 2010.

Big Data Analytics and Spatial Common Data Model Role

Ayman Ahmed Sami[a]

a) Senior GIS Analyst Engineer Openware (Kuwait Oil Company)

Abstract - *Big data analytics in terms of business perspective is the way to extract and derive new information based on analytical steps for the current raw data. Integration process to create the business logic model is one of the main challenges in this step, which is required to view raw data in new business objective dimension. This paper drills into the role of spatial common data model (SCDM) in the integration process between various workflows to derive unified data logic layer as big data analytics foundation. Through the case study, SCDM is capable to manage and analyse the basic big data dimensions (volume, velocity and variability).*

Keywords: Data Analytics, Spatial common data model, spatial risk model, quality performance index.

1 Introduction

Margaret Rouse says that data analytics (DA) is the science of examining raw data with the purpose of drawing conclusions about that information. Thus, raw data has to be captured in unified framework or environment to be able to analyse and manage the interaction between raw data to be able to represent it in the new business objective dimension. Integration process is responsible for interaction and communication between the various workflows to derive the target data logic layer. Ayman Sami shows how to deploy and implement spatial common data model (SCDM) in geographic information system (GIS) to act the domain model for the various workflows in the enterprise environment. SCDM is based on analysing the decomposed activities or workgroups for common processes and providing the spatial integrity or correlation between the decomposed activities, which could be established in GIS environment. The following case studies utilize this concept or methodology to develop the unified geo framework to derive decision-making information based on the integration between the current and heterogonous workflows in enterprise oil and gas industry.

2 SCDM and spatial management of emergency response plan

This case drills into the development of spatial risk model which would act as SCDM to be able to spatially manage the various modules of the emergency response plan for both preventive and response actions in oil and gas industry based on the H2S dispersion model.

The scope is to create dispersion risk model for H2S derived from EUB (Energy and Utility Board) dispersion calculation model. Then, calculated EPZ (Emergency Planning Zones) spatially for sour wells based on phase operation is implemented. Wind magnitude and direction impact in case of calculated protective action zone are considered.

Capability of creating actual EPZ zones is based on geospatial analysis relationships. These zones are recognized with respect to the available geographic objects. For example, these objects can be the access routes and their availability.

Implement the spatial risk assessment score matrix per sour well to enable the spatial management of ERP type related to the well based on its status.
Proposed spatial data model for ERP management based on the spatial risk model:

The proposed spatial ERP data model is based on the developed spatial risk model derived from the defined spatial and non-spatial probable parameters as well as the related consequence analysis for H2S dispersion through which the emergency level could be specified as well as related ERP modules. Different factors related to the spatial situation in question is being taking care off. Figure 3 shows proposed flow chart of spatial ERP management based on EPZ consequence analytical zones.

Customized spatial risk model using python script in ESRI environment:

The data needed are being gathered. These data are being put in the proper form for ESRI computer software environment. Figure 4 shows the customized GIS model used to extract the main output parameters from the calculated EUB (Energy and Utility Board) for H2S calculation model. Such parameters are the calculated EPZ (Emergency Planning Zone), PAZ (Protective Action Zone), IIZ (Initial Isolation Zone) distances, H2S concentration, wind magnitude, phase operation, and other mandatory parameters to be accounted for in the spatial risk model.

Create spatial risk model:

This model will indicate the amount of risk involved depending on the probability analysis needed. The spatial EPZ, PAZ, IIZ per analytical asset or element (sour well) is being created. Then, the spatial risk model can be developed based on EPZ consequence analysis and the defined impact parameters.

Figure 5 shows the initial output of the spatial risk H2S dispersion model based on the calculated EPZ zones from EUB for H2S dispersion model, as well as the creation of initial spatial risk model.

The final resized EPZ, PAZ, IIZ with the modified spatial risk model:

The final spatial model will detect and adapt to changes. Figure 6 shows how the spatial model is intelligent enough to detect any change in the defined risk parameters whether spatial, environmental (e.g., wind direction and magnitude from sensors, etc.) or non spatial risk parameters (e.g., uncontrolled flow, etc.) and adapt accordingly.

3 SCDM and pipeline asset management integrity

This case drills into the utilization of APDM (ArcGIS of Pipeline Data Model). ADPM is GIS template derived from PODS (Pipeline Open Data Standard) which enables GIS specialist to be able to easily manage the basic elements for pipeline asset management integrity in GIS environment. We rely here on the spatial integrity and relationship between 3 main basic elements which are : control points , station series and risk analysis layers to develop SCDM based on the derived spatial risk model from the mentioned relationship.

Figure 7 shows GIS QPI (Quality Performance Indicator) surface created based on the polynomial relation between the risk and the quality.

Figure 8 shows how SCDM can be utilized to detect the least QPI areas and detect the main events with its related consequence which lead to low QPI .This could be achieved through the spatial analysis and management of the integration process between the heterogonous workflows in pipeline asset management.

4 Equations for Spatial Risk Score Matrix Development and ERP Spatial Management

1) Refined EPZ radius = EPZ calculated (1 + Related consequence value). [1]

The actual EPZ calculated based on the impact of spatial and non-spatial parameters on the consequence matrix.

2) (Total Of Risk) TOR = (Probability * Consequence) per asset or analytical element. [1] [2].

The total of risk score per asset (object). This value is inserted into the risk score matrix for risk evaluation.

3) $$f(x)=\frac{1}{\sqrt{2\pi\sigma}}e^{-(x-\mu)^2/2\sigma^2}$$. [4]

Binomial distribution equation: that represents the probability relationship between the quality performance and related risk. This equation is used for the calculation of quality performance indicator of the target plan.

where $$\mu=\frac{TOR}{2}$$,

and $$\sigma=\sqrt{\frac{TOR^2}{4}}$$

X = 1 (Assumed Random Variable)

Scale Factor = 100 (Assumed)

4) QPI (Quality Performance Index) [3]

$$= f(x) \times 100$$

QPI of target plan is calculated assuming that the scale factor is 100.

5 Figures

4	8	12	16
3	6	9	12
2	4	6	8
1	2	3	4

Figure 1: The proposed total of risk score matrix for spatial ERP

Figure 2: Phases identified in a typical emergency management process

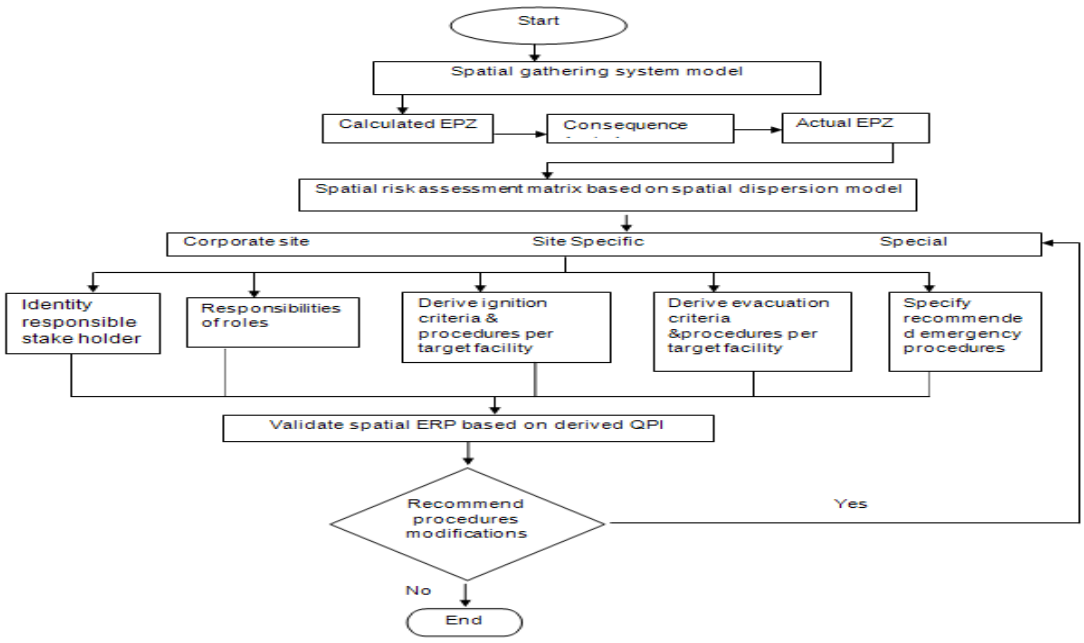
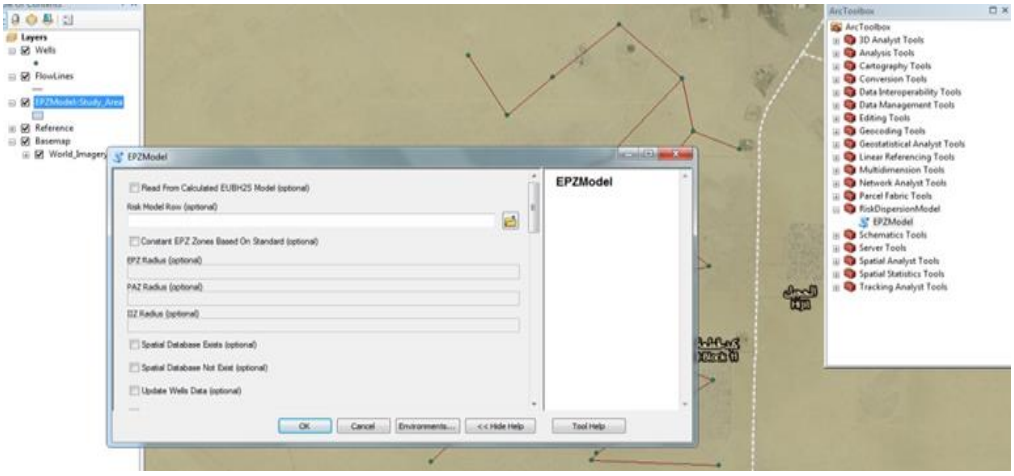


Figure 3 Proposed flow chart of spatial ERP management based on EPZ consequence analytical zones.



Fogure 4 The model interface of spatial H2S dispersion model.

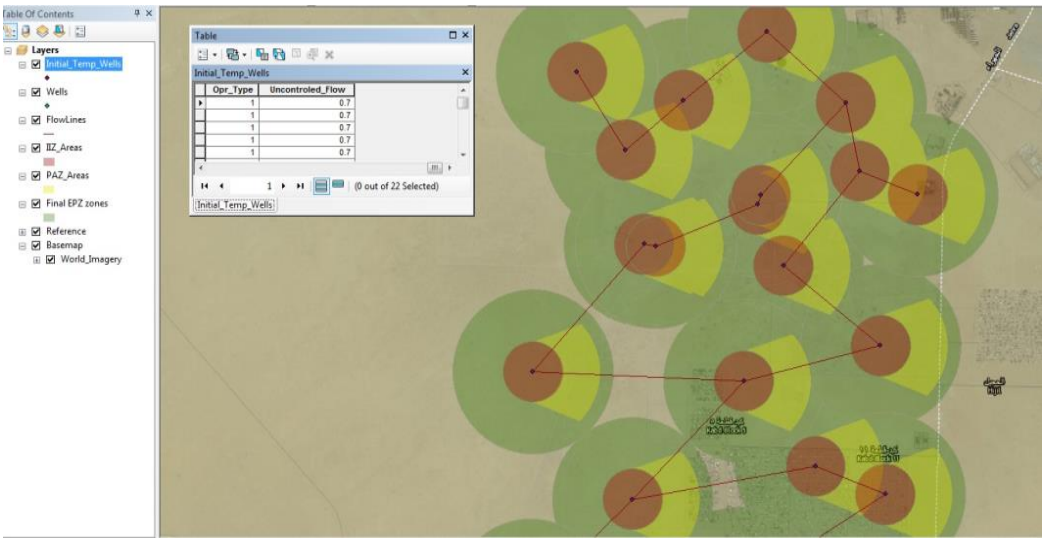


Figure 5 The calculated spatial EPZ zones.

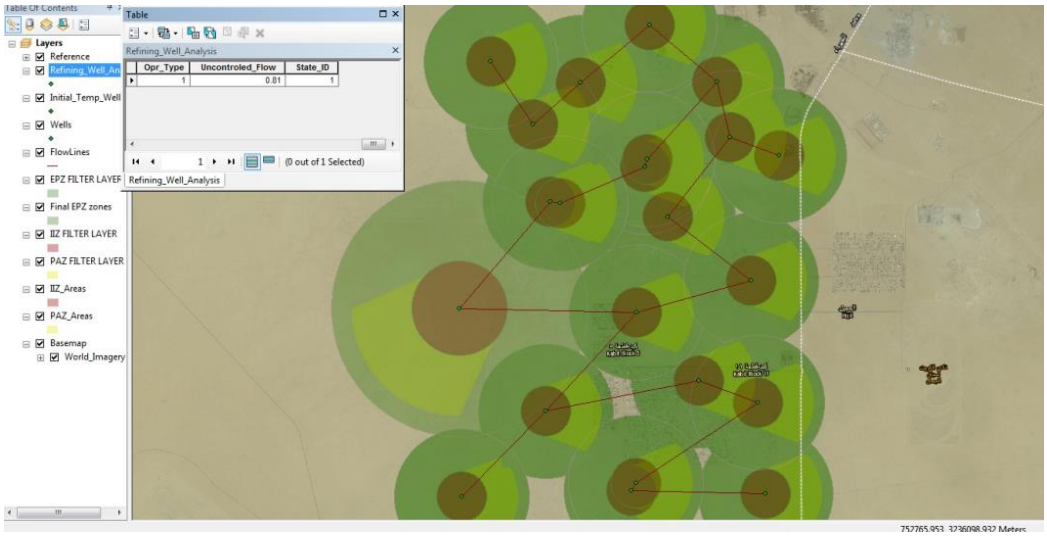


Figure 6 The refined spatial EPZ zones.

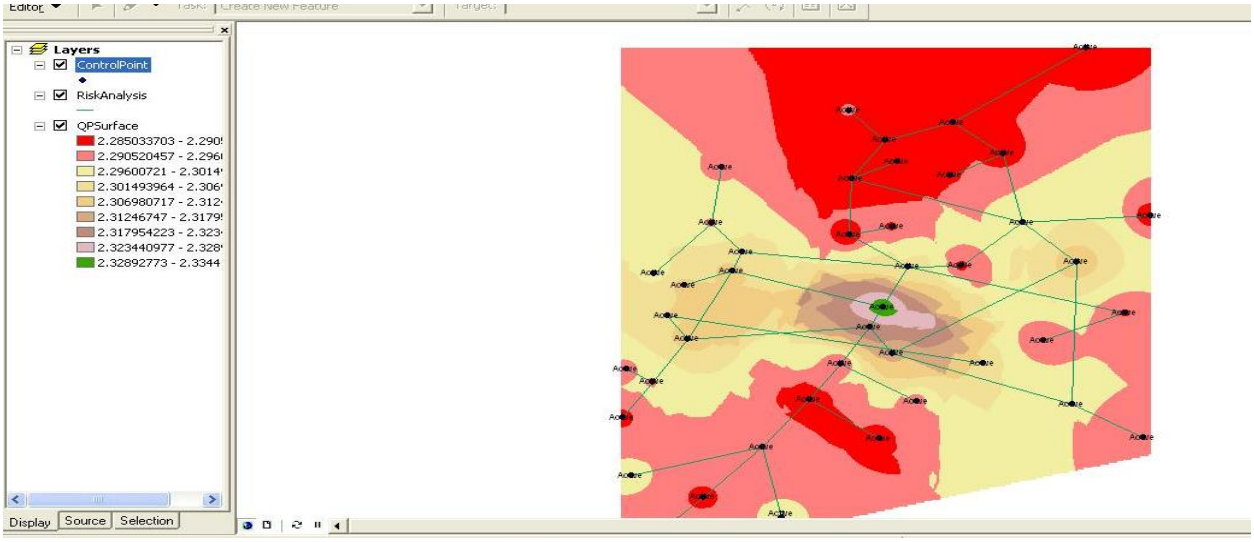


Figure 7 Spaial interpolated QPI surface

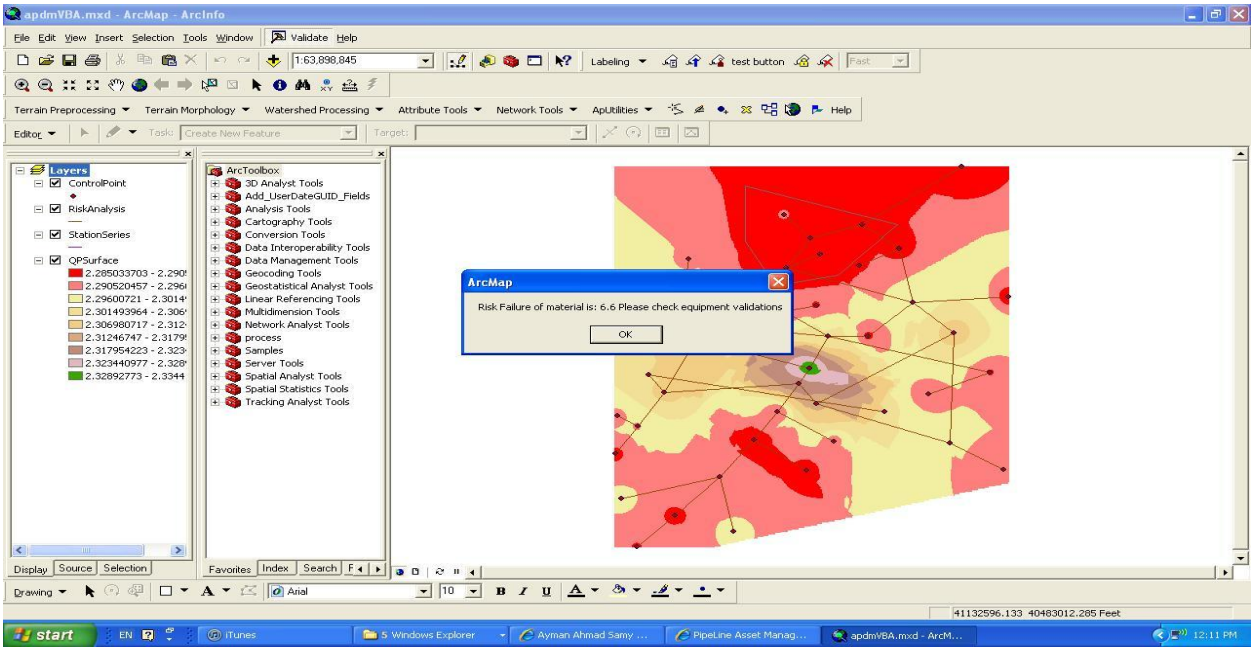


Figure 8 Rquired information for the Low quality cause based on spatial analytical of data sources

6 References

[1] "British Columbia Oil and Gas Handbook Emergency Planning and Requirements for Sour Wells,". Technical guideline documentation from BC Oil and Gas Commission.

[2] The Canadian Association of Petroleum Producers (CAPP), "CAPP Companion Planning Guide to ERCB Directive 071,". Technical guideline documentation from the Canadian Association of Petroleum Producers (CAPP).

[3] A. Sami, "Role Of Geographic Information System For Asset Management Information Risk Assessment For Pipelines Utility In Oil And Gas Industry," ESRI, UC, July 2012.

[4] A. Sami, "Spatial ERP Management for oil and gas," IARIA,Geoprocessing, Lisbon February 2015.

.

SESSION

DISTRIBUTED PROCESSING, MOBILE COMPUTING, AD HOC NETWORKS, AND WIRELESS SENSOR NETWORKS

Chair(s)

TBA

On Minimizing Broadcast Latency in Duty-cycled Wireless Sensor Networks

Duc-Tai Le[1], Thang Le Duc[1], Vyacheslav V. Zalyubovskiy[2], Dongsoo Kim[3] and Hyunseung Choo[1]
[1]Sungkyunkwan University, South Korea
[2]Sobolev Institute of Mathematics, Russia
[3]Indiana University-Purdue University Indianapolis, USA

Abstract— *Minimizing broadcast latency is one of the most important issues for broadcasting in duty-cycled wireless sensor networks. The existing broadcast schemes do not allow any collision in a schedule to ensure its completion, i.e. all nodes receive a broadcast message collision-freely. A delay of transmission caused by the collision-prevention may increase the broadcast latency if the delay occurs along a critical path of the network. In order to minimize broadcast latency, the paper proposes a broadcast scheduling scheme that provides preference to nodes along critical paths of a network. The proposed scheme allows collision at non-critical nodes to speed up the broadcast process for critical ones. It ensures the completion of a broadcast scheduling by retransmission. Simulation results show that the proposed scheme significantly reduces broadcast latency compared with the existing schemes, and slightly increases the number of transmissions due to retransmission.*

Keywords: latency efficiency, collision-tolerant, critical path, broadcast schedule, duty cycle

1. Introduction

Broadcast is a fundamental operation in Wireless Sensor Networks (WSNs) and plays an important role in communication protocol design. Like other operations in a wireless medium, broadcast scheduling also suffers from collisions, when two or more nodes transmit messages to a common destination simultaneously. With collision, the destination node is not able to receive any of these messages. The Minimum Latency Broadcast Scheduling (MLBS) problem aims to find a collision-free schedule for broadcast with a minimum latency. The problem is proved to be NP-hard [1], and has been widely studied in always-active WSNs, where all nodes are assumed to be active all the time.

The broadcast gets more complex when recent WSNs have adopted the duty-cycle scheme to conserve energy and to extend the network lifetimes [2]. In a duty-cycled WSN, a sensor node alternates between active and sleep states to reduce energy consumption. Due to the periodic sleeping period of each sensor node, a sensor node has to wait until its receivers wake up before transmitting a message and may need to transmit the message multiple times if its receivers have different active slots. The minimum-latency broadcast problem in duty-cycled WSNs has received significant attention over the last few years.

Hong et al. [3] proved the NP-hardness of MLBS in the Duty-Cycled (MLBSDC) problem and proposed a broadcast scheduling scheme following a top-down layered approach on a Shortest Path Tree (SPT). Jiao et al. [4] improved the work by exploring the geometric properties of the Maximal Independent Set (MIS). Recently, the Latency-Aware Broadcast Scheduling (LABS) scheme [5] further reduces broadcast latency by allowing nodes to transmit at multiple time slots in a single working period.

The existing schemes for the MLBSDC problem do not allow any collision in a schedule to ensure its completion, i.e. all nodes receive a broadcast message collision-freely. A delay of transmission caused by collision-prevention may increase the broadcast latency if the delay occurs along a critical path of the network. The paper proposes a broadcast scheduling scheme that provides preference to nodes along critical paths of a network to minimize broadcast latency. The proposed scheme allows collision at non-critical nodes to speed up the broadcast process for critical ones. It ensures the completeness of broadcast scheduling by retransmission.

The remainder of this paper is organized as follows. In Section 2, we discuss related works. Section 3 includes network model, problem formulation, and related terminologies. Section 4 presents the proposed scheme, and then its performance evaluation in Section 5. Finally, we conclude the paper and discuss our future work in Section 6.

2. Related Works

Research on the minimum-latency broadcast problem has increased over the past few decades. One of the earliest works is Gandhi et al. [1]. They proved the NP-hardness of MLBS problem in Unit Disk Graph (UDG) model [6], where all nodes have the same transmission range. Their proposed algorithm gives an approximation ratio in terms of latency of at least 400. Huang et al. [7] exploited the fact that 12 colors are sufficient to color all the nodes in any independent set of an UDG and thus the distance between nodes with same color is more than two hops. As a result, they improved the latency ratio to 16. The algorithm in [8] recently improved the ratio to 12 by allowing a node to transmit more than once to reduce broadcast latency.

However, the above mentioned algorithms fail to capture the intermittently connected characteristic of duty-cycled networks. The MLBSDC problem is proven as NP-hard [3], and there are a handful of directly related works so far. One-To-All Broadcast (OTAB) algorithm [4] utilizes a correlation function between network topology information and the sleep schedule of each node to find the minimum latency from a source node to every node in a network. All nodes in the network are divided into layers, according to the minimum latency. The algorithm applies a D2-coloring method on a MIS of each active slot, and schedules transmissions layer-by-layer based on the assigned colors. Nevertheless, the algorithm requires all forwarding nodes in a 1-hop propagation to finish their transmissions before all neighbors in the next hop. As a result, the mechanism unnecessarily increases the delay of ensuring collision-free transmissions from those 1-hop neighbors and hence leads to a high broadcast latency.

Another notable result is LABS [5] which employs a flexible layered approach to reduce broadcast latency by utilizing independent scheduling between consecutive layers. The scheme allows nodes to transmit at several time slots in a single working period to maximize the number of receivers in the working period. It also explores geometric properties of the MIS to reduce the number of transmissions. A D2-coloring method is applied to prevent interference between transmissions for nodes within one layer. Simulation results show that the scheme consistently outperforms existing schemes in terms of broadcast latency, total transmissions, and total energy consumption.

3. Preliminaries

3.1 Network model and assumptions

All sensor nodes are uniformly deployed in a square field with one randomized source node as in [3]–[5]. The network topology is modeled as a connected graph. Two sensor nodes form a bidirectional communication link and become a neighbor to each other whenever the Euclidean distance between them is within their transmission range. Each sensor node in the network is assigned a unique identifier.

In duty-cycled environments, time is divided into unit time slots. These discrete time slots are grouped into multiple working periods, with fixed length T. Each sensor node u randomly selects one time slot in $\{0, 1, ..., T-1\}$ as its active slot $A(u)$. The sensor node periodically wakes up at $A(u)$ for each working period, and stays in the active state for the time slot.

We assume that every transmission occupies one unit time slot. A sensor node can forward a broadcast message only after it receives the message. A sensor node can wake up at any time slot to transmit a message, and can receive a message only at its active time slot. Due to the properties of a wireless environment, whenever a node transmits a message, all its active neighbor nodes hear the message. If a node hears more than one message at a time slot, it cannot receive any of these messages due to a collision. Therefore, sensor node u successfully receives a message if only one neighbor of u transmits the message at time slot $A(u)$.

3.2 Problem statement

In a single-source broadcast, a message is disseminated from source node s to all other nodes in a network. For simplicity, the source node is assumed to have the message in advance. Every other node in the network can be scheduled to transmit the message to its neighbors after its reception. The broadcast schedule completes when every node receives the message.

A broadcast schedule assigns transmitting times for all nodes in the tree such that the broadcast can complete. The broadcast latency is determined by the maximum assigned transmitting time. The objective of the MLBSDC problem is to find broadcast schedule with a minimum broadcast latency. The MLBSDC problem is an NP-hard problem [3].

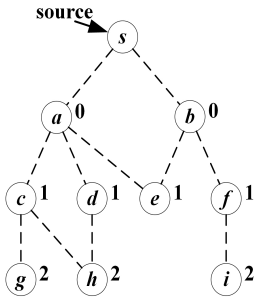
4. Proposed Scheme

4.1 Motivation

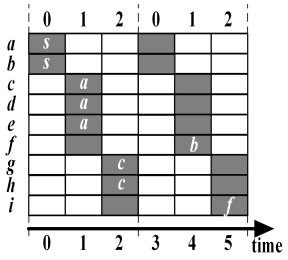
Existing broadcast scheduling schemes, such as OTAB [4] and LABS [5], are typically motivated to find a subset of nodes whose simultaneous transmissions will result in as many collision-free receptions as possible. In these works, the authors use the neighborhood information of nodes to determine whether a particular node needs to transmit a message. Different transmitting time slots are assigned to nodes having a common neighbor to prevent any collision.

A conventional broadcast schedule for a duty-cycled WSN with $T = 3$ is illustrated in Fig. 1b. In the schedule, the source node s broadcasts a message to its neighbors a and b at time slot 0. These two nodes are adjacent to node e whose active slot is 1. In order to prevent collision at the common neighbor, either a or b is allowed to forward the message at the time slot 1, and the other must be delayed to the next working period. As transmission of b is delayed in the example, its neighbor node f receives the message at time slot 4 and forwards it at time slot 5.

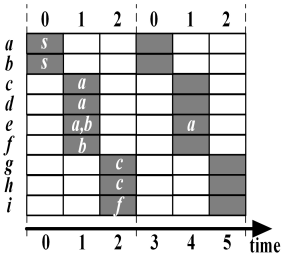
Observably, a delay of transmission from b to f increases broadcast latency as the node f is responsible for transmitting the message further. The problem is more severe in highly dense networks because there may be more nodes which cannot be scheduled simultaneously due to the collision. With the improved schedule in Fig. 1c, a transmission from b to f can be scheduled at time slot 1 despite causing a collision at e. Node f can forward the message at time slot 2. Consequently, the broadcast completes earlier than the conventional schedule, when e receives the message from a retransmission of a at time slot 4.



(a) Communication graph. A number next to a node indicates its active slot.



(b) Conventional schedule (latency: 5, number of transmissions: 5).



(c) Improved schedule (latency: 4, number of transmissions: 6).

Fig. 1: Motivation example. A node ID in a time slot indicates a transmitter.

4.2 Critical-aware Scheduling with Collision-tolerant (CSC)

In one-to-all broadcast, a critical path of a network is the longest path in terms of delay among the shortest ones from the source node to other nodes in the network. Nodes in a critical path are referred to critical nodes. Obviously, delaying the receiving times of a critical node will increase overall broadcast latency. The proposed scheme schedules a broadcast with preferred nodes along the critical paths of a network to reduce the broadcast latency. A collision-tolerant scheduling is employed to offer an opportunity of broadcast latency minimization.

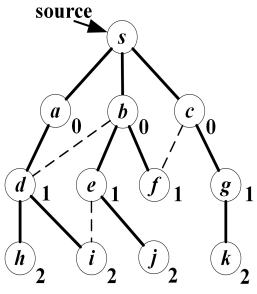
4.2.1 Criticality awareness

Let $G = (V, E)$ denote the communication graph of a duty-cycled WSN, where V is the set of vertices, and E is the set of edges. Let $s \in V$ denote a predefined source node of the network. For each edge in a communication graph $G = (V, E)$, we define two values, called *costs*, corresponding to two asymmetric directions of the edge. Each cost value is the number of time slots for which a transmission on a direction of the edge is delayed due to the sleeping period of its receiver. For simplicity, the source node s is assumed to have a message in advance, and its active slot is defined as $T-1$. The cost of an edge $(u, v) \in E$ with a direction from u to v can be determined as follows:

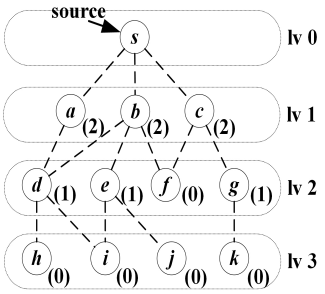
$$\text{cost}(u,v) = \begin{cases} A(v) - A(u), & \text{if } A(v) > A(u); \\ A(v) - A(u) + T, & \text{otherwise;} \end{cases}$$

The minimum of accumulated costs on the paths from the source node s to node $u \in V$ is referred as *level* of u. The level of u corresponds to the minimum latency for u to receive a broadcast message generated by s. As the source node s has generated the message in advance, its level is 0. Levels of other nodes in the network can be obtained by constructing a *Shortest Path Tree (SPT)* rooted at s on the latency cost. A critical path of the network is the shortest path from s to a node with maximum level.

In order to determine the criticality of nodes in the network, we define a *latency-ahead* value for each node u, denoted by $\text{la}(u)$, based on the SPT. The latency-ahead value of a node is accumulated costs from the node to a leaf node with maximum level in its sub-tree. The value presents the minimum latency for transmitting a message from a node to the furthest leaf node in the sub-tree rooted at the node. In other words, a node with a high latency-ahead value requires long delay to cover all nodes in its sub-tree. The higher latency-ahead value a node has, the more critical the node is. An example of SPT construction, level distribution, and latency-ahead calculation is showed in Fig. 2. It is worth noting that every nodes with a level k $(k > 0)$ has an active slot $i = (k-1) \mod T$.



(a) A shortest path tree along with active slots of nodes.



(b) A level-based topology along with latency-ahead values of nodes.

Fig. 2: An example for calculating latency-ahead value.

4.2.2 Collision-tolerant schedule

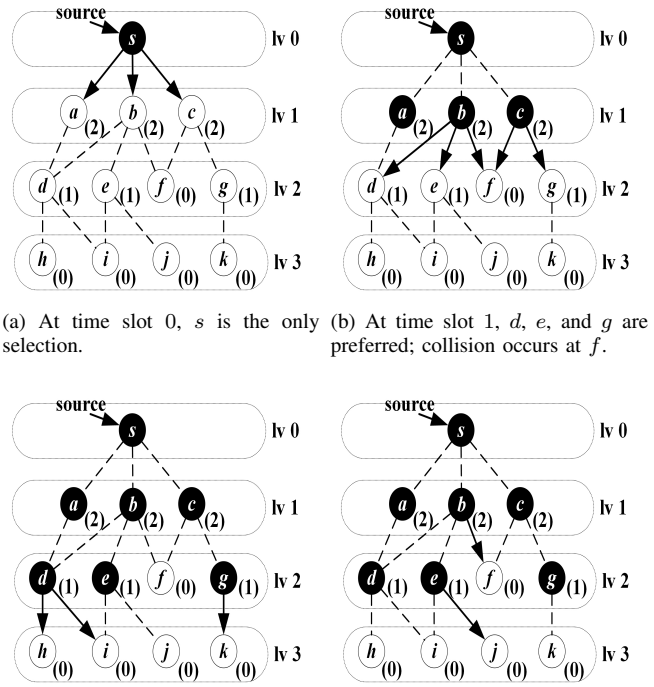
In the proposed scheduling, a node is called *covered* node if one of its neighbors has been scheduled to transmit a message to the node. The covered node is ready to be scheduled to cover its *uncovered* neighbors. Let C and $\overline{C}$ denote the set of covered nodes and the set of uncovered ones, respectively. Initially, C contains the source node alone and $\overline{C}$ contains the remaining ones. In order to reduce broadcast latency, Critical-aware Scheduling with a Collision-tolerant (CSC) algorithm prefers to cover critical nodes in $\overline{C}$ who have

high latency-ahead values. In each time slot, the algorithm minimizes the number of transmissions by selecting some nodes in C as forwarders so that the number of receivers in $\overline{C}$ is maximized.

The broadcast schedule starts at time slot 0 and iteratively works for each time slot. At each time slot i ($i \geq 0$), the algorithm searches for the most critical uncovered node u who has the highest latency-ahead and $A(u) = i \mod T$. All covered neighbors of u are considered as its forwarder candidates. Among the candidates, the algorithm selects the node $p(u)$ covering the largest number of uncovered nodes to reduce the number of transmissions. As a transmission from the selected forwarder covers all of its uncovered neighbors whose active slots are equal to $A(u)$, such neighbors become candidates for receiving a message at time slot i, later on called *listeners*.

CSC algorithm continues searching for the most critical node v, $A(v) = i \mod T$, among the uncovered ones excluding the listeners. The forwarder candidates of v should exclude all covered neighbors of listeners whose latency-ahead values are not smaller than $la(v)$ to prevent any collision at the critical listeners. A forwarder $p(v)$ for node v is selected in the same manner among the remaining forwarder candidates. All $p(v)$'s uncovered neighbors whose active slot is equal to $A(v)$, also become listeners. It is worth noting that collision may occur at some listeners who are common neighbors of $p(u)$ and $p(v)$. The algorithm allows such collisions to accelerate transmissions to critical nodes in the time slot i. Moreover, increasing the number of simultaneous transmissions is also beneficial for reducing broadcast latency.

The algorithm iterates the above procedure until it cannot find any forwarder, or all nodes who are active at the current time slot are covered. All listeners that have collided are still uncovered nodes as they cannot receive a message at the current time slot. The other listeners become covered ones. CSC algorithm iteratively moves to the next time slot, i.e. $i + 1$, until it covers all nodes in the network. Fig. 3 shows an example of the proposed algorithm on a network of 12 nodes and $T = 3$.



(a) At time slot 0, s is the only selection.

(b) At time slot 1, d, e, and g are preferred; collision occurs at f.

(c) At time slot 2, collision is not allow at i as $\mathrm{la}(i) = \mathrm{la}(j)$.

(d) A retransmission from b to f at time slot 4, and broadcast completes at time slot 5.

Fig. 3: An illustration example of proposed scheme. Black and white colors indicate covered and uncovered nodes, respectively.

5. Performance Evaluation

In this section, the performance of the proposed scheme is evaluated through extensive simulations. We demonstrate the efficiency of CSC algorithm using a simulator written in C#. Broadcast latency and number of transmissions given by the proposed algorithm are compared with those of OTAB [7] and LABS [8].

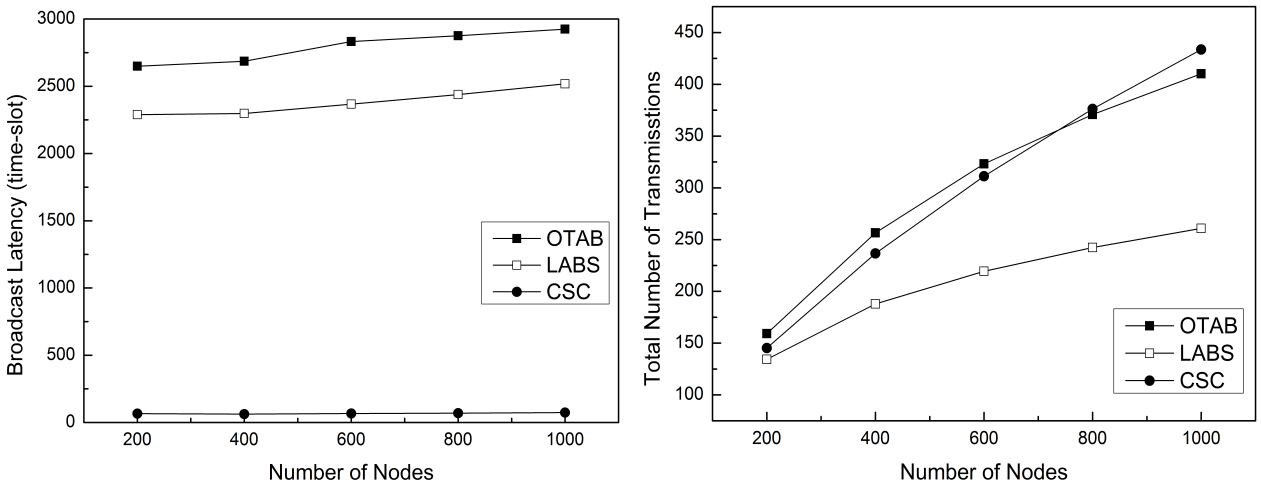
5.1 Simulation environment

For fair comparison, the simulation configuration is similar to the one in [8]. In all simulations, the network area is a square of $200m200m$, with sensor nodes deployed randomly within the area. We assume that all sensor nodes have the same transmission range of $30m$. The number of nodes ranges from 200 to 1000 with an interval of 200.

The duty cycle which equals to $1/T$ is changed by varying T from 10 to 50 with an interval of 10. All the simulations are conducted with two predefined parameters and the other one varied. For each configuration, the algorithms are run on 200 randomly generated graph topologies, and the source node is randomly chosen as well. Then, the average results of these simulations are reported to evaluate the performance of the proposed scheme.

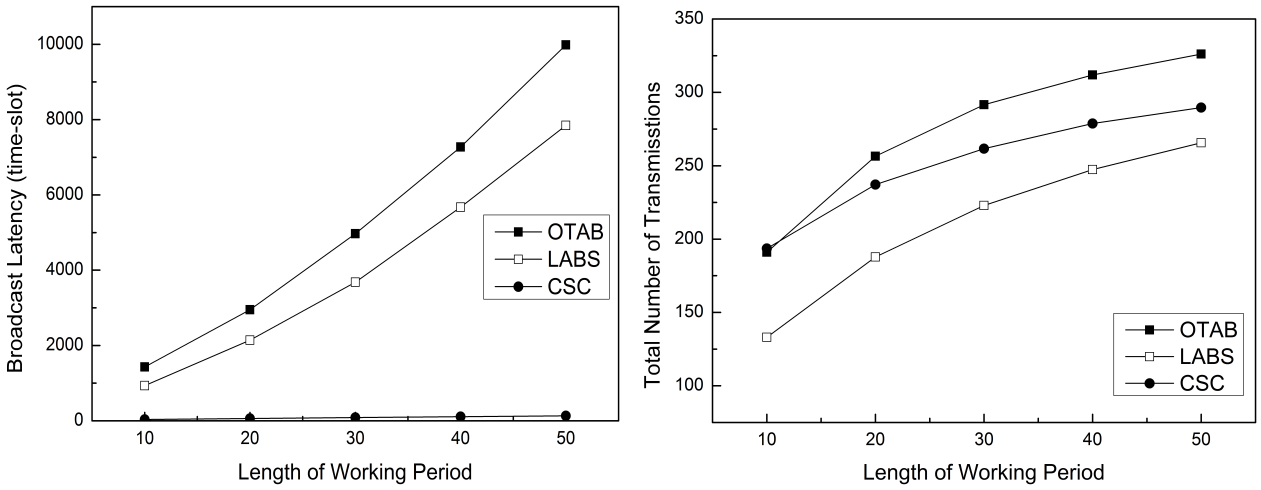
5.2 Simulation results

First, we study the impact of network size on the performance of OTAB, LABS and CSC. The working period length T is fixed to 20. When the network density becomes high, i.e. the number of nodes increases, a node needs to compete with more neighbors to transmit a message. The broadcast latency of all schemes increases as a node needs to wait longer to prevent interference with its neighbors. By accelerating transmissions for critical nodes and allowing some collisions at less-critical ones, the latency of CSC is $34-36$ times shorter than that of LABS as shown in Fig. 4a The total numbers of transmissions of all schemes grow as



(a) (b)

Fig. 4: Impact of network size (duty cycle is 0.05).



(a) (b)

Fig. 5: Impact of duty cycle (number of nodes is 400).

the number of nodes does. Fig. 4b shows that the proposed scheme produces $8.2-66.2$ percent more transmissions than LABS due to retransmissions.

Next, we examine the performance of the algorithms under different duty cycle $1/T$. These simulations are run with 400 nodes. Because a working period has more time slots, the number of levels in the shortest path tree increases, resulting in the increasing latency of broadcast schedules as shown in Fig. 5a. Instead of deferring a transmission to prevent collision at every node in a network, CSC algorithm allows some collisions at non-critical nodes to reduce the broadcast latency by $25-59$ times compared to LABS. Fig. 5b shows that the total number of transmissions grows with the decrease of the duty-cycle because each forwarding node may require more transmissions to inform all of its neighbor nodes with different active slots. We can observe that the proposed scheme requires $9-45.5$ percent more transmissions than LABS because of re-transmissions.

6. Conclusions

This paper presents a novel broadcast strategy for Minimum Latency Broadcast Scheduling in Duty-Cycled (MLB-SDC) wireless sensor networks. The proposed scheme is a combination of criticality awareness and collision-tolerant scheduling which offers the opportunity to reduce the broadcast latency. Extensive simulations show that our proposed algorithm provides at least 25 times shorter latency while increasing the number of transmissions at most 66.2 percent

compared with those of LABS. In future, we interested in finding a threshold so that collision is only allowed at nodes whose criticality are under the threshold. This approach is expected to reduce the number of re-transmissions. Broadcast schemes that balance energy consumption to maximize the network lifetime is also another exciting problem.

Acknowledgment

This research was supported in part by MOE and MSIP, Korean government, under ICT R&D program (B0101-15-1366), PRCP (NRF-2010-0020210) and Basic Science Research Program (NRF-2013R1A1A2064302) through NRF, respectively.

References

[1] R. Gandhi, A. Mishra, and S. Parthasarathy, "Minimizing broadcast latency and redundancy in ad hoc networks", *IEEE/ACM Transactions on Networking*, vol. 16, issue 4, pp. 840–851, 2008.

[2] G. Anastas, M. Conti, M. D. Francesco, and A. Passarell, "Energy Conservation in Wireless Sensor Networks: a Survey", *Journal Ad Hoc Networks*, vol. 7, issue 3, 2009.

[3] J. Hong, J. Cao, W. Li, S. Lu, and D. Chen, "Sleeping schedule-aware minimum latency broadcast in wireless ad hoc networks", *in Proc. of IEEE ICC*, 2009.

[4] X. Jiao, W. Lou, J. Ma, J. Cao, X. Wang, and X. Zhou, "Minimum latency broadcast scheduling in duty-cycled multi-hop wireless networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, pp. 110–117, 2012.

[5] D.-T. Le, T. Le-Duc, V. V. Zalyubovskiy, D. S. Kim, and H. Choo, "LABS: Latency aware broadcast scheduling in uncoordinated duty-cycled wireless sensor networks", *Journal of Parallel and Distributed Computing*, vol. 74, issue 11, pp. 3141–3152, 2014.

[6] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs", *Discrete Mathematics*, vol. 86, issue 13, pp. 165–177, 1990.

[7] S.C. Huang, P.-J. Wan, X. Jia, H. Du, and W. Shang, "Minimum-latency broadcast scheduling in wireless ad hoc networks", *in Prof. of IEEE INFOCOM*, pp. 733–739, 2007.

[8] R. Gandhi, Y.-A. Kim, S. Lee, J. Ryu, and P.-J. Wan, "Approximation algorithms for data broadcast in wireless networks", *IEEE Transactions on Mobile Computing*, vol. 11, issue 7, pp. 1237–1248, 2012.

A Message Efficient Group Membership Algorithm in Mobile Ad Hoc Distributed Systems

Yong Hwan Cho, Sung Hoon Park, Seon-Hyong Lee and Byeong Sun Hwang

School of Electrical and Computer Engineering, Chungbuk National Unvi. Cheongju

ChungBuk 361-763

E-mail: [spark,yhcho@chungbuk.ac.kr, genexwave@hanmail.net]

Abstract

The Group membership paradigm can be used as a building block in many practical problems such as group communication, atomic commit and replicated data management where a group membership protocol might be useful. The problem has been widely studied in the research community since one reason for this wide interest is that many distributed protocols need a Group membership protocol. However, despite its usefulness, to our knowledge there is no work that has been devoted to this problem in a mobile ad hoc computing environment. Mobile ad hoc systems are more prone to failures than conventional distributed systems. Solving group membership in such an environment requires from a set of mobile nodes to choose a unique node as a leader based on its priority despite failures or disconnections of mobile nodes. In this paper, we describe a solution to the Group membership problem from mobile ad hoc computing systems.

Key-words: Synchronous Distributed Systems, Group membership, Fault Tolerance, Mobile Ad Hoc Environment.

1. Introduction

Distributed systems consist of groups of processes that cooperate in order to complete specific tasks. A Group Membership Protocol is of particular use in such systems, providing processes in a group with a consistent view of the membership of that group. In this way, when a membership change occurs, processes can agree on which of them must complete a pending task or start a new task. The problem of reaching a consistent membership view is very similar to the one of achieving common knowledge in a distributed system, commonly referred to as the Consensus Problem [7].

The Group membership problem [1] requires that every node connected in a network has a consistent group membership view if all connected nodes are belong to one group. The problem has been widely studied in the research community [2,3,4,5,6] since one reason for this wide interest is that many distributed protocols need a Group membership protocol. However, despite its usefulness, to our knowledge there is no work that has been devoted to this problem in a mobile ad hoc computing environment.

When nodes are mobile, topologies can change and nodes may dynamically join/leave a network. In such networks, Group membership can be changed frequently, making it a particularly critical component of system operation.

Mobile ad hoc systems are more often subject to environmental adversities that can cause loss of messages or data [8]. In particular, a mobile node can fail or disconnect from the rest of the network. Designing fault-tolerant distributed applications in such an environment is a complex endeavor [9,10].

The aim of this paper is to propose a solution to the group membership problem in a specific ad hoc mobile computing environment. This solution is based on the termination detection algorithm that is a classical one for synchronous distributed systems. The rest of this paper is organized as follows. Section 2 describes the mobile system model we use. In Section 3, a solution to the Group membership problem in a conventional synchronous system is presented. A protocol to solve the group membership problem in a mobile ad hoc computing system is presented in Section 4. We conclude in Section 5.

2. System Model, Constraints and Assumptions

Before developing a Group membership algorithm for ad-hoc computing environments, we first define our system model based upon assumptions and goals. We model an ad hoc network as an undirected graph, i.e., $G = (V, E)$, where vertices V correspond to set of mobile nodes $\{1, 2,\ldots,n\}$ ($n > 1$) with unique identifiers and edges E between a pair of nodes represent the fact that the two nodes are within each other's transmission radii and, hence, can directly communicate with one another that changes over time as nodes move. Each process i has a variable N_i, which indicates the neighboring nodes, with that i can *directly* communicate the neighboring nodes. We assume that every communication channel is bidirectional; $j \in N_i$ iff $i \in N_j$. More precisely, in the network $G = (V, E)$, we can define E such that for all $i \in V$, $(i, j) \in E$ if and only if $i \in N_j$. The graph can

become disconnected if the network is partitioned due to node movement. Because the nodes may changes their location, N_i may be dynamically changed and so may G accordingly. We make the following assumptions about the nodes and system architecture:

- All nodes have unique identifiers. They are used to identify participants during the Group membership detection process.
- Links are bidirectional and FIFO, i.e. messages are delivered in order over a link between two neighbors.
- Node mobility may result in arbitrary topology changes including network partitioning and merging. Furthermore, nodes can crash arbitrarily at any time and can come back up again at any time.
- A message delivery is guaranteed only when the sender and the receiver remain connected (not partitioned) for the entire duration of message transfer. Each node has a sufficiently large receive buffer to avoid buffer overflow at any point in its lifetime.

The objective of our Group membership algorithm is to ensure that after a finite number of topology changes, *eventually* each node *i* has a consistent view of group membership of the group to which *i* belongs.

3. Group Membership Specification

We now define a specification, consisting of four properties, for a group membership algorithm. We assume the system to be initialized to a start state where the sequences are the same at all processes and the last nonempty views in their sequences are the ones reported by all failure detectors.

Property 1: Agreement. At any point in time, all processes have a consistent history.

Property 2: Termination. If there are no more changes in the local views of the processes, they eventually reach their quiescent states.

Property 3: Validity. If all processes in a view v* perceive view v* as their local view and they have reached their quiescent states, then the last nonempty elements of their sequences of global views are all at position j and must be equal to v*.

Property 4: Safety. Once a view is "committed" in the sequence of global views, it cannot be changed.

The first property expresses agreement. Consistent history must be an invariant for any program that satisfies the specification. The second property expresses termination. When the inputs of all processes are stable, the processes are eventually going to stop changing their output sequences. The third property rules out trivial solutions where protocols never decide on any new view or always decide on the same view. It ensures that a protocol that satisfies the specification does something useful, by stating that when all processes in a set agree on such set, they must commit this common view at the same position j in their sequences of global views. Note that this requirement is weak because a new membership is created only if the local views of the different processes in the membership reach agreement. The fourth property also rules out trivial solutions, requiring processes not to change old views in their sequences.

3. Group Membership Algorithm in an Ad Hoc Network

In this section, we describe a Group membership algorithm based on the termination detection algorithm, simply TDA, by diffusing computations. In later sections, we will discuss in detail how this algorithm can be adapted to a mobile setting.

3.1 A Group Membership based on TDA

We first describe our group membership algorithm in the environment of a static network, where we assume that nodes and links never fail. The algorithm consists of three phases operated at the node that initiates the group membership algorithm. 1) Scattering phase - it operates by first scattering the "who" message and 2) Gathering phase - it operates by then gathering the id of each node that is connected to the static networks. We refer to this computation-initiating node as the *source node.* 3) Completing phase – it operates by deciding the consistent view and announcing it as a consistent new view to all nodes.

As we will see, after gathering all nodes' ids completely, the source node will have the information enough to determine a consistent group membership view and will then broadcast it to the rest of the nodes in the network. The algorithm uses three messages, i.e., *Who, Ack* and *View*.

1) **Scattering phase.** *Who* message is used to initiate the group membership protocol by "scattering" the *Who message*. When group membership protocol is triggered at a source node *s*, the source node makes a waiting list *wl* and a received list *rl* and begins a *diffusing computation* by sending an *Who message* to all of its immediate neighbors. Initially the waiting list consists of only its immediate neighboring node's ids and the received list is empty.

When node *i* receives a *Who message* from the neighboring node for the first time, it immediately sends the *Ack* message to the source node and propagates the *Who message* to all its neighboring nodes except the node from which it first received an *Who message.*

The *Ack* message sent by node *i* to the source node contains the ids of all its neighboring nodes that are needed for the source node to decide the consistent view of the nodes connected with a distributed network. After that, any *Who message* received by other neighboring nodes will be ignored.

2) **Gathering phase**. When the source node receives the *Ack* message from the node *j*, it removes *j* from the waiting list and puts *j* into the received list and immediately checks one by one the every node's id contained in the *Ack* message. If there is the any id in the *Ack* which has already been acknowledged, i.e. that means it is in the received list, it is discarded. Otherwise, it is put into the waiting list of source node and the source node waits the *Ack* message from it.
The waiting list is growing and shrinking repeatedly based on the received *Ack* messages, but the received list is steadily growing by receiving the *Ack* messages. But the waiting list eventually could be empty and the received list could include all ids of nodes connected to the networks when the source node received the *Ack* messages from all other nodes. Hence the source node eventually has sufficient information to determine the consistent view of the group based on the received list, because the waiting list could be eventually empty and it means that the source node has received the *Ack* messages from all the nodes.

3) **Completing Phase.** Once the source node has received *Ack*s from all other nodes, it determines the consistent view based on the received list and broadcasts a *View* message to all other nodes announcing the current view of the group.
We illustrate a sample execution of the algorithm. We describe the algorithm in a somewhat synchronous manner even though all the activities are in fact asynchronous. Consider the network shown in Figure 1(a). In this figure, and for the rest of the paper, thin arrows indicate the direction of flow of *Who message* s and dotted arrows indicate the direction of flow of *Ack* messages to the source node. As shown in Figure 1, node A is a source node that initializes wl_a and rl_b with {B,C} and {A} respectively and starts a diffusing computation by sending out *Who message*s (denoted as "E" in the figure) to its immediate neighbors, viz. nodes B and C, shown in Figure 1(a).

As indicated in Figure 1(b), nodes *B* and *C* in turn propagate the *Who message* to its immediate neighbors only except the source node and send the *Ack* message with neighboring node list to the source node *A*. Hence *B* and *C* also send *Who message* s to one another. But the *Who message*s are not acknowledged to the source node since nodes *B* and *C* have already received *Who message* s from the source node respectively. The information about neighboring node is piggybacked upon the *Ack* message sent by each node. Upon received *Ack* messages from B and C, node A updates wl_a = { B,C }, rl_b = { A } with the neighboring node information piggybacked on the *Ack* messages.

5. Concluding Remarks

In this paper, we proposed an asynchronous, distributed group membership algorithm for mobile, ad hoc networks and showed it to be correct. We formally specified the property of our group membership algorithm using temporal logic. We have assumed the ad-hoc network topology is dynamically changing and nodes are frequently connected and disconnected over the networks. With this approach, the group membership specification states explicitly that progress and safety cannot always be guaranteed. In practice, our requirement for progress is that there exists a constant *c* such that if connection or disconnections occur for a period of at least c, then by end of that period, the system reaches a state satisfying a consistent view. Furthermore, the system remains in that state as long as no failures or disconnections occur. In fact, if the rate of perceived a node failures in the system is lower than the time it takes the protocol to make progress and accept a new consistent view, then it is possible for the algorithm to make progress every time there is a node failure in the system.

In real world systems, where process crashes actually lead a connected cluster of processes to share the same connectivity view of the network, convergence on a new consistent view can be easily reached in practice. However, the algorithm should work correctly even in the case of unidirectional links, provided that there is symmetric connectivity between nodes. We are currently working on the proof of correctness in the case of unidirectional links. We are also investigating on how our group membership algorithm can be adapted to perform clustering in wireless, ad hoc networks.

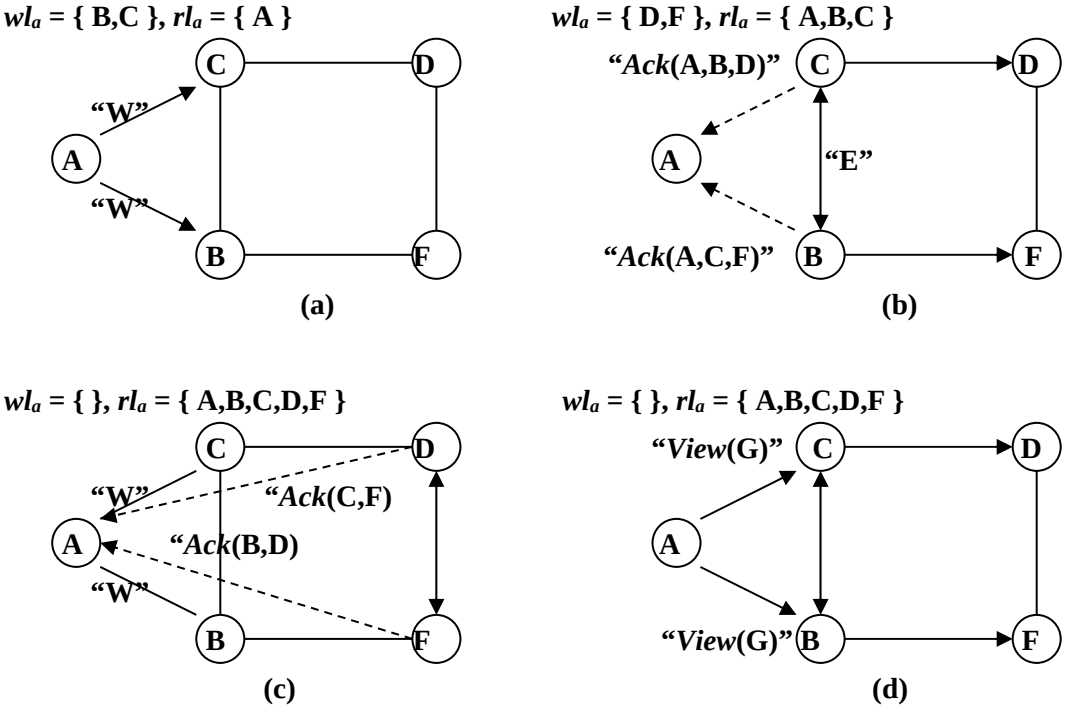


Figure 1: An execution of group membership algorithm based on the node detection algorithm. Arrows on the edges indicate transmitted *Who message*s, while dotted arrows parallel to the edges indicate *Ack* messages. In Figure 1(c), the node D and F also send the *Ack* messages to the sources node when they received the *Who message* s from the B and C respectively. Each of these *Ack* messages contains the identities of the neighbor. Eventually, the source *A* hears all acknowledgments from all of other nodes except itself in Figure 1(d) and then decides the consistent view among the group and broadcasts it, via the *View* message shown in Figure 1(d).

6. References

[1] Y. Amir, L. E. Moser, P.M. Melliar-Smith, D.A. Agarwal, and P. Ciarfella, "The Totem Single-Ring Ordering and Membership Protocol," ACM Trans. Computer Systems, vol. 13, no. 4, pp. 311-342, Nov. 1995.

[2] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg, "On the Formal Specification of Group Membership Services," Technical Report 95-1534, Computer Science Dept., Cornell Univ., Aug. 1995.

[3] T. Anker, G.V. Chockler, D. Dolev, and I. Keidar, "Scalable Group Membership Services for Novel Applications," Proc. Workshop Networks in Distributed Computing (DIMACS 45), pp. 23-42, 1998.

[4] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev, "A Client Server Oriented Algorithm for Virtually Synchronous Group Membership in WANs," Proc. 20th Int'l Conf. Distributed Computing Systems, Apr. 2000.

[5] J. Brunekreef, J.-P. Katoen, R. Koymans, and S. Mauw, "Design and analysis of dynamic leader Group membership protocols in broadcast networks," *Distributed Computing*, vol. 9, no. 4, pp. 157-171, 1996.

[6] D. Bottazi, R. Montanari and G. Rossi, "A self-organizing group management middleware for mobile ad-hoc networks," Computer Communications, vol. 31, no. 13, pp. 3040-304, 8 Elsevier, 2008.

[7] David Powell, guest editor. Special section on group communication. *Communications of the ACM*, 39(4):50-97, April 1996.

[8] Pradhan D. K., Krichna P. and Vaidya N. H., Recoverable mobile environments: Design and tradeoff analysis. FTCS-26, June 1996.

[9] L. Briesemeister and G. Hommel, "Localized group membership service for ad hoc networks," Proc. International Conference on Parallel Processing Workshops, IEEE Computer Society, pp. 94-100, 2002.

[10] K. Hatzis, G. Pentaris, P. Spirakis, V. Tampakas and R. Tan. Fundamental Control Algorithms in Mobile Networks. In *Proc. of 11th ACM SPAA*, pages 251-260, March 1999.

Optimizing the Global Execution Time with CUDA and BIGDATA from a Neural System of Off-line Signature Verification on Checks.

Francisco Javier Luna Rosas[1,2], **Julio Cesar Martínez Romo**[1], **Damián Martínez Díaz** [2], **Gricelda Medina Veloz**[3], **Valentín López Rivas**[1], **Cesar Dunay Acevedo**[1]
e-mail: fcoluna2000@yahoo.com.mx

[1] Computer Science Department, Inst. Tec. Aguascalientes, México.
[2] Universidad Cuauhtémoc, Campus Aguascalientes, México
[3]Universidad Tecnológica del Norte de Aguascalientes, México

Abstract - The CUDA platform enables us to use the graphic cards not only to process the graphs but also to process and perform instructions in a parallel way, these improvements in the software have generated a wide catalogue of applications powered by the CUDA architecture. In this article we propose to optimize the global execution time of an off-line system to verify signatures on checks, our architecture operates on two phases, the training phase and the verifying phase. The training phase is made up of various stages with the purpose of generating a model of neuronal networks to recognize an off-line signature on checks. The verifying phase consists in repeating the first stages of the training phase with the purpose of extracting features from the signature. The features extracted from the signature on the verifying phase are compared on the classifier with the results gotten from the training phase model.

Keywords: CUDA, Redes Neuronales, Verificación de Firmas Off-line, Checks, Optimization.

1. Overview.

Signature verification consists on determining if given a number of samples of the signature of a person, an additional signature was performed by the same person. In this case the signature verification can be used as an authentifier of personality. A signature can be verified on-line and off-line. In the first case, an instrumented pen or a digitizing tablet is used to "capture" the shape of the signature and the dynamic movement of the hand [15] and, of course, it requires the signature's owner to be present. The off-line technique refers to situations in which the signature was performed on paper previously and it was recorded as an image [15], in this way, the valuable dynamic information is lost, and it is basically not recovered. In both methods, one signature is available, we proceed to get a number of features that should be reliable in order to recognize genuine signatures as well as to reject forged signatures, even skilled forgeries; a certain number of features is extracted and figured out from each of the sample signatures, and this way a group of patterns is formed and at the same time it is useful for the training and testing of a classifier. The job of the classifier is to "learn" the habitual behavior of the features in a signature "to test" later if such features behave the "same way" in a test signature. Off-line signature verification is a problem in which the performance archived can not be as high as in the case of on-line signature verification, due to the lack of dynamic information.

1.1 Previous work in off-line handwritten signature verification.

Off-line signature verification has been a research problem during many decades and in different countries [15], with many practical and potential applications. In [8], the authors identified the three kinds of forgeries described in Table 1. In the literature related to off-line verification we have generally seen good performances of error classification in random forgeries, as in [8] where it is about 0.38% or in [4] of 3%; however, in simulated forgeries and skilled forgeries the percentage of classification errors grows dramatically, and just to mention one case, lets say Fang in [5], who reports up to a 33.7%. However, this situation is not related with the ability of the researcher but with the nature of the off-line signature verification problem itself and it is also due to the loss of dynamic information; a factor that worsens the problem of high error under skilled

forgeries, this is because a model of signature from each individual should be prepared based on a few samples say 8 to 15 [15], which generates uncertainty in the probability to recognize genuine signatures as well as to reject forgeries.

Table 1. Types of Forgeries According to Justino. Courtesy of [8].

No.	Name	Description
1	Random Forgery	No attempt is made to reproduce the shape or aspect of the genuine signatures. No resemblance of the genuine signature is desired by the forger.
2	Simulated Forgery	Signature is loosely copied, not too detailed or accurate. The false signature "tends" to be similar to a genuine one.
3	Skilled Forgery	Signature is very similar to a genuine one.

Another main problem in the off-line verification approach is to establish and extract a group of features from the test signatures that should be enough to allow a high capacity to recognize genuine test specimens and at the same time to discriminate and reject forgeries. In the literature of this topic we can find basically three approaches to generate features.

One consists on isolating and characterize some segments of the signature, (curvature, smoothness) [5]; another is to place a grid or array of squares on the signature and consider each element (square) of the grid as an area to be characterized; a classical example of the last mentioned is in [17]; in both cases we are searching for a clear representation using vectors that describe the values of each feature. In some cases, squares overlapping is considered to represent the signature like in [12], in which Murshed and others used squares of 16x16 pixels with an overlapping of 50%. Finally, another approach is based on schemes in which the features are implicit in one kind of parameter, such as the case of Gouvêa in [7], who used neuronal networks in its auto associative version. The features are implicit in the neuron's weights.

In the last stage, the classifier will decide whether the signature is genuine or not. Neuronal networks of different types have been used [7], [4], [12], [15]; and hidden Markov models [8] and other less sophisticated classifiers such as the nearest k-neighbors [17] and the minimum distance based on Mahalanobis distance [5]. No matter the kind of classifier, the reported results on skilled forgeries are lower than those obtained under on-line verification.

2. Verifier Architecture in Checks.

The architecture of our automatic verifier for off-line handwritten signatures is shown on Fig. 1; two phases are distinguished: the training phase (upper part) and the verification phase (lower part). The objective of the training phase is to generate a model for each person enrolled in the system, while the verification phase is to do the verification itself. Next we describe the constitutive blocks of each phase.

2.1 Training Phase.

The stages of the training phase are found on the upper part of Fig. 1 and are described in this section.

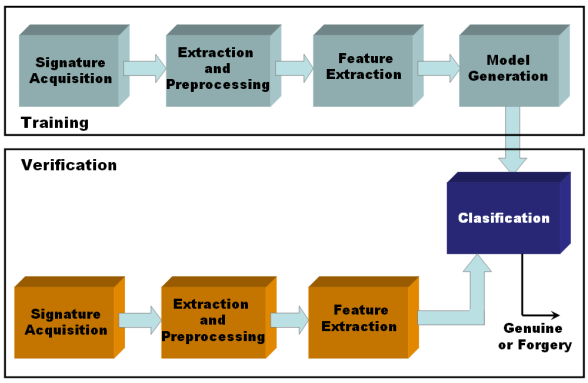


Fig.1 Verifier's Architecture in Checks. Courtesy of [11].

2.1.1 Acquiring signatures on checks.

The first block is the acquisition of images on checks with signatures, and it is performed with an image searcher on checks on the WEB.

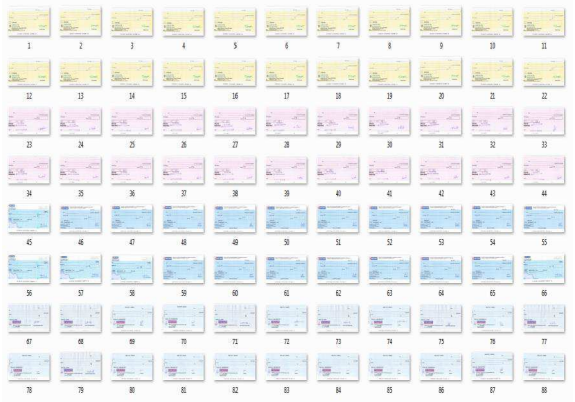


Fig. 2 Data Base of Checks. Courtesy of [13].

The searcher gets different images of checks of different signatures and are stored on a data base of signatures on checks (Fig. 2). An example of "John Joner`s" signature is shown on Fig 3.

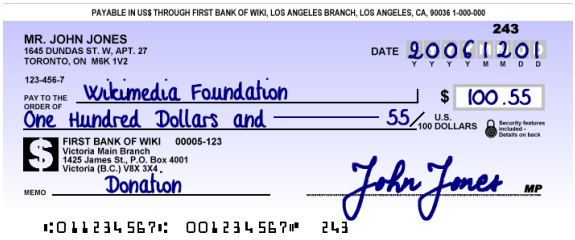


Fig. 3 Acqusition of "Johon Joner`s" Signature on a Check`s Image.

2.1.2 Extracting and Processing of Signatures.

On each image of the digital check there is a variety of related patterns, therefore, it`s necessary to separate the outstanding features from the rest of the image (Fig. 4).

Fig. 4 Segmentation of a Check`s Image. Courtesy of of [1].

As we can observe, point eight is the place reserved for the person who signs the check, we have to take on account that such signature has to be hand-written. The signature remains on a fixed region of the image (lower right side) and it`s necessary to extract the signature from the check, after being located, in order to do this we apply the following algorithm:

1. Change the RGB into a grey scale.
2. Binarize the image using a specific threshold per image.
3. Apply the search of lighted bits (black) on a binary image.
4. Add a list of coordinates where the lighted bits were found.
5. With the coordinates in the list we generate the size of the outcoming image (see Fig. 5). In case that the generated image doesn`t have a threshold, repeat steps 2 to 5 incrementing the threshold for the binarization of the image until it fulfills optimal dimentions.

Fig. 5 Extraction of a Signature from a Check.

.

As we can see on Fig. 5, the signature still has a noise so it`s necessary to apply some post-processing to obtain only the signature.

2.1.3 Feature Extraction.

Features Description.

Since we have only the static information of the handwritten signature, the problem of verifying a signature is more complex than the verification on-line if the purpose is to reject the skilled forgeries. Our verification strategy is based on the verification method of handwritten signatures used by human experts. The elements on which a human verifier is based, according to Slyter [16], include static and dynamic elements. The static elements have to do with the shape and design of the signature. The dynamic elements include the absolute pressure, the variations of pressure and speed grouped in what Slyter calls "the rhythms". The rhythms and shape are mixed during the performance of the signature in a unique way for each individual, which shows the habits developed when performing his signature consecutively and for a long period of time, thus any attempt to verify a signature should consider the balance between rhythms and shape.

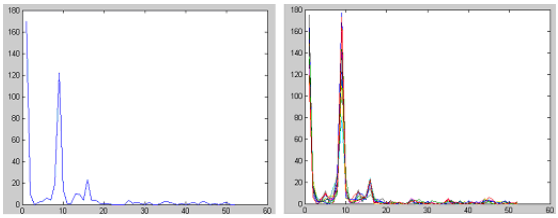


Fig. 6 Feature Vector of 10 Signatures of "John Doe". Courtesy of [11].

Using mathematical morphology and a number of structural elements it is possible to detect for any signature the position of the curved lines (equivalent regions of low speed). A graph of one feature vector of "John Doe´s" signature is shown in the left side of Fig. 6. Each value in that vector represents the number of pixels "lighted" (which represent a value of 1) after the signatures image have been erosioned; such number is the same as the number of

occurrences of the structural element in the image. The right side of Fig. 6 shows a graph of ten feature vectors corresponding to the same number of "John Doe´s" signatures. Notice the repetition in the vectors.

2.1.4 Generation of the Model.

In this sub-section the generation of the model of the signature is explained and the classifier's design is included. After the signature has been binarized, we applied the following steps to generate the model of the signature and classifier.

1. Morphological Filtering. The basic idea of the morphological image processing is that the structural element be used to examine a group of images. A group of operations produces structural information about the image. Historically, the morphological image processing is with binary images and image processing in the gray scale. In this case, we will work only on the binary case, and the morphological operation used is the erosion.
2. Generation of Training Patterns. On Table 2 we observe the entering patterns that will be provided to the neural network, note that each row on the table is considered as a pattern (or training example). The table is divided in the following sections:

Real Patterns: Come from line 1 to 10 and are formed by the erosion of each one of the ten signatures of a single signer (our fictitious signing person "John Doe"). Notice that each column of this section belongs to a value that represents the number of the structural element from which such signature was eroded, thus the whole number that appears in the column is the sum of the lighted bits (ones) which remained after eroding the signature with the structural element in question, each line is a feature vector.
Synthetic Positive Patterns: Belong from line 11 to 60. To obtain a column of these numbers, random numbers are generated in a range of

$$\frac{1}{10}\sum_{k=1}^{10} x_k \pm \sigma \qquad (1)$$

where x_k is the value of each EE from Table 2 along the signatures Sig1, Sig2, Sig3,…,Sig10.
Synthetic Negative Patterns: Belong from 61 to 110. To obtain a column of these numbers, random numbers are generated in a range of 1 and 300.

Table 2. Examples of Training for the Neural Network. Courtesy of [11].

Kind of Pattern	Number of Sig.	Structural Element (ocurrences in the signature)						
		EE1	EE2	EE3	EE4	EE5	--------	EE54
Real Patterns	Sig 1	1153	829	568	448	405	--------	393
	Sig 2	1077	696	447	369	331	--------	227
	Sig 3	1221	817	535	427	368	--------	393
	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-
	Sig 10	1238	856	516	386	317	--------	276
Synthetic Positive Patterns	Sig 11	1117	745	549	431	270	--------	340
	Sig 12	1135	879	541	311	359	--------	352
	Sig 13	1083	723	450	397	272	--------	270
	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-
	Sig 60	1176	858	460	392	314	--------	339
Synthetic Negative Patterns	Sig 61	247	252	156	213	247	--------	178
	Sig 62	73	295	221	20	63	--------	274
	Sig 63	80	114	10	226	70	--------	6
	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-
	Sig 110	137	201	47	32	164	--------	226

3. Backpropagation Neural Network (Classifier). The architecture of the neural network [2] from Fig. 7 is formed by an input layer with 54 neurons, a single hidden layer with 108 neurons and a neuron in the output layer, with a sigmoidal function. There is a single neuron in the output layer to map each input pattern to +5 (genuine) or -5 (forgery).

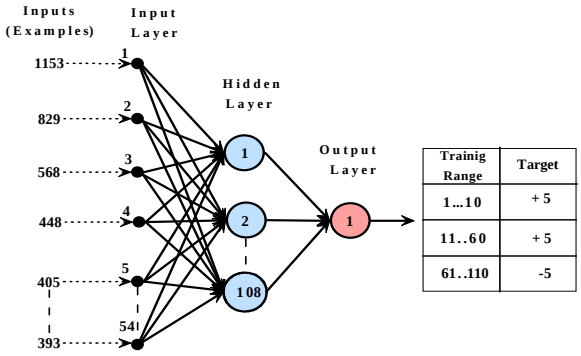


Fig. 7 Architecture of the Neural Network. Courtesy of [11].

4. Classifier Training. After the classifier architecture has been designed the next step is to train the classifier in such a way that it works as a recognition tool in the next phase of the verifier. The quadratic error was considered a 0.003, the maximum number of interactions in case the quadratic error is not reached was determined in 1000 interactions, the learning rate was established in a value of 1E-20 to advance over the surface of the error with small increases of the weights.

2.2 Verification Phase.

When a signature under testing is presented to the verifier, the following events take place:

1.- The first three verification stages are carried out, which generate the features that originate the training patterns in each signature.
2.- After the training patterns are generated, the classifier is not trained it only verifies the signature declaring it as a genuine (+5) or false (-5).

Table 3 shows the results of verification over a group of training and test signatures from a single person. As we can observe, the genuine test signatures (Sig1-Sig5) obtain outputs from the neural network very close to +5, the success in the verification is because in the training group a plenty of samples of genuine signatures were provides to the neural network (real plus synthetic) and that the neural network also has knowledge of the way the genuine signatures are NOT, information contained in the group of negative synthetic examples.

Table 3. Neural Network Output Obtained During Verification. Courtesy of [11].

Kind of Pattern (testing phase)	Number of Sign. (test)	Some Structural Element (ocurrences in the signature)						Neural Network Output
		EE1	EE2	EE3	EE4	EE5	EE6	
Real Patterns	Sig 1	159	17	4	3	2	1	4.6868
	Sig 2	118	6	0	3	10	5	6.6339
	Sig 3	133	13	4	4	7	5	4.6552
	Sig 4	182	13	2	3	7	6	4.9243
	Sig 5	138	12	2	4	9	4	5.2071
Positive Synthetic Patterns	Sig 6	134	17	2	1	-1	6	5.306
	Sig 7	153	19	2	1	10	1	5.262
	Sig 8	153	21	4	3	4	3	4.897
	Sig 9	154	20	1	3	4	0	5.341
	Sig 10	148	12	1	2	6	1	5.109
Forgeries (not skilled)	Sig 11	33	2	0	9	20	30	0.495
	Sig 12	38	120	10	38	1	22	-0.392
	Sig 13	51	16	120	22	21	19	-0.409
	Sig 14	6	34	87	17	50	120	1.577
	Sig 15	120	11	45	299	42	110	-4.582
Forgeries (skilled)	Sig 16	98	12	6	9	0	6	1.923
	Sig 17	100	6	0	16	4	12	2.786
	Sig 18	152	22	5	12	13	4	4.002

The negative outputs close to -5 are signatures greatly different from the genuine due to the knowledge provided by the negative synthetic examples. The neural network was further tested with more positive synthetic examples (Sig 6 – Sig 10), which were recognized as genuine. With non-skilled forgeries (Sig 11 – Sig 15), the neural network showed a good rejection; with skilled forgeries (Sig 16 – Sig 17) the neural network could reject two (according to the criteria that we will establish on the next paragraph) and was unable to reject one (Sig 18). There is a region of uncertainty in the output of the neural network to classify a signature as genuine or false, which is the region between +2 and +3. In terms of similarity from a test signature to a genuine signature, it is the region where a forgery can look greatly like a genuine signature; therefore, the decision must be made based on a threshold θ, being the effectiveness of the verifier affected by this parameter. A very low θ will permit false signatures to be classified as genuine and on the other hand, a very high θ will cause that genuine signatures be classified as forgeries; the output of the neural network is transformed to a degree of certainty that the signature is genuine to a range of 0-100 %, so the final classification is given on a basis of a parameter function type S in such a way that θ will be mapped as shown on Fig. 8, and the genuine/false verdict is also shown in such figure.

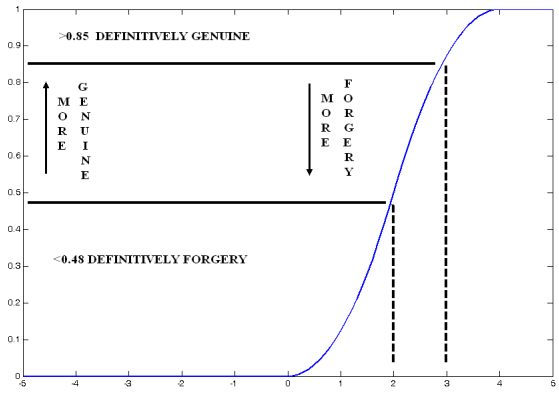


Fig. 8 Making Decision About the Genuineness of a Signature as an Output Function of the Neural Network. Courtesy of [11].

3. Optimizing the Global Execution from a Neural System of Off-line Signature Verification on Checks.

3.1 Big Data and CUDA.

The Big Data current applications require great capacity of computing, which can be combined with new programing architectures in parallel through the use of graphic cards (GPU`s) to try to improve the performance. CUDA is a platform that allows parallel programing making use of the processing power and the memory that the current video cards have (GPU`s), which have a greater number of processing nucleus compared to a CPU [3]. The CUDA platform allows us to use graphic cards not only for the processing of graphs but also for performing and processing instructions and information in a parallel way, doing the same task with different data thus reducing the processing time on operations with high arithmetic costs. We use the CUDA technology for the process of extracting the signature from the check, the creating of structural elements, training and recognizing of the signature. The process was done as a global process, and we consider each global

process as a sprint, we made 10 sprints, as can observe on Table 4.

Tabla 4. Number of Signatures by Global Process.

Sprint	Signatories	Signature by Signatory	Total
1	15	15	225
2	30	15	450
3	45	15	675
4	60	15	900
5	75	15	1125
6	90	15	1350
7	105	15	1575
8	120	15	1800
9	135	15	2025
10	150	15	2250

It is due to mention that the segmentation of the check and the mathematic morphology were done in a sequential process and only in part of the training and verification of the signature is where we did the parallel process. Table 5 shows the training times in CPU vs GPUs for the Neuronal Network Backpropagation.

Tabla 5. Training Times of the Neuronal Network Backpropagation in CPU vs GPU.

Sprint	CPU	GPUs
1	0:14:01.000	0:01:31.000
2	0:29:15.000	0:02:59.000
3	0:55:21.000	0:04:29.000
4	1:02:00.000	0:06:01.000
5	1:21:00.000	0:07:19.000
6	1:51:00.000	0:08:53.000
7	2:10:00.000	0:10:21.000
8	2:19:00.000	0:27:57.000
9	2:51:00.000	0:40:18.000
10	3:13:00.000	0:48:28.000

Fig. 9 shows the graph of training times of the signature, as we can observe on Fig. 9, the performing times on GPU`s are minor compared to those on CPU, demonstrating the applications that implement parallel process in CUDA architectures (GPU`s) are more efficient vs CPU, because the times reflect a wide distance between each architecture.

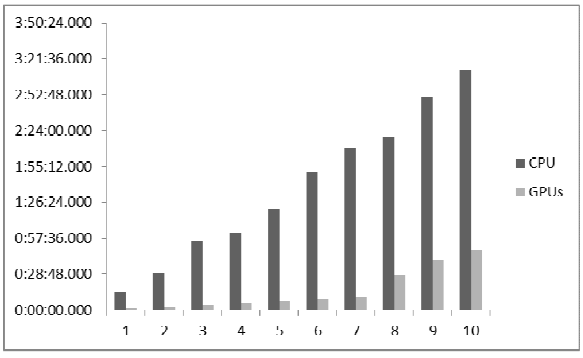


Fig.9 CPU vs GPU`s Processing Times in the Training of Neural Network.

Table 6 shows the global performing times for the process of extraction from the check`s signature, creation of structural elements, training and recognizing of the signature.

Tabla 6. Global Execution Time.

Sprint	Total of Signatures	CPU	GPUs
1	225	0:24:12.000	0:11:52.000
2	450	0:50:17.000	0:23:47.000
3	675	1:25:00.000	0:35:38.000
4	900	1:44:00.000	0:47:34.000
5	1125	2:12:00.000	0:59:57.000
6	1350	2:53:00.000	1:11:00.000
7	1575	3:24:00.000	1:23:00.000
8	1800	3:43:00.000	1:52:00.000
9	2025	4:27:00.000	2:26:00.000
10	2250	4:58:00.000	2:57:00.000

On Fig. 10 we can observe the difference of times between both implementations (CPU vs GPU`s) to find a solution for the off-line verification system of signatures on checks. Based on the results, it`s demonstrated that using GPU`s can reduce the performing time of a verification architecture of checks off-line.

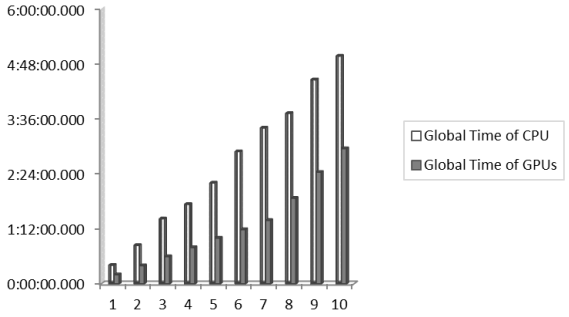


Fig. 10 Global Execution Time from a Neural System of Off-line Signature Verification on Checks.

4. Conclusions.

The current Big Data applications require great computing capacities that can be combined with new programing architectures in parallel through the use of graphic cards (GPU`s) to try to improve performance. CUDA allows us to use the graphic cards not only for the processing of graphs but also for processing and performing instructions and information in a parallel way, doing the same task to different data thus reducing the processing time on operations with high arithmetic costs, these improvements have generated a wide catalogue of applications powered by the CUDA architecture. In this article we proposed a verification neuronal system of handwritten signatures on checks off-line. Our architecture operates on two stages, the training

stage and the verification stage. The training stage is made up of various stages with the purpose of generating a neuronal network to recognize the signature. The verifying stage consists on repeating the first phases of the training stage with the purpose of extracting features from the signature. The features of the signature extracted in the verification stage are compared in the classifier against the results from the model on the training stage. The neuronal verification system of hand-written signatures off-line on checks requires great computing capacities, which can be combined with new programing architectures in parallel through the use of graphic cards GPU`s to optimize the global answering time.

5. References.

[1] Asesores Bancarios y Financieros. Cheque Bancario, Conceptos y Caracteristicas 2015. http://www.abanfin.com/?tit=cheque-bancario-concepto-y-caracteristicas&name=Manuales&fid=eh0bcab.

[2] R. Baron, and R. Plamondo. Acceleration measurement with an instrumented pen for signature verification and handwriting analysis. IEEE Transactions on Instrumentation and Measurement, 38:1132-1138, 1989.

[3] Cook Shane. CUDA Programming A Developer´s Guide to Parallel Computing With GPUs. Morgan Kaufmann 2013, ISBN:978-0-12-415933-4.

[4] J. P. Drouhard, R. Sabourin, and M. Godbout. Evaluation of a training method and of various rejection criteria for a neural network classifier used for off-line signature verification. IEEE International Conference on Neural Networks, pp. 4294-4299, IEEE World Congress on Computational Intelligence, NY, USA, 1994.

[5] B. Fang, Y. Wang, C. H. Leung, Y. Tang, P. C. K. Kwok, K. W. Tse, and Y. K. Wong. An Smoothness Index Based Approach for Off-line Signature Verification. IEEE., Proceedings of the Fifth International Conference on Document Analysis and Recognition, pp., 785-787. ICDAR, N.Y., USA, 1999.

[6] J. B, Fasquel, C. Stolz, and M. Bruynooghe. Real-time verification of handwritten signatures using a hybrid opto-electronical method. Proceedings of the 2nd.International Symposium on Image and Signal Processing and Analysis, pp., 552-557, Pula, Croatia, 2001.

[7] R. J. N. Gouvêa, and G. C. Vasconcelos. Off-line Signature Verification Using an Autoassociator Cascade-Correlation Arquitecture. IEEE Proceedings of the Fith International Conference on Document Analysis and Recognition, pp. 2882-2886, NY, USA 1999.

[8] E. J. R. Justino, F. Bortolozi, and R. Sabourin. Off-line signature verification using HMM for random, simple and skilled forgeries. IEEE Proceedings of the Sixth International Conference on Document Analysis and Recognition, pp., 1031-1034, Seattle, WA, USA., 2001.

[9] Kirk David B. and Hwu Wen-mei W. Programming Massively Parallel Processors. Second Edition Morgan Kaufmann 2013, ISBN:978-0-12-415992-1.

[10] L. Lee, and M. G. Lizárraga. An Off-Line Method for Human Signature Verification. IEEE Proceedings of the 13th International Conference on Pattern Recognition, pp. 195-198. N.Y., USA, 1996.

[11] Luna Rosas Fco. Javier, Martínez Romo Julio Cesar. Improving Dynamic Load balancing Under CORBA with a Genetic Startegy in a Neural System of Off-line Signature Verification. The 2007 International Conference on Paralled and Distributed Processing Techniques and Applications. In Computer Science & Computer Engineering, Las Vegas Nevada, USA, ISBN: 1-60132-093-0, 1-60132-094-9 (1-60132-095-7) CSREA Press, June, 2007.

[12] N. A. Murshed, F. Bortolozi, and R. Sabourin. Off-line Signature Verification, Without a Priori Knowledge of Class w2. IEEE Proceedings of the Third International Conference on Document Analysis and Recognition, pp. 191-196. N.Y., USA, 1995.

[13] Pemeena Priyadarsini M. J., Murugesan K., Rao Inbathini S., Jabeena A., Sai Tej K., Bank Cheque Authentication Using Signature. Intenational Journal of Advanced Research in Computer Science and Software Engineering, Volumen 3, Issue 5, May 2013. ISSN:2277128X.

[14] R. Plamondon and M. Parizeau. Signature verification from position, velocity and acceleration signals: a comparative study. IEEE Proceedings of the 9th International Conference on Pattern Recognition, pp. 260-265. USA, 1988.

[15] R. Plamondon and S. N. Shihari. Online and off-line handwritting recognition: a comprehensive survey. IEEE Tr ansactions on Pattern Analysis and Machine Intelligence, 22:63-84, 2000.

[16] S. A. Slyter. Forensic Signature Examination, 1 ed. Springfield, Illinois 1995.

[17] R. Sabourin and G. Genest. An extended-shadow-code based approach for off-line signature verification. IEEE Proceedings of the 12th. IAPR International Conference on Computer Vision & Image Processing, pp. 450-453. N.Y., USA. 1994.

[18] R. Sabourin, G. Genest and F. Preteux.Off-line signature verification by local granulometric size distributions. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19:976-988, 1997.

NDN-based Relational Query Processing in Ad-hoc Networks

Zhuhua Liao [1], Aiping Yi[2], Yizhi Liu [1] and Guoqing Zhang[3]
[1] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, Hunan, China
[2]School of Foreign Studies, Hunan University of Science and Technology, Xiangtan, Hunan, China
[3]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

Abstract *Several design and technical challenges arising in querying the distributed relational data as the ad-hoc network connected a host of relational databases. In this paper, we propose a relational data query scheme and the in-network query processing algorithms for dealing with the distributed relational data querying on the Named Data Networking (NDN), since the content can be retrieved by their name, but not their address, in the NDN, which can be deployed on the ad-hoc network infrastructure. Furthermore, we detailed the implementations to prove the feasibility and practicability of the NDN-based relational data query in the Ad-hoc network which possesses a distributed, dynamic, and content-rich cloud of internet-connected resources.*

Keywords: Named Data Networking; Query Translation; Name-based Routing; Ad-hoc Query Processing

1 Introduction

The mobile ad-hoc network (MANET), wireless sensor network (WSN) and self-organizing P2P network have trouble dealing with the querying of the dynamic and distributed data, since it is necessary to deal with the problems of the right and efficient routing, the dynamic distributed aggregation and the reliable response [1-4]. The routing problem has a significant effect on the query recall, precision and data acquisition when querying in the dynamic and distributed network. And the dynamic aggregation of query results and reliable response will affect the availability, reliability of the large scale data in the ad-hoc networks. However, the complex form of query will affect the personalized ability to retrieve the dynamic and distributed data. To better facilitate the complex query and open data sharing across the dynamic and distributed network, it is important to process the structured metadata of distributed databases and form certain network packets to forward all queries across the network.

For retrieving the dynamic and distributed content, a routing mechanism (we called *hierarchical name-based routing*) is proposed based on the hierarchical identifiers in the NDN (also called CCN) network [5] which can be deployed on the ad-hoc networks. The routing mechanism can identify the hierarchical classes of content. And more important it can replace the IP address with the distributed content as the basic processing unit in the routing. The basic component of the routing mechanism is the routing indexes which were built in routers according to the hierarchical name that comes from different sources. All the routing indexes, which stored in the *forwarding information base* (FIB) in routers, establish the different forwarding trees with the (*hierarchical prefix, face(s)*) form. So each user's interest can be sent to right sources through the multicast infrastructure and the data transmission is very efficient as the interest and the returned results will pass through the fastest path to or from a right source.

Facing the dynamic and distributed databases, the relational query should convey the complex requirement issued by different users on one hand, and adapt to the uncertain changes of network topology and the distributed data when the query in routing on other hand. Then, all the results can be returned and aggregated in the dynamic and distributed network. Furthermore, when a client or router forms a query it should best not be decomposed to multiple sub-queries, since the distribution of data and the forwarding path are unpredictable. The methods of current distributed query are mainly dependent on the third-party or broadcasting all semantics on the network to discover. So it is hard to retrieve the dynamic data efficiently and adapt quickly to the changes of the network topology or the varieties of data.

Just as well, the *name-based routing* mechanism can unify the manipulation of interests' routing and semantic parsing, asynchronously multicast interests, and achieve data retrieval independent of the address of data. So it can realize the semantically matching in routers and adapt to the changes of the network topology or the movement of data, and find the efficient path of data transmission. If we can explore the *name-based routing* mechanism to study a query method on the relational data networking, the complex and similar to the SQL-form query in the dynamic and distributed network may be solved. When that happens, the higher query recall and precision will be beyond the existing query, such as the probabilistic search methods [6], identifier-address mapping-based query methods [7], or the semantic-address mapping-based query methods [8].

In the paper, we will address the above challenges in the relational query processing for the NDN that deployed on the ad-hoc networks and make the main contributions as follows:

(1) We present a relational query model and query rewriting scheme in order to gear to the needs of relational query in the NDN connected with relational databases;
(2) We put forward a routing mechanism of relational query for forwarding the relational query based on the *name-based routing* mechanism in the NDN;
(3) We present the in-network query processing strategies for various relational queries in the NDN, and the aggregation algorithm for the distributed and asynchronously response results in the dynamic and distributed NDN environment.

2 Related works

Nowadays, for querying the relational data in distributed systems, the distributed DBMS can be realized in mainly three paradigms: sources with central data repository, database federation and P2P. In the central data repository paradigm, structural or semantic data are crawled from distributed data sources and put into a centralized data store. In the database federation paradigm [9, 10], data are distributed and a centralized portal is used to receive all data requests, decompose them, and finally distribute the resulting requests to appropriate data stores. But it is difficult to query the relational data distributed the dynamic network. In the P2P paradigm, the unstructured Peer to Peer system has the potential to employ more complex semantics to query data. Early days' unstructured P2P systems, as well as sensor networks and mobile networks [11 – 13], rely on flooding or random walk to locate data, which has poor scalability. In response, several methods are proposed to address the scalability issue, for example, range query [14], multi-dimensions query [15], and the routing indices (RIs) [16]. RIs provide a list of "directions" towards the potential content sources for the query. However, the "directions" information maintained in a node's indices is a list of "coarser" topics and the number of files falling into each topic summarized from its nearby neighbors, which can't fully support complex queries (e.g., queries with relational constraints). The structured Peer-to-Peer system, on the other hand, builds around the theory of distributed hash table which uses flat identifiers to store and locate data [17, 18]. Using flat identifiers, however, is not suitable for complex data queries.

Support of semantic queries over large-scale networks has attracted much attention in recent years. In the layered semantic overlay networks (SONs) [19], nodes with semantically similar contents are "clustered" together, which makes network topology dependent on the contents and causes substantial maintenance overhead. In the SQPeer middleware [20] approach, each peer has to broadcast its data schema—which includes all RDF classes and properties to (or requested by) other peer nodes to support semantic queries.

The NDN, however, mainly focuses on retrieving a single named content efficiently by issuing an interest. Zahariadis et al. [21] have presented an *Autonomic Layer-Less Object Architecture* (ALLUA) framework which assigns different types of properties to content objects. From these properties, one can know several things such as the creator of the object, its relationship with other objects, and the way it is used. But, the problem of data querying with complex constraints in the network was not under consideration. The approach of similarity content search [22] moves one step further. Based on CCN/NDN, it introduces "search" as a top level namespace and uses flooding to search similar objects in a network, but it does not tackle the complex queries. The relational routing scheme [23] is devoted to routing the semantic query and aggregating the relevant content for querying the semantic data in the NDN, but it does not consider the relational data query in the NDN.

In short, although there are various distributed query techniques and systems, almost none of them were committed to deal with the relational data query in the dynamic and distributed network (e.g. MANET, WSN, mobile P2P and so on). But, this capability is becoming increasingly important as network is more and more considered as a large distributed data store with rich relationships data than a simple communication medium.

3 The view of relational query in ad-hoc networks

3.1 The feasibility of relational query based on NDN

Generally, the attributes related to data objects will be defined in a database and some hierarchical categories can be extracted from these attributes [24-26], and most of the relational data can be organized in hierarchies [27]. Therefore, if taking the hierarchical categories (or taxonomy) as the hierarchical routing indexes, the relational query could be implemented in the NDN based on the *hierarchical name-based routing* mechanism. An implementation scheme of the relational query is showed in Fig.1. The detailed thoughts are as follows: (1) Firstly, extracting the hierarchical categories (or taxonomy) information about data entity from relational databases in each source. (2) The information is then used as the routing indexes and spread them to routing nodes, in order to serve as the forward information (stored in FIB) in NDN. Since the *interest* can be forwarded to the right sources in NDN by using the hierarchical name, the query request may also be forwarded to the right sources by using the hierarchical information. An approach for describing the query is to appropriately extend the *interest* request. The main idea is to extract the hierarchical information and other attribute constraints from the user's demand for forming the relational query. If a query statement contains the hierarchical categories pertained to the desired data of the user, and carry other attribute constraints, the query can be forwarded to all related data sources by matching the routing indexes. (3) Finally the local query processing can be performed so as to get relevant relational data that meet user's requirements.

3.2 Data mode

In a relational database, the table is the mainly form of metadata structure. Generally the range of fields' value and other constraints will be stated by a user. Each table contains one or more data attributes in fields and each record contains a unique data entity. Besides, the link structures (primary key

to foreign key relationships) often indicate the hierarchy of categories (or classes) of data objects. Normally, the hierarchical information can be obtained even if the relational data set vertically partitioned to multiple slices. Therefore, in a given application domain, the relational database established in every source essentially includes the 3 types of information: (1) things (or concepts); (2) the classification system, usually it gets from the hierarchical attributes or the link structures; (3) the instances belonged to a class.

Besides the above information, we can designate the range of wanted data and other attribute constraints in a query request. In addition, we can extract the hierarchical indexes and attribute constraints from the user's requirement, and this information can be used for routing and distributed query processing in the NDN. For example, if a NDN network stored a lot of electronic books, and the query request is "query all books that pertain to 'network engineering' and published in the past 3 years". The query with SQL-form is:

Select book name, author, publishing house, publishing time From booktable Where subject = " computer" and specialty = " network engineering" and (2013- Year(publishing time))<=3

The principal information can be extracted from the above SQL-form query statement:

Things: book**;**

Classification system: subject;

Hierarchical identifier: "computer / network engineering /…/";

Attributes' constraints: "(2013- Year(publishing time))<=3".

3.3 Relational query scheme

Basically, we can use the above information to constitute a query packet, and then forward it by the *hierarchical name-based routing* mechanism. So, provided we can extract the information (the *hierarchical information* and *attributes' constraints*) from a query, we may perform the query in the NDN which is deployed on the existing network infrastructure, such as the MANET, WSN and so on. Fig.1 shows the scheme of relational query in NDN. When a query packet (NDNQL) is forwarded to all related data sources by using the Longest Prefix matching (LPM) with FIB, the distributed query results (packaged into NDNRS) will be returned via the original query routing path, and the results aggregation will be performed, which mainly includes the fields aggregation and the records aggregation, in the convergence nodes. For example: if there are results backtracked by using the information of PIT (Pending Interest Table) in NDN from different sources that stirred by a query packet, the results can be aggregated and then returned to the user.

4 The query translation for NDN

Whether the syntax of query for NDN can describe these operations determines the description ability and flexibility of the relational query based on the *name-based routing* scheme. The main query operations in relational database are: SELECT, JOIN, PROJECT, AGGREGATION (e.g. SUM), etc. If a relational query involves multiple concepts, the equivalent NDNQL is necessary to define the query which involved one or more routing indexes for discovering all the data sources related to every concept of them.

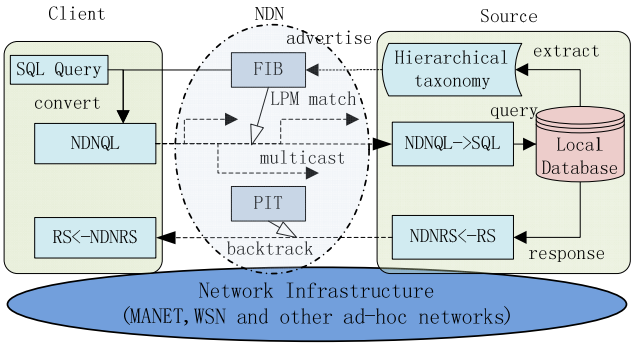


Fig1. The scheme of relational query in ad-hoc networks

4.1 The queries involved with a single concept

Here, we consider only one hierarchical identifier as the routing index for the queries. For the basic SQL query statement it mainly involves only a single table or concept.

What means of each clause in the general form of the SQL statement are:

"**Select** list<attribute name>": the clause declares the attribute value information that should be returned;

"**From** table<table name>": the clause specifies the response information that belongs to the category of the concept (table name);

"**Where** <expression list>": the clause points out the constraints for selecting the needed relational data (or records), such as restricting the category, partial name (the uncertain part uses the wildcard to match any character), as well as other attribute values (such as time, author, etc.) of relational data;

"**Group by** <name>": this clause is to classify the returned data for aggregation (or Grouping);

"**Having** <conditional expression>": the clause states the filtering calculation for grouping results according to the conditional expression. Generally, the processing needs to delay at the last convergence node.

In view of the semantic information of the SQL query statement, we can define a formalized query for the query processing in the NDN by using the *hierarchical name-based routing* mechanism. That is, the query can consist of two parts: *constraint factor* and *routing factor*. The routing factor is used to perform the correct and fast routing for the relational query and the results back in the dynamic and distributed network, while the constraint factor, combining with the routing factor, is used to declare the local query and the operations of response results aggregation. So we can form the NDNQL query packet for performing in NDN as follows:

The format of the NDNQL query used in NDN is " hierarchical identifier (plus partial name) + constraints". The "hierarchical identifier (plus partial name)" describes the hierarchical category which includes the entities what the user

wants to query, even specifies the partial name of these entities. The "constraints" is an attribute expression.

That means for processing a relational query in NDN we should extract three types of information, which are routing factor, constraint factor and the form of returned results (response). As following, we will present how to convert the SQL-form query into the above 3 types of information for forming the equivalent NDNQL.

For example: an aggregation (sum) query is that summarizing the sales of various commodities (ProductID) in line with area (AreaID). The SQL query is:

```
Select AreaID,ProductID,Sum(Total)
From CW_Orderdetail
Where ostate=1
Group By AreaID With cube
```

The SQL-form query can be converted into a query packet for NDN, in which the main 3 types of information are:

Type	*Information*
Routing Indexes	AreaID/CW_ Orderdetail
Constraints	ostate=1
Response	AreaID,ProductID,***Sum***(Total); ***Group By*** AreaID ***With*** cube

4.2 The queries involved with multiple concepts

When a relational query involves more concepts (or tables), and the concepts associated the data that across multiple data sources, the query should be sent to each data source where stored one or more concepts (tables) of them and then response the relevant results, after that the returned results need to be aggregated in the convergence nodes. So there are multiple hierarchical identifiers extracted from these concepts that are used to route the query to the corresponding data sources in which stored the relational data and contained one or more concepts aforementioned. Below we will deal with the problem of the relational query transformation which involved with multiple concepts.

(1) Join query in SQL-form:

```
Select customer-name, borrower.loan - number, amount
From borrower, loan
Where borrower.loan-number=loan.loan-number
```

The SQL-form query can be rewritten into the formalized query for NDN, and the main 3 types of information are:

Type	*Information*
Routing Indexes 1	the hierarchical identifier of the "borrower"
Response1	customer-name, loan-number
Routing Indexes 2	the hierarchical identifier of the "loan"
Response2	loan-number, amount
Constraints	borrower.loan-number=loan.loan-number

To translate the query with union, intersect or except operation, which also involved with multiple concepts, is used in the same manner.

(2) Nested query in SQL-form:

```
Select CustomerID From Sales.Customer Where TerritoryID = (Select TerritoryID From Sales.SalesPerson Where SalesPersonID = 276)
```

The SQL-form query can also be rewritten into:

Type	*Information*
Routing Indexes(external)	the hierarchical identifier of the "Sales.Customer"
Response(external)	CustomerID
Routing Indexes(internal)	the hierarchical identifier of the "Sales.SalesPerson"
Response(internal)	TerritoryID
Constraints (external)	TerritoryID = (**Select** TerritoryID **From** Sales.SalesPerson **Where** SalesPersonID = 276)
Constraints (internal)	SalesPersonID = 276

For extracting hierarchical identifiers from relational databases, the name of concept should be identified and extracted from the tables of local relational database firstly, and then the hierarchical relationship between concepts can be extracted from the connections between tables or from the attribute fields in the tables. This work can be accomplished by DBAs or other authorized users, or we also can predefine a unified classification criterion before setting up the distributed databases and ask users to identify all added concepts and entities according to the classification criterion when they import fresh data.

5 Implementation

For transmitting the relational query, the principal information, mainly includes *routing indexes*, *response form* and *attribute constraints* that extracted from the SQL-form query, and the security parameters (e.g. authenticating and authorizing information) and other processing constraints added by the user or the client, for example, limits the type of sources or forwarding count, specifies whether or not aggregating results and so on, should be packaged into a network packet (called NDNQL). The format of the NDNQL is shown in the Fig.2. Each relevant query processing node will receive the packet and then resolve it for query processing. The response results (called RS) will also be packaged into a response packet (called NDNRS, its format is shown in the Fig.3) by sources. In the section, we will introduce the implementation scheme of the NDLQL processing to some specified types of SQL-form query.

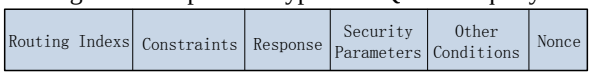


Fig.2 NDNQL: Query Packet

Fig.3 NDNRS: Response Packet

5.1 The in-network processing for simple query

The simple query mainly refers to the query that involves a single concept. For example, the basic relational operation, such as projection (e.g. $\pi_{a1,\ldots,an}(R)$) and selection (e.g. $\sigma_{\varphi}(R)$), mainly involves a single concept (generally it was named relation in DBMS). For this type of query, we have given how to define its routing factor (refers to the routing indexes) and constraint factor (mainly refers to the attribute constraints or the other selection conditions) before. To implement this type of distributed query in the NDN, it needs a self- organizing processing, which mainly includes two parts: the self-organizing routing processing and the self-organizing query processing. Fig.4 shows the relational query processing for a single concept in the NDN.

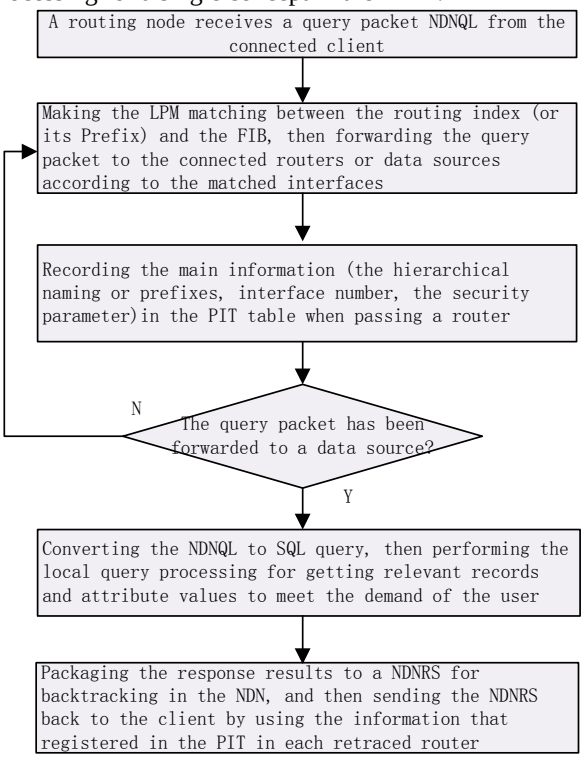


Fig.4 The processing of the query that involving single concept in NDN

(1) The self-organizing routing processing

For implementing the self-organizing routing of query, the processing steps are as follows: Firstly, making the LPM matching between the hierarchical identifier in query packet and the prefixes in FIB, and then forwarding the query to the next routing nodes according to the interface value of the matched records in FIB. In every routing node that received the query, repeating the above steps, until the query is forwarded to all relevant data sources. In addition, after local querying in each data source node, it needs to return the results (or the aggregated results processed in a convergence node) to the user who issued the query through the path that the query has passed through.

(2) The self-organizing query processing

The self-organizing query processing mainly includes the local query processing and the results aggregation processing. To the local query processing, the query should be rewritten into the SQL- form in the data source that received the query, and then performing the SQL- form query in the local database.

The aggregation processing should be performed in the convergence nodes for aggregating the returned results. According to the characteristics of the returned results, the aggregation processing can be divided into two kinds:

One is the fields aggregation (or vertical aggregation), that is, performing the aggregation of the different returned values of fields (or attributes) that are distributed different data sources, but the duplicates of attribute-value pair should be removed in each record.

The other one is the records aggregation (or horizontal aggregation), that is, performing the aggregation of the different records that satisfied the conditions of a query and returned from the relevant data sources, but the redundant records should be eliminated.

5.2 The in-network processing for complex query

The complex query mainly refers to the query that involves multiple concepts. To the **Union**, **Intersect and Except** operation, it should be performed in a distributed and self-organizing mode, that is, performing the **Union**, **Intersect or Except** operation multiple times at different source nodes or convergence nodes for returning the local results. The Fig.5 shows the overall processing of the query that involving multiple concepts in NDN. A simple approach is to route and forward each hierarchical identifier to relevant data sources, and then to aggregate the distributed response results and return the aggregated results back to the user.

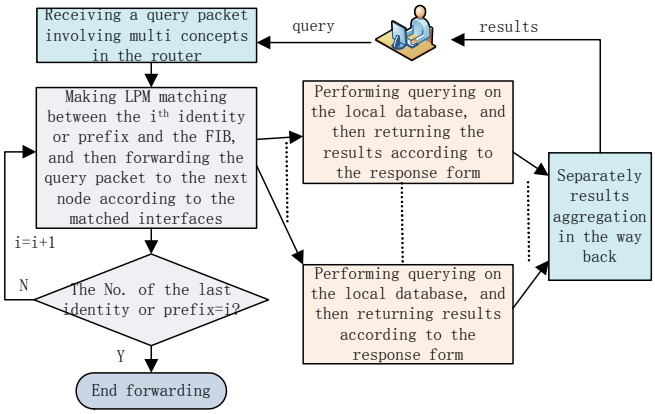


Fig. 5 The processing of the query that involving multiple concepts in NDN

But we should to make an exception in some particular queries. The nested query is a query that should be treated specially. The processing scheme of nested query is shown in Fig.6. If a user issues a nested query to the NDN, the query packet will have multiple hierarchical identifiers or prefixes. The specific processing steps of the nested query are:

(1) Set the innermost layer as the query i, and take the query i as a single query;

(2) Judging the number of concepts that the i^{th} layer query involving in is whether more than one. If it is, processing this query according to the processing of the query that involving multiple concepts in NDN, otherwise processing this query according to the processing of the query that involving a single concept in NDN;

(3) When receiving the i^{th} layer results, determining whether the i^{th} layer query is the outermost layer query. If it is, turn (5), otherwise turn (4);

(4) According to the i^{th} layer results, rewriting the $(i+1)^{th}$ layer query, and treating it as a query packet, then increasing the i by 1, and turn (2);

(5) Returning the results of the i^{th} layer query back to the user.

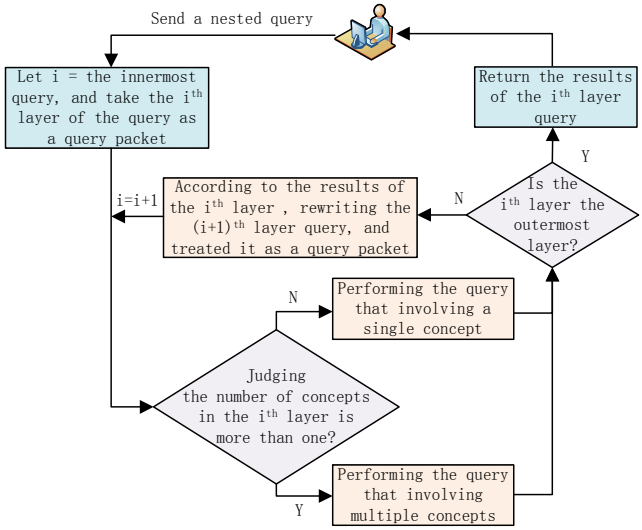


Fig. 6 The processing of Nested query in NDN

5.3 The in-network processing for results aggregation

The SQL language provides five predefined aggregation functions: AVG, MIN, MAX, SUM and COUNT. To perform the aggregation function, we can use a recursive method with multiple-step aggregation operations in the dynamic and distributed network. As for the AVG aggregate function, the average calculation can be repeatedly to all the local average values that responded from different neighbor nodes, but it should be considered the different weights to different local average values for the local average values that contained different number of the original data.

5.5 Performance analysis and Application

To the aforementioned relational query processing scheme, the performance of its distributed processing determines the availability and suitable application scenarios. In this section, we analyze the performance characteristics of the distributed processing for the NDNQL.

(1) The recalling rate to the NDNQL query. The recalling rate is determined by the query routing model, which affects the query whether can be forwarded to all right sources. Once the query is forwarded to all right sources, all relevant results can be returned to the user. In fact, the query routing model is determined by the LMP matching of NDN routing model. Many researches or experiments [5,28-29] confirm that the NDN routing model can be forwarded the query to nearly all right (and even mobile) sources.

(2) The precision of NDNQL query. It is determined by the semantic constraints of NDNQL query. So long as we defined clearly the semantic constraints, high precision of NDNQL query can be achieved.

(3) The cost to the asynchronous and distributed results aggregation. In the results way back, the aggregation operations were dispersed along the multicast tree of the query forwarding. So the cost is determined by the times of aggregation operations of one farthest result in the way back. If we can neglect filtering the duplicate results in the way back, we can delay the aggregation to the client for the user receives the separately results early.

The relational query processing scheme we proposed can be applied in the following scenarios: (1) For querying the relational data on the MANET databases, we can use the name-based routing mechanism to perform its routing and forwarding. So the query can be forwarded to the right sources and the local query results can be returned to the client via the routing mechanism. (2) For accessing the information in the ad-hoc Vehicle-to-Vehicle (V2V) network, we also can implement the relational data query in the ad-hoc network based on the NDN naming scheme [30] and query processing scheme we proposed on the V2V network. (3) At present, the data-centric routing techniques [31] can only apply to the simple information query or complex semantic query with inefficiency on WSNs. Our scheme can help to perform the relational data query and response results in the WSNs, after named the hierarchical data in sensors and deployed the NDN on the whole WSNs.

6 Conclusions

In this paper, we present a relational query scheme and the implementation methods for the relational data query in the ad-hoc network (e.g. MANET, WSN) on which deployed the Named Data Networking. In particular, for realizing the new relational query, the method of the relational (SQL-form) query transformed into the query form that suited the targeted forwarding of the query in the NDN by using the *name-based routing* mechanism, and the processing capabilities of the new in-network query in the dynamic and distributed network are all proposed. Furthermore, as for the plethora of distributed local query results, this paper also presented the distributed aggregation tactics for aggregating all relevant and distributed query results in the NDN network. In the future, we will be dedicated to the software development and applications of the relational query in practical ad-hoc network environment.

7 ACKNOWLEDGMENTS

This work was supported in part by the project supported by the National Natural Science Foundation of China (Grants

No. 61370227), the Hunan Provincial Natural Science Foundation of China (Grant No. 13JJB004) and Hunan Province Universities Innovation Platform of Open Fund Project (Grant No.14K037).

8 References

[1] A. Eyal, A. Gal. "Self Organizing Semantic Topologies in P2P Data Integration Systems", International Conference on Data Engineering (ICDE), March 29- April 2, 2009.

[2] Samuel R. Madden, Michael J. Franklin, et al. "TinyDB: an acquisitional query processing system for sensor networks". ACM Trans. Database Syst. Vol.30, No.1, 122-173, 2005.

[3] Mario Gerla, Leonard Kleinrock, "Vehicular networks and the future of the mobile internet, Computer Networks", Vol.55, No.2, 457-469, 2011.

[4] P.Padmanabhan, L.Gruenwald, et al. "A survey of data replication techniques for mobile ad hoc network databases". The VLDB Journal, Vol.17, No.5, 1143-1164, 2008.

[5] V. Jacobson, D. K. Smetters, et al. "Networking named content", the ACM Conference on Emerging Networking Experiments and Technologies, New York, USA, Dec. 2009.

[6] D.Tsoumakos, N.Roussopoulos. "Adaptive probabilistic search for peer-to-peer networks", International Conference on Peer-to-Peer Computing (P2P), 1-3 Sept. 2003.

[7] M. Harren, J.M. Hellerstein, R. Huebsch, et al. "Complex Queries in DHT-Based Peer-to-Peer Networks". Int'l Workshop Peer-to-Peer Systems (IPTPS), 2002.

[8] D. Fayel, G. Nachouki and P. Valduriez. "Semantic Query Routing in SenPeer, a P2P Data Management System". NBiS 2007, LNCS 4658, 365–374, 2007.

[9] L.M.Haas, E. T. Lin, andM. A. Roth. "Data integration through database federation", IBM Systems Journal, vol. 41, No. 4, 578 – 596, 2002.

[10] G. Olaf and S. Steffen. "Federated data management and query optimization for linked open data", New Directions in Web Data Management, vol. 331, 109 – 137, Springer, Berlin, Germany, 2011.

[11] Gnutella, http://gnutella.wego.com/.

[12] J.N.Al-Karaki andA. E.Kamal. "Routing techniques in wireless sensor networks: a survey", IEEE Wireless Communications, vol.11, no. 6, 6 – 27, 2004.

[13] M. Meisel, V. Pappas, and L. Zhang, "Ad hoc networking via named data", the ACM International Workshop on Mobility in the Evolving Internet Architecture, ACM, New York, USA, 2010.

[14] F. Banaei-Kashani and C. Shahabi. "Swam: a family of access methods for similarity-search in peer-to-peer data networks", ACM Conference on Information and Knowledge Management (CIKM), New York, USA, Nov. 2004.

[15] X. Sun. "SCAN: a small-world structured p2p overlay for multidimensional queries", International World Wide Web Conference, New York, USA, May 2007.

[16] A. Crespo and H. Garcia-Molina. "Routing indices for peer to peer systems", International Conference on Distributed Systems, July 2002.

[17] I. Stoica,R.Morris, D.Karger, et al. "Chord: a scalable peer-to-peer lookup service for internet applications", ACM SIGCOMM, San Diego, Calif, USA, Aug. 2001.

[18] S. Ratnasamy, P. Francis, M. Handley, et al. "A scalable content-addressable network", ACM SIGCOMM, San Diego, Calif, USA, 2001.

[19] A. Crespo and H. Garcia-Molina. "Semantic overlay networks for p2p systems", in Agents and Peer-to-Peer Computing, LNCS, vol. 3601, 1 – 13, Springer, 2005.

[20] G. Kokkinidis and V. Christophides. "Semantic query routing and processing in p2p database systems: the ics-forth SQpeer middleware", Proceedings of the Current Trends in Database Technology Workshops, vol. 3268, LNCS, 433 – 436, Springer, 2005.

[21] T. Zahariadis, P. Daras, J. Bouwen et al. "Towards a content centric internet", in Towards the Future Internet CA European Research Perspective, IOS Press, 227 – 236, 2010.

[22] P. Daras, T. Semertzidis, L. Makris, and M. G. Strintzis, "Similarity content search in content centric networks", ACM International Conference on Multimedia ACM Multimedia, 775 – 778, New York, USA, Oct. 2010.

[23] Zhuhua Liao, Guoqiang Zhang, Aiping Yi, and Guoqing Zhang. "A Relation Routing Scheme for Distributed Semantic Media Query", The Scientific World Journal, vol. 2013, 2013.

[24] Chakrabarti S, Dom B, Agrawal R, et al. "Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies". The VLDB journal, 7(3), 163-178, 1998.

[25] Han J, Fu Y. "Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases", KDD Workshop, 157-168,1994.

[26] W. Dakka, P. G. Ipeirotis. "Automatic extraction of useful facet hierarchies from text databases". IEEE International Conference on Data Engineering (ICDE), 2008.

[27] D. Martinenghi, R. Torlone. "Taxonomy-based relaxation of query answering in relational databases", The VLDB Journal, 466-475, January, 2014.

[28] V. Jacobson, D. K. Smetters, et al. "VoCCN: voice-over content-centric networks". Proceedings of the workshop on Re-architecting the internet. New York, USA, 1-6, 2009.

[29] G. Xylomenos, C. N. Ververidis, et al. "A Survey of Information-Centric Networking Research", Communications Surveys and Tutorials, Vol. 6, No. 2, 1024–1049,2014.

[30] L.Wang, R.Wakikawa, R.Kuntz, et al. "Data naming in Vehicle-to-Vehicle communications," IEEE Conference on Computer Communications Workshops, 328-333, 25-30 March 2012.

[31] N. Vats Doohan and S. Tokekar. "A Survey on Routing Techniques of Data-Centric Wireless Sensor Networks". International Journal of Computer Applications, 53(16), 1-5, Sept. 2012.

SESSION

HARDWARE AND SOFTWARE RECONFIGURABILITY + NETWORK ON CHIP (NOC) AND EMBEDDED SYSTEMS

Chair(s)

TBA

A More Efficient Use of Separable Allocator With Bypass Buffer

Chung-Da Wu, Yarsun Hsu and Kuo-Feng Liao
Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan

Abstract - *As more processors are integrated into a single chip, a robust interconnection network between them is becoming more important consequently. Allocator, a critical component in a router, plays a key role in determining which flit would be delivered forward through crossbar switch. As one kind of allocator, separable allocator is a common choice for its simple structure and low cost. However, its simple operation inevitably leads to lower matching quality. To address this problem, bypass buffer is therefore introduced in this research to enhance the utilization of the unused ports induced by the inefficiency of separable allocator. According to the performance and cost evaluation by simulation tools, a design with bypass buffer will get most benefit and bring less cost percentage when there are more virtual channels in a router. Moreover, the more balanced a traffic pattern is, the more bypass buffer could be utilized.*

Keywords: NoC, Allocator, Separable Allocator

1 Introduction

With the development of semiconductor technology, integrating all the components of an electronic system into a single chip is an obvious trend in the future [1]. Owing to the increasing number of IP cores in a system, how to build a stable and reliable interconnection network between each core is an important issue in recent years. As a result, a theory on NoC (network on chip) [2] has been proposed to achieve better performance and higher scalability than the traditional network based on bus or crossbar. Thus, how to enhance the performance of NoC is becoming increasingly important.

To improve the performance of NoC, different aspects of a network can be focused on, such as topology, routing, flow control and router architecture [3] [4]. For router architecture, it is the organization of each router in a network. Among all the components in a router, switch allocator is a key unit which determines how each flit passes through the crossbar switch to its corresponding output port. To enhance the effectiveness of a switch allocator, the main purpose of this study is to examine the impact of switch allocator on the network performance, and a reformed allocator is proposed in comparison to its original design.

For a switch allocator, the matching quality is a critical factor which directly affects the time a flit has to wait in the input port buffer of a router. This time is further responsible for the packet transmission latency. Moreover, a bad switch allocator could easily lead to network congestion because packets cannot be transmitted to their destination nodes rapidly.

To design an allocator, there is always a trade-off between matching quality and delay. Separable allocator is a common allocator with both low delay and small area, but the produced matching quality is not very high. On the other hand, wavefront allocator [5] could attain better matching quality but is accompanied by higher cost.

The objective of this research is to address the low matching quality problem of separable allocator by adding some hardware units to the prototype router. The remainder of this research is structured as follows: Section 2 provides a brief introduction to allocation and separable allocator. Section 3 describes the functionality of bypass buffer, which are some additional components used to improve traditional separable allocator. Section 4 carries out simulations in various circumstances to examine the effect of bypass buffer. Section 5 deals with the cost evaluation of bypass buffer. Finally, conclusion is presented and some suggestions are made for future research in Section 6.

2 Related Work

2.1 Separable Allocator

Separable allocator is one kind of allocator which separates the allocation into two arbitration parts. In the first part, the five arbiters are used for each corresponding input port to determine which virtual channel [6] would be selected to proceed its request to the second stage of arbitration. Next, in the second part, if more than one request bid for the same output port, the five output arbiters are used to make sure that only one request would be granted. Through the allocation, which flits could utilize the crossbar switch to their corresponding output ports is therefore decided.

2.2 The Pros and Cons of Separable Allocator

In comparison with other allocators, because of the relatively simple structure and much lower delay and power consumption [7], separable allocator is a typical choice for allocation. Nevertheless, the lack of coordination between the two arbitration stages inevitably results in some side effects that the number of generated matching pairs is relatively lower.

One possible solution to this problem is to perform many iterations to allocate the unused resources again [8]. Though this method enhances the matching quality, the cost of area and delay is increased considerably as a consequence. One another common solution is to use wavefront allocator [9],

which can concurrently take the two arbitration parts into consideration. Although the matching quality is therefore improved, the cost of area and power for this allocator is about two times larger than separable allocator under most circumstances [7]. The price of wavefront allocator is still too high to be the best choice.

To solve the dilemma of hardware cost and matching efficiency, in the next section, a method of bypass buffer will be introduced. With this additional hardware, a router with separable allocator could thus achieve better performance, while the cost of this structure is relatively lower.

3 Bypass Buffer

3.1 Architecture

Figure 1 is the architecture of bypass buffer. As can be seen in this figure, many extra hardware units are added to the original router. The five demultiplexers next to each input port are used to choose where the flits in the input port buffers should be transmitted. Either the crossbar or the bypass buffer queue is a possible choice. Similarly, the five multiplexers next to each output port are responsible for selecting the place from which the flits are received. Likewise, an output port could accept flit from either crossbar or bypass buffer queue. To determine which bypass buffer queue a flit should be placed, the five demultiplexers in front of the five arbiters are essential, and the five arbiters are used to deal with the situation that several flits are sent to the same bypass buffer queue. Since each bypass buffer queue has its own corresponding output port, the numbers of bypass buffer queues and output ports are equal. The depth of each bypass buffer queue is assumed to be one for now, and it will be modified in the simulation later.

3.2 Operation

After performing the switch allocation by separable allocator, those ports failed in the allocation would be inactive in the switch traversal stage. Bypass buffer is thus trying to utilize these idle resources of ports. Even though an input port has failed in the allocation, it still has selected one virtual channel from all the virtual channels it contains. According to the output port destination of this virtual channel, we would know to which bypass buffer queue the flit in this virtual channel should be directed because each bypass buffer queue corresponds to only one output port. Since the depth of bypass buffer queue is assumed to be one, only when the corresponding queue is empty will the input port attempt to deliver flit to this bypass buffer queue. While those matched input ports transmit flits through crossbar, those input ports not granted in allocation would send their flits to the bypass buffer queues if empty.

Take Figure 2 for example, assuming the bypass buffer queue 1 and 4 are empty, the failed input port 2 and 4 will send their flits to the bypass buffer queue 4 and 1 respectively, and accordingly utilize those inactive input ports.

With regard to the output part of bypass buffer, it is trying to utilize the inactive output ports. The flit in each bypass buffer queue would be sent to the corresponding output port which is not matched in the allocation. After sending out a flit, the bypass buffer queue would be empty and can receive a new flit again.

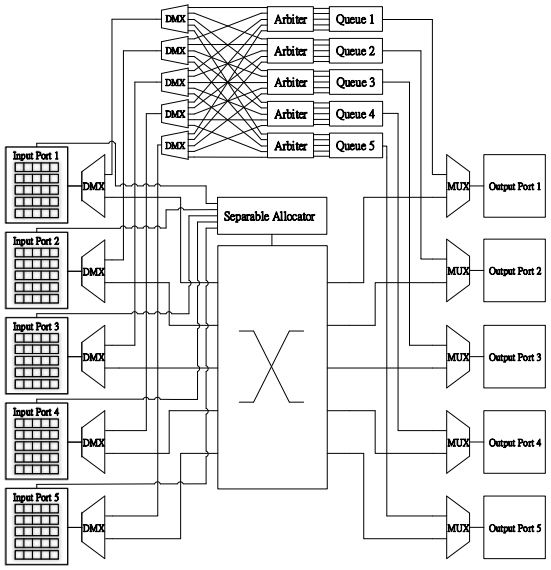


Figure 1: The architecture of bypass buffer

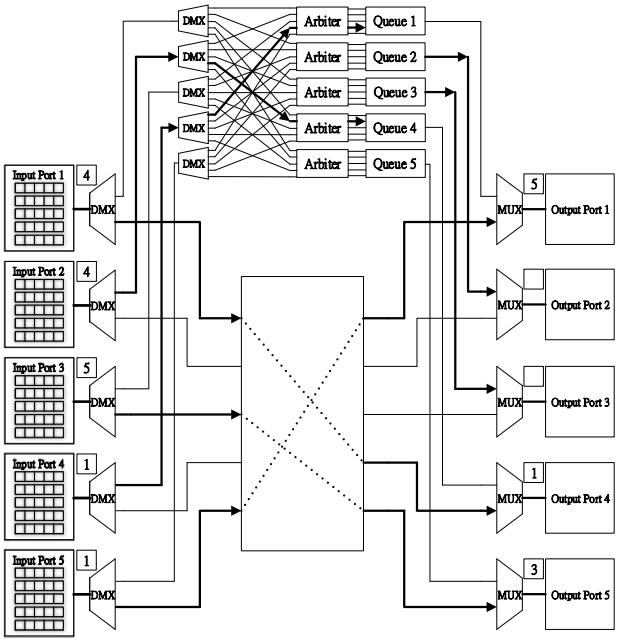


Figure 2: A bypass buffer example

However, the newly received flit in the bypass buffer queue cannot be delivered immediately. Instead, it would have to wait until next cycle and check if its corresponding output port is idle before been sent out. That is, it takes at least two clock cycles for a flit to get to the output port through bypass buffer, since the input part and output part of bypass buffer are operated independently.

This part of operation can also be observed in Figure 2. Under the circumstances that the bypass buffer queue 2 and 3 both contain flit, the originally inactive Output Port 2 and 3 will receive flits from their corresponding bypass buffer queues, and the idle resources of output ports are therefore utilized.

During the switch traversal stage of router pipeline, the output part of bypass buffer will be executed first before the input part. Thus, bypass buffer could be read and written in one cycle.

3.3 Restrictions

As described above, the flits in bypass buffer could only be sent out when their corresponding output ports are unused. As a consequence, the ordering of the flits within a packet may change once part of these flits passes through bypass buffer. Nevertheless, the ordering could be restored when a packet arrives at its destination node through the sequence number contained in each flit. However, since the head flit and the tail flit are always the first flit and the last flit of a packet, if they are involved in the reordering, a packet would not be able to arrive at its destination successfully. To prevent this from happening, some restrictions must be imposed on bypass buffer. Three possible issues are listed below:

Issue 1: The body flit outpaces the head flit in the same packet

Issue 2: The tail flit outpaces the body flit in the same packet

Issue 3: The head flit of a packet outpaces the tail flit belonged to the former packet

The issue 1 will happen when a body flit has reached the output port through crossbar while the head flit of the same packet remains in bypass buffer. The body flit would replace head flit to become the first flit in the packet. Because a body flit does not contain any routing information, this packet could not proceed anymore. To avoid this fault, a head flit would never be allowed to use bypass buffer.

The second issue happens when a tail flit has left a router while the body flit of the same packet is still stuck in bypass buffer. As a result, this packet would lose one body flit. To address this problem, a tail flit will always be sent to bypass buffer even though it has succeeded in the allocation. By exchanging the tail flit for body flit in bypass buffer, a tail flit would always be the last flit of a packet. Since these two flits belong to the same packet, they have the same output port destination, and the preserved output port for the granted tail flit would otherwise be utilized by the body flit in bypass buffer.

The third issue happens when a packet outpaces its former packet by crossbar with one flit of the former packet still in the bypass buffer. The flits of two different packets would be mixed up on account of bypass buffer. To solve this problem, a method primitively used to prevent deadlock is applied here.

In most cases, to reuse a virtual channel more quickly, a virtual channel can be reallocated to another packet if the currently occupying packet has left this router. However, if part of this leaving packet is still in the downstream router, any packet which successfully gets the just released virtual channel might appear simultaneously with the former packet in the input port buffer of the downstream router. A dependency relation between two packets is thus created, so a deadlock situation may occur in the network.

An approach to this deadlock problem is to wait to reallocate a virtual channel until receiving the credit from the leaving packet. This could make sure that the leaving packet has left the downward router, and accordingly the possible dependency relation between two packets is broken. As a result, two different packets would never coexist in the same virtual channel buffer, and the third issue mentioned above is avoided by using this method.

4 Performance Evaluation

4.1 Simulation Methodology

Booksim, a cycle-accurate simulator, is designed to evaluate the performance of NoC architecture, as it could accurately model the behavior of network components [10]. With this simulator, the performance of the bypass buffer design is compared with that of the baseline, a prototype router with separable allocator.

The router pipeline used here is a four-stage pipeline, and a credit-based flow control is adopted. Table 1 lists some parameters used in the simulation, and some of them including traffic pattern, virtual channel number, packet size and bypass buffer size will be changed later according to different simulation objectives.

4.2 Traffic Patterns

In this section, several different synthetic traffic patterns are used to evaluate their impacts on performance. Besides uniform traffic, which will be used in all the rest simulations of this section, five other traffic patterns are also introduced. Their behaviors are described in Table 2. Given the source node, the destination node of each traffic pattern could be determined by this table. The S_i and D_i represent the i^{th} bit of the source and destination address, while the S_x and D_x denote the x^{th} radix-k digit of the source and destination address. The number b is the bit length of an address and it is computed by $b=\log_2 N$, where N is the number of nodes in the network. Since the network to be simulated contains 64 nodes, the number b is calculated as 6.

According to this table, all the traffic patterns except for uniform traffic have a fixed destination node. By contrast, though generated in the same source node, the packets of uniform traffic may be sent to different destinations.

The simulation results are presented in Figure 3. Because uniform traffic could produce a balanced packet distribution in

Table 1: The network parameters

Topology	8-ary 2-cube Torus
Number of Router Ports	5
Routing Function	Dimension-order
Virtual Channel Allocator	Separable Allocator
Arbiters of Separable Allocator	Round-robin Arbiter
Buffer Size of Virtual Channel	5 Flits
Traffic Pattern	Uniform Traffic
Number of Virtual Channels per Port	15
Number of Flits per Packet	10
Queue Depth of Bypass Buffer	1

Table 2: The descriptions of traffic patterns

Traffic Patterns	Behavior Descriptions
Uniform	**From each source, there would be equal amount of traffic sent to each destination**
Bit Complement	$D_i = \neg S_i$
Bit Reverse	$D_i = S_{b-i-1}$
Shuffle	$D_i = S_i - 1$
Transpose	$D_i = S_{i+b/2 \bmod b}$
Tornado	$D_x = S_x + \lceil k/2 \rceil - 1$

the network, it has the largest throughput. On the contrary, owing to the fixed destination node, the five other traffic patterns would result in congestion easily at higher injection rates, so their throughputs are relatively lower.

Since bypass buffer could be fully utilized when many requests exist simultaneously during an allocation, and this situation is more likely to happen in a traffic that could generate a more balanced network load. As a result, bypass buffer has improved the baseline design most in uniform traffic.

As could be observed in the figure, except for uniform traffic, the improvement of bypass buffer is more visible with bit reverse traffic and tornado traffic. Despite not being worse, bypass buffer is unable to lead to obvious performance enhancement with the other three traffic patterns. This could be explained by the characteristics of different traffic patterns. For example, for bit reverse and tornado traffic, in each router, the amount of input ports into which flits would be injected is higher than the three other traffic patterns. Consequently, their behaviors are much like a uniform traffic, and accordingly the improvement that bypass buffer bring about is more apparent. However, for the other three traffic patterns, some input ports are always left unused; therefore, bypass buffer could not be fully utilized, and the improvement is nearly invisible.

4.3 The Size of Packet

In this section, the size of each packet varies to examine its influence on the performance. As can be observed in Figure 4, when the packet size is increased, the network become congested more quickly. This is because a larger packet is more likely to spread over many routers; besides, the occupied virtual channels could not be used by other packets even

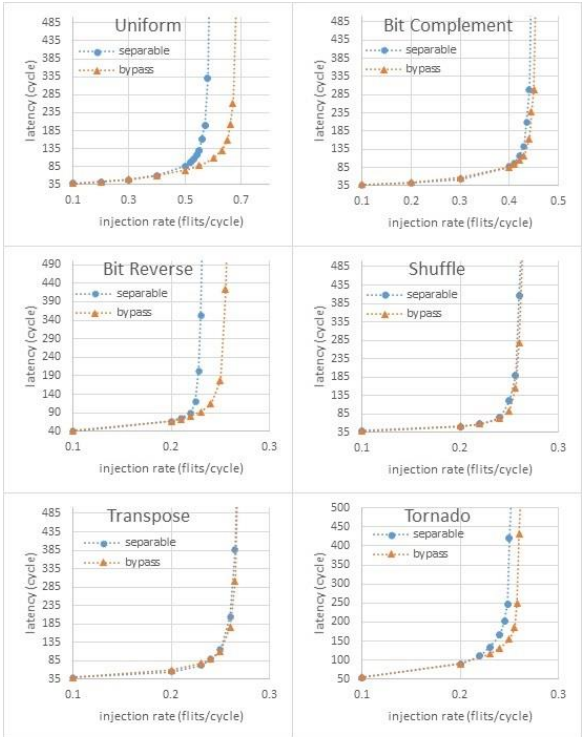


Figure 3: Performance of different traffic patterns: (a) uniform (b) bit complement (c) bit reverse (d) shuffle (e) transpose (f) tornado

though some buffers are unused.

If the network becomes congested quickly, the design with bypass buffer can enhance the throughput greatly because bypass buffer has made the allocation more efficient. Thus, all flits could be delivered forward smoothly, and the throughput of the network is accordingly improved.

At lower injection rates, however, since the number of requests is relatively lower, the contention between these requests is not obvious. As a consequence, the advantage of using bypass buffer is not significant, and the latency results between these two designs are nearly the same.

Another explanation for using bypass buffer could achieve better performance under the circumstances of larger packet is due to the first restriction that bypass buffer must deal with. As was described in Section 3, a head flit would never be transmitted to bypass buffer. If a packet contains more flits, the proportion of head flit to other flits would be lower, so more flits are able to use bypass buffer. Since bypass buffer is utilized more often, a better performance could thus be obtained.

4.4 The Number of Virtual Channels

The impact of virtual channel number is explored in this section, and the simulation results are shown in Figure 5.

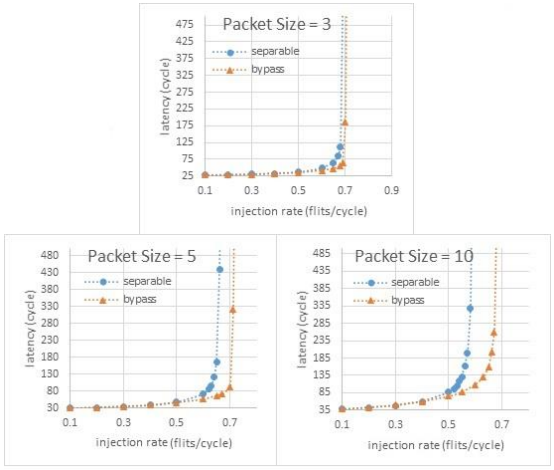


Figure 4: Performance of different packet sizes: (a) 3 flits (b) 5 flits (c) 10 flits

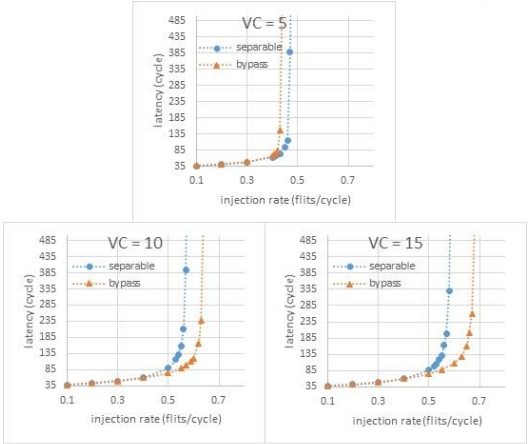


Figure 5: Performance of different virtual channel numbers: (a) 5 virtual channels (b) 10 virtual channels (c) 15 virtual channels

Table 3: The percentage of flit using bypass buffer

The Percentage of Flit Using Bypass Buffer	Injection Rate (Flits / Cycle)	
	0.2	0.6
One-flit Queue	1.3%	5.8%
Three-flit Queue	3.4%	9.8%
Five-flit Queue	4.3%	12%

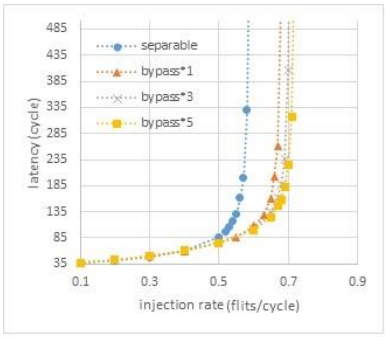


Figure 6: Performance of different bypass buffer queue depths

With only five virtual channels, the bypass buffer design is worse than the baseline design. However, by increasing the virtual channel number, bypass buffer could raise the saturation point of the network substantially.

This result is attributable to the strict mechanism for the bypass buffer design to reallocate a virtual channel. That is, it will take longer time for the bypass buffer design to reuse a virtual channel. Therefore, by providing more virtual channels, this disadvantage could be reduced effectively.

4.5 The Depth of Bypass Buffer Queue

In this section, the depth of bypass buffer queue is the next parameter to be adjusted. Each bypass buffer queue still has only one read port and one write port regardless of the increased depth. Namely, in each flit cycle, each bypass buffer queue could only receive one flit from input port and send out one flit to output port.

By increasing the depth, each bypass buffer queue could accommodate more flits, and consequently there are more chances to utilize bypass buffer. Theoretically, a larger buffer queue would enlarge the benefit of bypass buffer, and accordingly leads to better performance.

As depicted in Figure 6, though the throughput of the network is enhanced indeed by using larger bypass buffer, the improvement decreases progressively. If the buffer size is large enough for flits to use in most cases, a larger size than this could not bring great benefit any more. Compared with the base design, using one-flit bypass buffer queue could increase the saturation point by 16%, and the three-flit queue design could further enhance 4% more than the one-flit queue design. However, the five-flit queue design only increases 2% saturation point than the three-flit queue design.

Table 3 shows the ratios under different circumstances that a flit would use bypass buffer rather than crossbar to pass a router, and they are measured by the choices of all collected flits during their transmissions. As the table presents, the five-flit queue design only increases the utilization of bypass buffer little. In spite of the bigger improvement induced by the three-flit queue design, considering that it takes three times the bypass buffer queue size to enhance less than two times the ratio at high injection rate, using only one flit depth for each bypass buffer queue may be an economical option.

The table also accounts for the almost identical latency of the bypass buffer design and the baseline design at lower injection rates. Bypass buffer could not bring much benefit at lower injection rates because only few flits would use it.

Table 4: The modeling parameters for cost evaluation

Technology	32nm
Frequency	1G Hz
V_{dd}	0.9V
Temperature	340K
Number of Router Ports	5
Number of Virtual Channels per Port	5, 15
Buffer size of Virtual Channel	5 Flits
Number of Bits per Flit	64

Table 5: The area evaluation of bypass buffer

Area (Micrometer^2)	Virtual Channel	
	5	15
5 One-flit Buffer Queues	9.39E02	
5 Five-input Arbiters	4.40E02	
5 Five-output DeMux	1.75E03	
5 Two-input Mux & DeMux	8.75E02	
Bypass Buffer Total	4.00E03	
Original Router	4.43E04	1.18E05
Increased Percentage	9.04%	3.4%

5 Cost Evaluation

5.1 Simulation Setup

DSENT, a C++ based tool for hierarchical modeling of NoC, has been proposed to evaluate the area and power of different network configurations [11]. Once the parameters of a network are provided, it will build a corresponding model to evaluate the related area and power cost.

As shown in Figure 1, extra components including five one-flit buffer queues, five five-input arbiters, five five-output demultiplexers, five two-input multiplexers and five two-output demultiplexers. The increased percentage of cost which is brought about by these components will be evaluated later.

Table 4 lists the parameters which are used in the subsequent experiments, and the simulated buffer is DFF-based RAM. Because using different amounts of virtual channels will make a significant impact on the cost of router, two kinds of virtual channel number will be simulated to compare the increased cost percentage of the additional components under different router architectures. To simulate the conditions under uniform traffic pattern, all the router input ports have the same amount of injected flits.

5.2 Area

Table 5 presents the increased area percentage caused by those components. The result reflects that the percentage is largely influenced by the amount of virtual channels. A router with more virtual channels could considerably lower this increased percentage caused by bypass buffer.

5.3 Leakage Power

The increased percentage of leakage power which is produced by those additional components are listed in Table 6.

Table 6: The leakage power evaluation of bypass buffer

Leakage Power (W)	Virtual Channel	
	5	15
5 One-flit Buffer Queues	4.54E-04	
5 Five-input Arbiters	2.12E-04	
5 Five-output DeMux	7.01 E-04	
5 Two-input Mux & DeMux	3.51E-04	
Bypass Buffer Total	1.72E-03	
Original Router	1.88E-02	5.25E-02
Increased Percentage	9.15%	3.27%

Since each hardware unit always accompanies by the leakage power, the percentage results here are very similar to those of the area evaluation. The amount of virtual channels has a striking effect on the increased leakage power percentage.

5.4 Dynamic Power

In this section, the cost to be evaluated is the dynamic power, and the results are shown in Table 7 and Table 8. Since dynamic power is a data-dependent energy consumption, the cost percentage at two different injection rates are presented.

Except for the two-input multiplexers and two-output demultiplexers, for other additional components, dynamic power consumption is only dissipated when a flit is sent to bypass buffer.

As have been seen in Table 3, the ratio for a flit to use bypass buffer rather than crossbar to leave a router is quite low even at high injection rate, so the dynamic power brought about by those additional components is not significant.

The reduced crossbar power item in the table represents the saved power by reducing the workload of crossbar. Because using bypass buffer would preclude the use of crossbar, this item is to be subtracted from the produced dynamic power by bypass buffer.

Adding up all the items in the table, the increased percentage of dynamic power is much lower than that of area and leakage power. Comparatively speaking, the dynamic power caused by bypass buffer could nearly be neglected.

6 Conclusions

To address the inefficiency problem of separable allocator, this research has proposed the bypass buffer method as a possible solution. Through bypass buffer, the idle ports caused by matching inefficiency could be used to deliver flits, and therefore the time for a flit to wait in an input port buffer is reduced. As a result, with bypass buffer, the packet latency would be lower than the base design.

The more the traffic behavior is balanced in a network, the better performance that the bypass buffer design could obtain. Therefore, uniform traffic is the most suitable traffic pattern for the bypass buffer design.

To prevent some possible faults from happening during packet transmission, there are some restrictions for bypass buffer to follow. However, by increasing packet size and using more virtual channels, the disadvantages induced by these restrictions could be largely reduced, and better performance could be achieved accordingly.

Table 7: The dynamic power evaluation of bypass buffer at injection rate = 0.2 flits/cycle

Dynamic Power (W)	Virtual Channel	
	5	15
5 One-flit Buffer Queues	1.65E-06	1.66E-06
5 Five-input Arbiters	1.95E-07	1.97E-07
5 Five-output DeMux	1.57E-06	1.58E-06
5 Two-input Mux & DeMux	3.79E-05	
Reduced Crossbar Power	1.57E-06	1.58E-06
Bypass Buffer Total	3.97E-05	3.97E-05
Original Router	1.14E-03	2.28E-03
Increased Percentage	3.48%	1.74%

Table 8: The dynamic power evaluation of bypass buffer at injection rate = 0.6 flits/cycle

Dynamic Power (W)	Virtual Channel	
	5	15
5 One-flit Buffer Queues	1.58E-05	2.15E-05
5 Five-input Arbiters	1.86E-06	2.54E-06
5 Five-output DeMux	1.50E-05	2.05E-05
5 Two-input Mux & DeMux	1.14E-04	
Reduced Crossbar Power	1.50E-05	2.05E-05
Bypass Buffer Total	1.31E-04	1.38E-04
Original Router	2.81E-03	6.16E-03
Increased Percentage	4.66%	2.24%

Since the increased percentage of cost considerately depends on the router architecture, if more virtual channels are used, the cost produced by those extra components would be less dominant. Because using more virtual channels could also enhance the performance of bypass buffer, the amount of virtual channels is the most important parameter to be considered while using bypass buffer.

If the objective of using bypass buffer is to achieve better performance with reasonable cost, a one-flit depth of bypass buffer queue may be an economical choice. However, if the objective is to enhance the throughput as much as possible, a deeper bypass buffer queue might be an option, although the enhancement will decrease gradually.

Since bypass buffer could not bring much benefit at low injection rate, using bypass buffer in a throughput-oriented network is more appropriate than in a latency-oriented network. Moreover, using bypass buffer in those networks that need a lot of virtual channels to be distributed to several classes of packets is also suitable.

In this research, to reflect the improvement under general circumstances, the network setting is quite simple. The routing algorithm used in all the previous simulations is dimension-order routing, which is one kind of deterministic routing that will never lead to a deadlock situation. To apply adaptive routing, however, the deadlock problem must be solved first; thus, each router in the network is prone to use more virtual channels [12]. For this reason, using bypass buffer in a network with complicated adaptive routing might be possible.

As presented in Section V, the main power consumption for bypass buffer is from leakage power. Since bypass buffer could not improve much performance at lower injection rates, a feasible method to reduce power consumption is to turn off bypass buffer under these circumstances.

To simplify the design of bypass buffer, there are only one read port and one write port in each bypass buffer queue mentioned earlier. However, when the depth of each queue is enlarged, increasing the write port number could be a method to further improve bypass buffer. Although a more complicated control logic is needed, the utilization of bypass buffer would be enhanced greatly.

7 References

[1] L. Benini and G. de Micheli. Networks on Chip: A New Paradigm for Systems on Chip Design. In Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02), 2002.

[2] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Inteconnection Networks. In Proceedings of the 38th Conference on Design Automation (DAC-38), 2001.

[3] W. J. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[4] J. Duato, S. Yalamanchili, and Lionel Ni, Interconnection Networks: an Engineering Approach, Morgan Kaufmann Publishers Inc., 2002.

[5] Y. Tamir and H.-C. Chi. Symmetric Crossbar Arbiters for VLSI Communication Switches. IEEE Transactions on Parallel and Distributed Systems, 4(1), 1993.

[6] W. J. Dally. Virtual-Channel Flow Control. IEEE Transactions on Parallel and Distributed Systems, 3(2), 1992.

[7] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in Proc. Conf. High Perf. Comput Network. Storage Anal. , Nov. 2009, pp. 1–12.

[8] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High speed switch scheduling for local area networks," ACM Trans. Comput. Syst., vol. 11, no. 4, pp. 319–352, Nov. 1993.

[9] J. G. Delgado-Frias and G. B. Ratanpal. A VLSI Wrapped Wave Front Arbiter for Crossbar Switches. In Proceedings of the 11th Great Lakes Symposium on VLSI, 2001.

[10]. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, J. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in ISPASS'13.

[11]C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Net-works-on-Chip Modeling," in Proc. NOCS, 2012.

[12]Daniel H. Linder and Jim C. Harden. "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes." IEEE Transactions on Computers, 40(1):2‑12, Jan. 1991.

ACKNOWLEDGMENT

The authors thank the support from MOST under grants 104-2220-E-007-006 and 103-2220-E-007-021, and MOEA under grant 103-EC-17-A-02-S1-202.

Dynamic Interrupt Controller and Conflict Management for Transactional Memory in Embedded System

Jun Young Moon[1], **Jun Gil Ahn**[2] **and Jong Tae Kim**[1,2]
[1]Department of IT Convergence, Sungkyunkwan University, Suwon, South Korea
[2]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, South Korea

Abstract – *In hardware transactional memory system, selecting an interrupt handling mechanism is the one of problems. To handle interrupts occur in transactions, all systems need special mechanisms but these require more hardware or software resources, so this is not acceptable to the embedded system that has limitations. In this paper, we proposed interrupt handling process and interrupt controller that distributes interrupts to proper core in the system. Then we present performance metrics for the system and how transactional memory policies affect to the metrics. Simulation results show that roughly 80% of interrupts that occur in transaction can be improved and eager conflict detection, polite and polka contention managements are proper polices for the proposed system.*

Keywords: Transactional memory, parallel computing, interrupt handling, contention management, conflict detection

1 Introduction

In multi-core system, the complexity of lock-based synchronization is well-known problem to almost programmers. The transactional memory (TM) is proposed to solve the problem [1]. TM system traces all memory operations in critical section and records these in special buffers. If the critical section is committed, all memory operations is written to memory. In contrast, if the critical section is aborted, information about the critical section in the buffer is discarded and TM system retries the critical section.

There are some problems to implement a real hardware transactional memory (HTM) system for embedded systems. One of the problems is how to handle interrupts in transaction time that is interval between transaction start and end. In an embedded system, decreasing response time of user inputs is critical to the performance of whole system, because almost applications implemented by an embedded system want to response to users as soon as possible. As a result how to handle interrupts in HTM for embedded systems is more important than for high performance computer systems.

The traditional method to handle interrupts that occur in transaction time is that the interrupts have to be pended or the transaction has to be aborted [2]. This is simple and easy to implement by hardware. But this method has disadvantages because the average response time is decreased so this is not applied to embedded systems directly.

To solve the problem, some types of TM use special mechanisms [3, 4, 5]. Nested institutes a nested transaction concept, so the interrupt handler can be executed in transaction time [3]. To implement nested transaction, the complexity of cache structures and the size of buffers is more increased. VTM can suspend transactions so interrupt can be handled in transaction time [4]. To support that, it uses virtualization concept and need additional hardware controller, memory spaces and software layers to save and restore overflowed or suspended transaction data. MetaTM uses stacked transaction concept to handle interrupts [5]. By using transaction push-pop primitives, interrupt handler can suspend and restore transaction information. Those solutions always need more hardware buffer to store transaction state and complex TM controller. As a result, those also have disadvantages to embedded systems. Because embedded systems that always want to less cost, power and size and the systems have more strict criteria to hardware and software resource than high performance computer. Eventually, HTM for embedded system need interrupt handling mechanism that is easy to implement and has short response time.

In this paper, we focused on the interrupt handling process in TM for embedded systems. First, we present simple interrupt controller for TM to be used on embedded system. Second, we proposed some performance metrics for interrupt handling in the proposed controller and evaluate performance of TM systems by using the metrics. Then we analyze how many interrupt handlers can be improved and how choosing policies affect performance of interrupt handling and propose the best policy for the suggested interrupt handling process.

In section 2, we present the interrupt handling process in the proposed system and suggest the performance metrics. In section 3, we analyze the effects of contention management policies and conflict detection methods to proposed metrics. In section 4, we explain the evaluation environment to check tendencies explained in section 3. In section 5, we describe the result of experiments. In section 6, we discuss the conclusion of this paper.

2 Interrupt handling

In an embedded system, overheads needed to implement HTM system should be small. That means architecture of the HTM should have only essential components that has to be remained in the architecture to implement transaction concept. For example, transaction buffers and modified buses should be always in the architecture. In contrast, supporting to handle interrupts that happen in transaction time is not an essential part to the system. Because, by using pending interrupt concept, all interrupts in transaction time always can be handled correctly. Only drawback is increasing response time. To reduce that, we modify an interrupt controller that almost embedded systems already have.

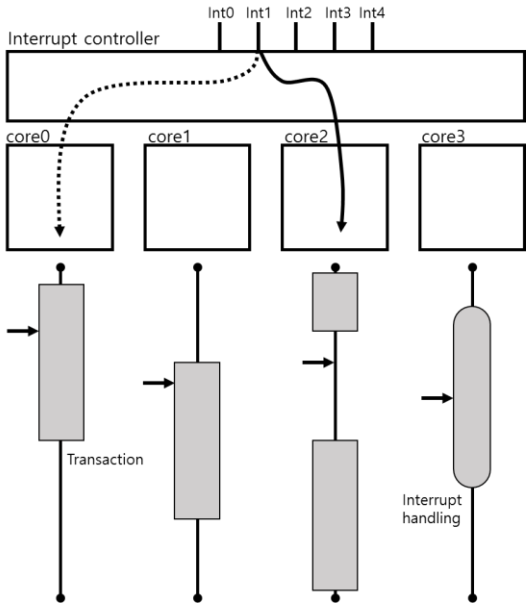


Figure 1. Dynamic interrupt distribution for TM

2.1 Interrupt controller

Because the proposed architecture cannot support handling interrupts happened in transaction time, interrupts should be sent to empty core that doesn't handle interrupt or execute transaction at that time as much as possible to decrease response time. To find empty cores, the interrupt controller read TM status register that each core already have. The TM status registers have the information about whether each core execute a transaction. Interrupt status of each core is stored in interrupt controller registers. Almost interrupt controller in multi-core system can distribute interrupts to each cores and select what core is a target processor for a specific interrupt. As a result it may be not expensive that overhead to implement the proposed interrupt controller.

Figure 1 shows the concept of the suggested interrupt controller. The proposed interrupt controller can know whether cores are busy or empty. In figure 1, the state of core0, core1 and core3 are busy because core0 and core1 execute a transaction and core 3 handles an interrupt. But the core2 is empty core because it does not execute any transaction or handle an interrupt at that time. If the interrupt 1 is happened at that time, the interrupt controller try to send the interrupt to the core 0 that is decided by target register statically. But, the interrupt controller know the core0 is busy so that it modifies the target core for the interrupt 1 from core0 to core2 dynamically to decrease the response time for the interrupt 1.

2.2 Performance metrics

To evaluate the interrupt handling performance of the suggested TM system, there are three types of performance metrics.

- The ratio of occurring interrupt in transaction time. If the number of interrupt is few, the average response time of all interrupts is decreased.

- The interrupt response time of occurring interrupt in transaction time. Because the proposed system using the interrupt pending mechanism, interrupts that occur in transaction time have to wait the transaction end if there are not empty core.

- The number of empty cores at occurring an interrupt in transaction time. If existence probability is high, the average response time of interrupts is decreased.

The last is important to the suggested controller since it impacts to the first and second metric. If the existence probability of empty cores is increased, the ratio of interrupts that occurs in transaction time and how much the second metric affects to the total performance of interrupt handling are decreased. By using the metrics, we find the effective TM policies for the proposed interrupt handling system.

3 TM policies

In TM system design, choosing contention management and conflict detection are the most important steps, because the performance of almost TM systems is sensitive to the policies [6]. Since finding the most effective policy for the proposed architecture is important.

3.1 Existing TM policies

There are popular contention management policies are proposed in [6]. The polite policy uses the randomized back off increased exponentially. The karma policy has priority concept to determine what transaction will be aborted and uses fixed back off. The timestamp policy also employ fixed back off, but priority of transaction is start time of each transaction. The karma and polite combine to form the polka policy. The priority in the karma and back off in the polite are used in the polka policy. The conflict detection mechanisms

are eager and lazy [7]. The lazy conflict detection checks whether there is a conflict between transactions at only commit time. But the eager conflict detection detects a conflict at every memory operations in transaction.

3.2 Analyzing the effect of TM policies

3.2.1 The ratio of occurring interrupt in transaction

The first performance metric of a specific system is influenced by how long a core execute transactions. If the execution time of transaction is long, the probability of occurring interrupt in transaction is increased.

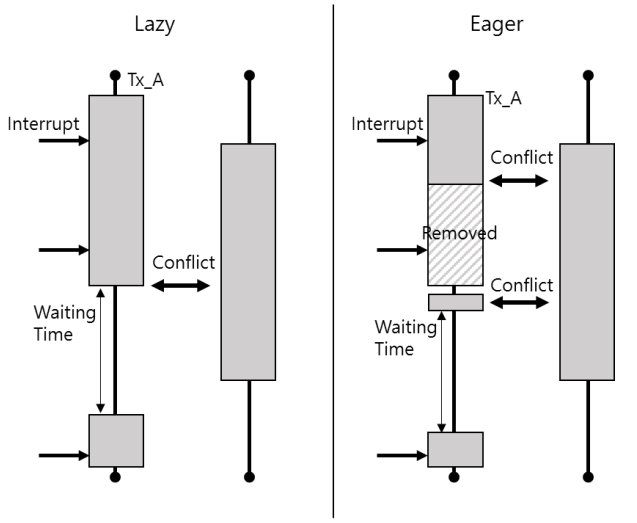


Figure 2 Effect of conflict detection to the first metric

From the perspective, the lazy conflict detection method have more disadvantages than the eager method. The features of the eager is that it checks whether there is a conflict at every memory operations in transaction so that aborting is happened earlier than the lazy method. In figure 2, the length of CPU time that execute transactions in the eager is shorter than the lazy. So the number of interrupt in the lazy case is higher than in the eager case.

The efficiency of the contention management policies also can be analyzed by using the same perspective. The proportion of the time for executing transaction to whole CPU time is influenced by what contention management policy is used in TM, especially what back off policy is employed in each policy. The contention management policy that use fixed back off execute more transactions in certain time than other policies that use exponentially increased back off. Therefore the polite and polka policy that use the back off time increased exponentially have advantages to the first metric.

3.2.2 The interrupt response time of occurring interrupt

Average length of transactions affects to the second performance metric since interrupt happened in a transaction is pended in the proposed system. The average length of transactions of the lazy is longer than that of the eager methods. Contention management policy does not affect interrupt response time. Even if the proposed system selects any contention management policy, the length of transactions is not changed.

3.2.3 The number of empty cores

Factors that decide the value of the third metric are similar to the factors of the first metric. How many CPU times are used to execute transactions is important factor to this metric. Because the smaller the amount of CPU time that execute transactions or interrupt handler is, the higher the probability of existence empty core is. As section 3.2.1 explained, the eager conflict detection method, polite and polka contention management policies that use exponentially increased back off time have advantages.

4 Implementation

To evaluate tendencies of transactional memory policies in the proposed interrupt handling system and how many interrupts can be improved by the proposed interrupt handler, we make an ESL (Electronic system level) platform that support transactional memory concept. The proposed system's target is an embedded system therefore the evaluation platform uses four ARM Cortex-A9 that is provided by OVP (Open virtual platform) [8]. This core model uses the transaction level modeling and doesn't support pipeline and cache system so that the model assumes execution time of all instructions is just one-cycle.

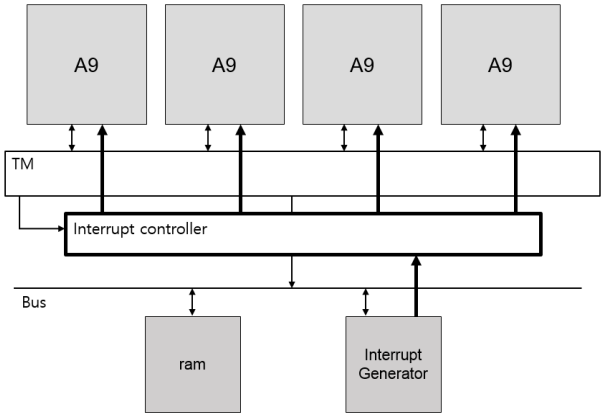


Figure 3 Evaluation platform

Except the cores, other platform components that are transactional memory controller, interrupt controller and ram are implemented by using SystemC and OSCI TLM-2.0 [9]. The transactional memory controller can change internal mechanism among contention management policies and conflict detection methods. Supported contention management policies are the polite, karma, timestamp and polka. The conflict detection is lazy or eager. The controller only

supports the lazy version management. So, the controller can supports eight types of transactional memory policy.

There are two types of workload for evaluating the TM system. One is an interrupt workload. In this paper, we implement an interrupt generator module that makes randomized interrupts that are sent to each cores in figure 3. Another is transaction program that is executed on each cores. In this paper, we use the counting benchmark [2] and the number of transactions that is needed to complete the benchmark is set to $4x10^5$. Additionally, to check effects of size of transaction, we simulate the benchmark as we change the size to 16, 32, 52 or 92 bytes.

5 Simulation Results

Figure 4 shows the proportion of pended interrupt that occurs at transaction time to normal interrupt that is handled instantly. The size of transaction used to simulations is set to 32 bytes. The average performance of eager is 39.7% and that of lazy is 19.8%. As we explained in section 3.2.1, the performance of eager method is better than that of lazy in every contention management policies. Between the contention management policies, the polite is better than the others because that employs exponential back off. The polka policy also use same back off mechanism of the polite but the performance of the polka is lower than that of the polite. The reason of the phenomenon is that the polka policy uses the priority concept in karma policy, so the probability of increasing waiting time is low.

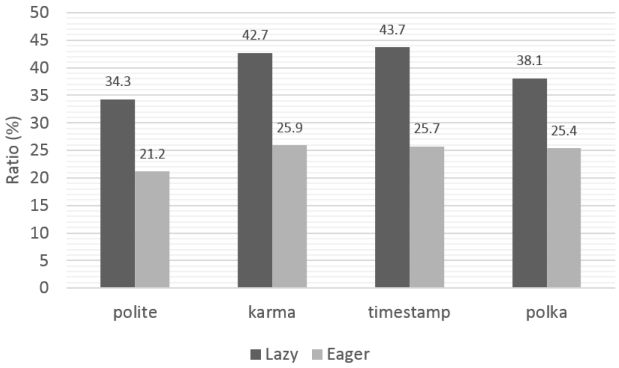


Figure 4 Proportion of interrupt in transaction time

Table 1 shows length of response time that is needed to handle an interrupt that occurs at transaction time. The unit of length is the number of instructions. In almost contention management cases and all transaction size, the eager methods is better than the lazy. There are not difference between contention management policies except the polite-eager case. The reason of polite-eager case is this uses exponential back off. If there are many conflict between transactions in the polite, the average of back off time is increased continuously. The longer the average of back off becomes, the lower probability of conflict is. As a result, the eager is being similar to the lazy. The polka has also same tendency but that is not critical as the polite because the former uses priority concept.

		16	32	52	92
lazy	polite	21.9	50.0	85.0	150.0
	karma	21.1	47.9	83.5	152.0
	timestamp	21.0	47.5	84.5	150.0
	polka	21.4	44.3	77.5	143.8
eager	polite	20.6	44.0	77.0	150.0
	karma	20.7	35.0	57.6	95.0
	timestamp	20.5	36.0	56.0	95.0
	polka	20.2	37.4	62.4	108.3

Table 1 Average response time of interrupt in transaction time

Figure 5 shows how many interrupts can be sent to empty core when the interrupt occurs at transaction time. The y-axis of figure 5 is the percentage of existence of empty core. The size of transaction used to simulations is set to 32 bytes. In lazy system, 72% of pended interrupts can be sent to other empty core. In eager, 97% of interrupt handler for interrupt occurs at transaction time can be improved. In addition, the polite and polka is better than the karma and timestamp.

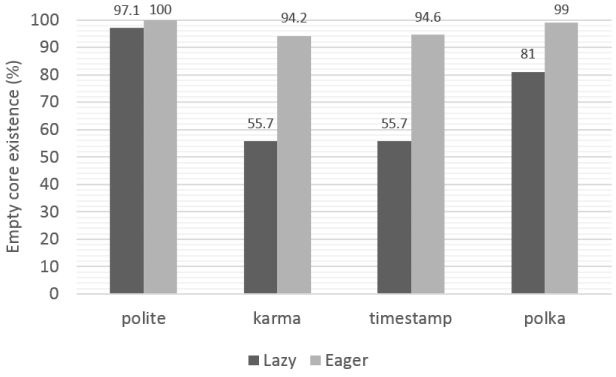


Figure 5 Existence probability of empty cores

If the size of transaction is increased, CPU time for executing transaction is also increased. As a result, the probability of existence is also decreased. Figure 6 shows the result of simulations that use various size of transaction and the lazy conflict detection. Karma, timestamp and polka has same tendency except the polite policy. The polite policy has a unique tendency that the bigger the size of transaction become, the higher the probability of existence is. The phenomenon is explained by using the same reason used to why the first metric of polite is higher.

According to the simulation results, the proposed interrupt controller can improve the performance of interrupt handling. But there are some performance gap in what

contention management and conflict detection method are selected to TM system. Between conflict detection methods, the eager method for conflict detection has many advantages about the proposed system. Every contention management has better performance when that is combined with eager.

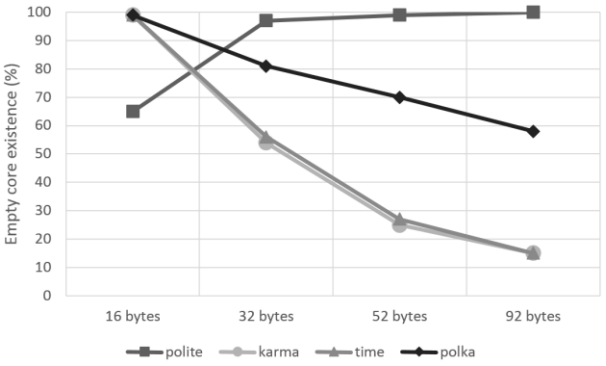


Figure 6 Probability of existence tendency in various transaction size

Among contention management policies, the polite and polka have advantages because those employ exponential back off mechanism. In addition, required hardware resource to implement the polite is smaller than that of the polka because the polite does not use any priority concept so that the polite is proper contention management to the proposed system. The effect of the size of transactions also shows the polite is more appropriate than the polka. But, the proposed result is only evaluated about the interrupt handling performance. So, other types of performance metric should be considered such as elapsed time, abort rate and so on.

6 Conclusion

In this paper, we proposed interrupt handling mechanism in TM for embedded system. The proposed mechanism uses interrupt controller that has transaction information about each cores and distributes interrupts to empty core by using the information. Then we present the performance metrics to evaluate proper contention management and conflict detection method for the proposed system. Also we explain the features of each policies that affect to the proposed performance metrics. Then we implement the ESL platform that has Cortex-A9 quad-core, transactional memory controller and interrupt controller and evaluate the effect of each policies.

The simulation results present that 72% of pended interrupts can be improved at lazy on average and 99% of pended interrupts also can be improved at eager at the size of transaction is 32 bytes. But improvement amounts are affected by the size of transaction and what policies is used by the transactional memory controller. By using the simulation results, we conclude that the eager has advantages to handling interrupts. In addition, the exponentially increased back off time has advantages so that the polite and polka contention management policies are proper to the proposed transactional memory system to handle interrupts effectively. Between them, the polite is better because the implementation cost is lower and the performance tendency about the size of transactions is better.

7 References

[1] Harris, T. Cristal, A., Unsal, O. S., Ayguade, E. Gagliardi, F., Smith, B. and Valero, M. "Transactional memory : An overview"; IEEE Micro, Vol. 27., Issue 3., 8-29, 2007.

[2] Herlihy, M. and J. E. B. Moss. "Transactional memory : Architectural support for lock-free data structures" ; ISCA 93 Proceedings of the 20th annual international symposium on Computer Architecture, 289-300, May 1993.

[3] Moss, J. Eliot B., and Antony L. Hosking. "Nested transactional memory : model and architecture sketches" ; Science of Computer Programming, Vol. 63., Issue 2., 186-201, Dec 2006.

[4] Rajwar, Ravi, Maurice Herlihy, and Konrad Lai. "Virtualizing transactional memory"; ISCA 05 Proceedings of the 32nd International Symposium on Computer Architecture, 494-505, Jun 2005.

[5] Ramadan, H. E., Rossbach, C. J., Porter, D. E., Hofmann, O. S., Bhandari, A., and Witchel, E. "MetaTM/TxLinux : tranasctional memory for operating system"; ACM SIGARCH Computer Architecture News, Vol. 35., Issue 2., 92-103, May 2007.

[6] Scherer III, W. N. and Scott, M. L. "Advanced contention management for dynamic software transactional memory"; Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing., 240-248, 2005.

[7] Rachid Guerraoui and Paolo Romano. "Transactional Memory. Foundations, Algorithms, Tools and Applications". Springer International Publishing, 2015.

[8] Imperas Inc. Open Virtual Platform World. http://www.ovpworld.org/.

[9] Initiative, Open SystemC. "OSCI TLM-2.0 language reference manual." Version JA32, http://www.systemc.org, 2009.

Hardware reconfigurability: from concept to implementation

Schagaev Igor[1*], Castano Victor[2], Kaegi Thomas[2]

[1] London Metropolitan University 166-220, Holloway Road, N7 8DB, London, UK

[2] ITACS Ltd, 157 Shephall View, SG1 1RR, Stevenage, England

** Correspondence should be addressed: i.schagaev@londonmet.ac.uk, info@it-acs.co.uk*

Abstract. *Hardware reconfigurability is essential property of the next generation of computer architectures. We introduce and explain it. We show that reconfigurability is not a property, but a process that should be supported at all stages of design and further functioning. A novel instrument for system reconfigurability, the Syndrome, is described with application and the control of all three zones of computer architectures: active, interfacing and passive. At a lower hardware level, a special reconfiguration element - so-called T-logic is introduced and its operations explained. Using the Syndrome, we illustrate a process of control of reliability and support of graceful degradation. Applications of syndrome for reconfigurability control gaining Performance, Reliability and Energy-wise operations are discussed.*

Keywords: *reconfigurability; performance-, reliability-, energy-smart functioning; hardware, computer system, hardware syndrome*

1. Existing myths and what is really required

The 21st century has started by and with repeating of the previous century mantra on technology limitations and the need of parallelization of computing. Parallelization is discussed at the level of tasks, software, hardware as well as schemes of implementation and support. Thus, accordingly a simple Google search (at 20.02.15) "parallel programming" provides 6,510,000 entries, "parallel programming model" reaches 2,340,000 entries and "parallel computing" is healthy at 13,400,000. At the same time, typical desktops with 4 or 8 processors, now renamed as "cores" consume from 350 to 850 Watts and surprisingly do not provide "at the order of magnitude" higher performance.

Similarly - transfer of similar amount of "cores" to mobile devices, again, did not gain a lot: we recharge our new mobile phones every day now. Add here: our new devices much more often fail to operate: hangs for no reason, self-restarting and unexplained loosing of battery charge - all this indicates that new designs do not really address challenges of new applications performance, reliability or power efficiency.

Accounting, banking, health monitoring and similar applications are becoming critical and wider spread; the router clusters and network centers consume a visible amount of power generated by power stations. One might ask: "what is missing?" and "why are our designs not greatly efficient?"

Clearly, the myth of parallelization and success of its implementation did not serve a "silver bullet" role. Surprisingly, the term "*reconfigurability"* is much less popular in Google search - 243,000 entries.

What is missing? What properties of computer systems do we expect to have in the nearest future? One of those properties is flexibility,

the ability to evolve [2]. In turn, system flexibility assumes the presence of hardware flexibility and system software flexibility.

System flexibility can be considered as several connected properties and supporting features, as described in [1],[2],[3],[4]. We list: flexibility (of hardware and system software), resilience, scalability (task-wise, frequency-wise, technology-wise), performance-, reliability-, energy smart- functioning. We aggregate these properties by the name of PRE-smart systems (performance-, reliability-, energy-). We argue that all mentioned properties are required and should be implemented within a computer system. Good point that it is possible as all of them are based at some level on *reconfigurability*.

Therefore, efficient design and implementation of *reconfigurability* becomes a task of utmost importance for the next generation of computer systems.

2. Reconfigurability: a concept

As it is defined in [1], [2], [3] evolving system (EvSy) in terms of evolving properties might be considered and designed as a pair of hardware and system software; formally:

EvSy := <HW, SSW>

Both main elements, hardware and system software, must possess their own reconfigurability features and support each other: a) Hardware support of system software reconfigurability and b) System software support of hardware reconfigurability.

System reconfigurability must be introduced at the design level and pursued along at other levels, especially for critical applications. System reconfigurability should also be considered, throughout the entire life cycle.

Therefore, a system reconfigurability becomes not only a *feature* or *property* of a system, but *a process*. This process serve the need of introduction and maintenance of reconfigurability, including ways of changing configurations and ability to reconfigure.

For example, during critical missions, reconfigurability should be executed within real-time constraints, *invisible* for applications.

In turn, during regular operation, system reconfigurability should be used to adapt the system to different requirements in terms of efficient performance, reliability and power consumption. We call this PRE-smart functioning as defined in [2].

Reconfigurability for reliability should be implemented with supports the ability of system to recover with minimum time overheads. Here it is worth to mention that reconfiguration might have internal and external reasons. For instance, the system might exclude or isolate some hardware elements from the configuration due to a transient/permanent hardware fault detected via checking schemes (external reason). This isolation should be considered as a process and be “fine-tuned" by minimizing the hardware loss. Additionally, with energy saving in mind, the system could setup a simple hardware configuration for a particular task execution (internal reason).

In energy saving functioning, reconfigurability has to provide a mechanism to disconnect or switch to a lower consumption mode all hardware elements that are not required for the active program processes.

3. Reconfigurability implementation

Hardware of computer systems in terms of information processing consists of three semantically different zones: active, passive and interfacing, as Figure 1 illustrates.

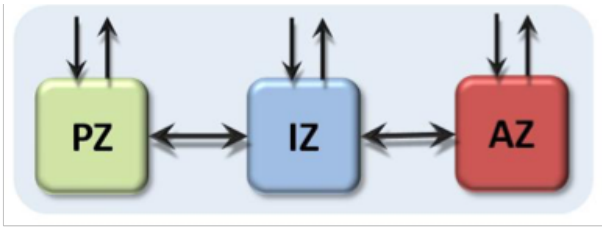


Fig. 1 Computer zones of information processing

At first, the information transformation area – further called active zone (AZ); secondly the information storage area – called passive zone

(PZ). The interconnection of these zones is the interfacing zone (IZ).

Active Zone: The active zone consists of the microprocessor elements including the arithmetic unit (AU) and logic unit (LU). Both units are separated for better fault isolation and easier implementation of hardware tests and, primarily, reconfigurability.

Interfacing Zone: This includes all communication components such as the memory buses and the reconfiguration logic. A configurable bus allows the reconfiguration of the hardware to exclude any failed hardware components and switch into a degraded state, or to replace the failed component with a working one.

Passive Zone: This includes basic storage systems, such as memory, that do not act by themselves but are handled by controllers or devices, saving data.

All three zones have different properties and might use different redundancy mechanisms to tolerate internal faults and to reconfigure efficiently for performance, reliability or energy saving purposes. The proposed computer structure of each zone is shown in

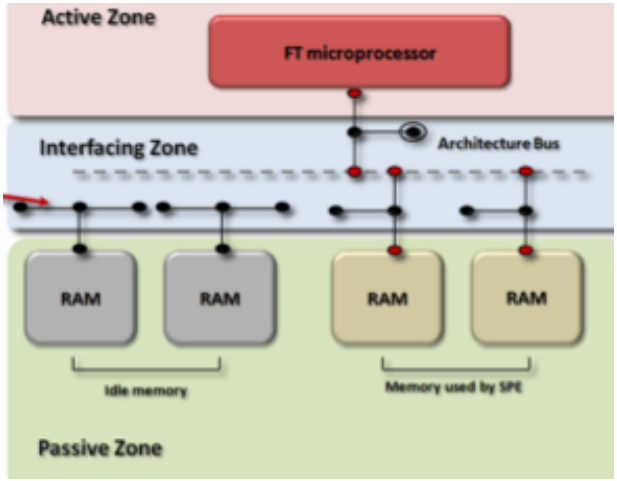


Fig. 2 Reconfigurability of computer system

Note that efficient reconfigurability can be achieved when it is implemented with minimum deliberate redundancy been introduced in the system [4]. In our case hardware redundancy exists in the form of buffer, register files, replicated memory modules, majority voting schemes and interfacing logic.

With regards to SSW, some extra elements required to support reconfigurability and fault tolerance are: checkpoint monitor, recovery point monitor, process synchronization and reconfiguration monitor. These are named monitors to express their uninterruptible mode of operation [1].

All three zones, (see Fig. 1 and Fig. 2) must be reconfigurable for their own purposes as well as other zones requests. Each zone might have different reconfiguration properties. Interactions between zones define the level of reconfigurability and flexibility of the architecture.

4. System Syndrome

4.1 Definitions

As mentioned earlier, the new system property must be supported by hardware and system software implementation of all required processes that make this property. For this purpose we introduce a special hardware scheme called a S*yndrome*.

The term Syndrome is new Latin (origin 1535-45) and was originated from Greek "Syndrome" where: "Syn-" from combination, concurrence. For our purposes a Syndrome is not just passive, i.e. presenting "a snapshot status" of a system, but also active, a serving tool to control the system configuration. For us a Syndrome is

"a group of related or coincident things, events, actions, signs and symptoms that characterize a particular abnormal condition".

A Syndrome also might help to answer the questions that have been omitted in the vast majority of research on fault tolerance and performance: *"what provides the fault tolerance of the system?"* and *"how big a performance, reliability, energy-saving gain might be achieved?"*

It is usually assumed that the hardware core logic is ultra-reliable and guarantees control of configuration and reconfiguration. Unfortunately, using homogeneous redundancy limits the reliability gain - since techniques based on the same type of redundancy are vulnerable to the same threats. Hybrid techniques based on heterogeneous redundancy can be more effective.

Thus, even when memory or processor checking schemes detect error and transfer information to the Syndrome, this information might not be useful if the system does not include either one or both: "External elements" responsible for exercising reconfiguration and making decisions on configuration/ reconfiguration. Reconfiguration might be initiated externally, by other system elements to create best –fit for task configuration, or, if necessary, by "Internal elements" that are capable to initiate the required sequence of reconfiguration for internal purposes and reasons – faults, errors or power-saving.

Indeed, in regular computing systems, when there are faults in the processing logic, to expect that it is able to perform self-healing and then control and monitor the configuration of the rest of the system looks like a part of fairy tale, not engineering.

There is a solution though, as described below. To be able to absorb any trustworthy information about the status of system elements we have to aggregate all checking and status signals about the condition of registers, memory, AU and LU as well as control unit. This aggregating scheme we call *a Syndrome*.

Clear, reconfigurabilities of passive, interfacing and active zones are different. Therefore, a scheme of implementation of reconfigurability should separate the passive zone and active zone of the proposed architecture.

A clear separation of the functions of processing (of data operation) and storing (memory) enables to apply various checking, recovery and reconfiguration solutions and making system more flexible. The Syndrome acts as a control center for three main functions including *fault monitoring*, *reconfigurability* and recovery.

Fig 3 illustrates Syndrome application.

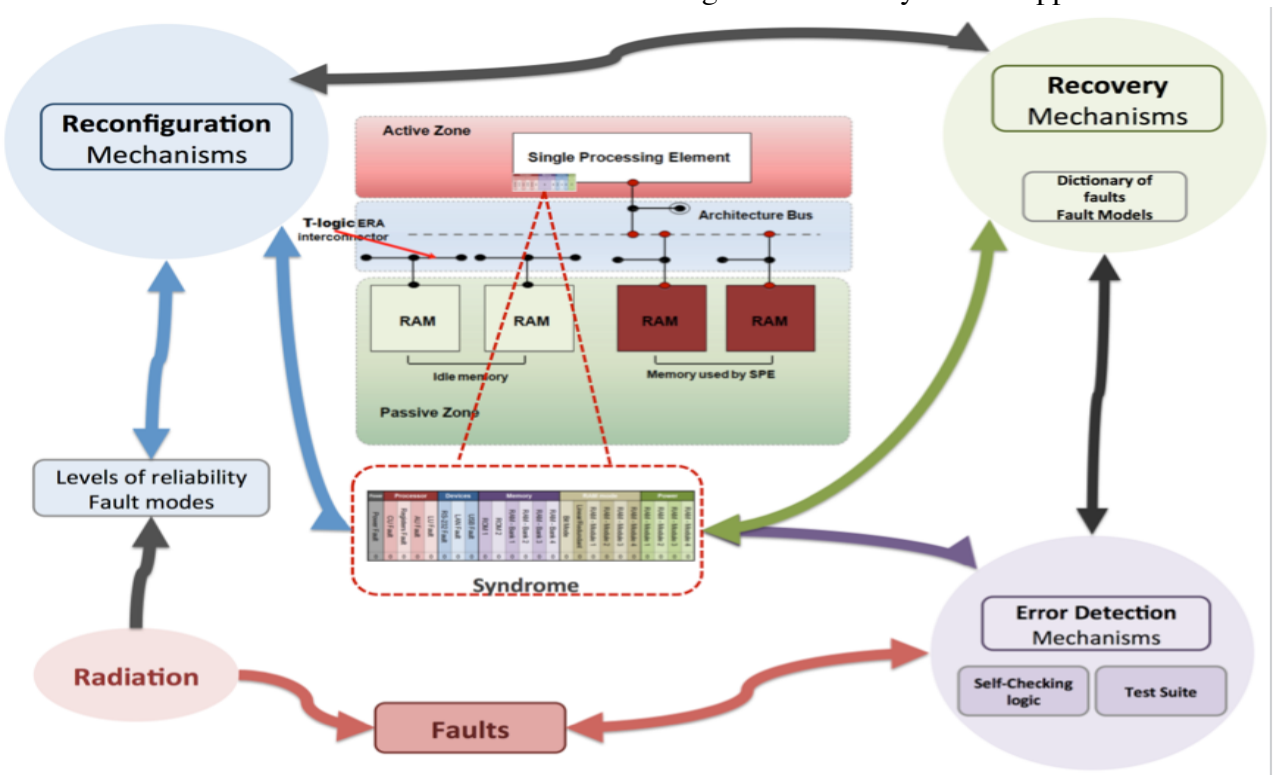


Fig. 3 Syndrome functions for reconfigurability

These three functions serve for the purpose of performance, reliability and power efficiency.

The principle and function of Syndrome from system software point of view are presented using our Evolving System Architecture (ERA) as described in [3], here we address hardware/ hardware configuration concept that might help to implement reconfigurability as essential property of the next generation of computer systems.

4.2 Implementation

From a hardware point of view the Syndrome is represented as a special register that interacts with the system via hardware interruption schemes. Semantically, the structure of the Syndrome is subdivided in three different areas namely, (Fig. 4): Fault control, Configuration control and Power control areas.

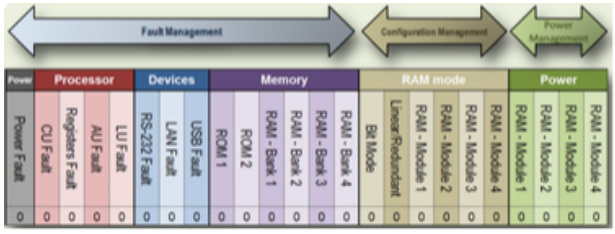


Fig. 4 Syndrome for reconfigurable architecture

4.3 Memory Reconfigurability

The configuration area of the Syndrome reflects the current memory mode that ERA is currently using.
The Bit mode field defines whether the addressing mode is 16- or 32-bits, whereas the L/R field defines whether the memory banks are in linear or redundant mode.
Bit mode "0" means RAM is used in 16-bit addressing mode; mode "1" is a 32-bit addressing mode. RAM modes define how memory can be used: main ("0") or redundant ("1"). RAM Module 1, 2, 3 and 4 represent whether the respective memory module is powered: "0" = Power Off; "1" = Power On.
The power management area reflects the status of the modules in terms of power.

The combination of the three areas of the Syndrome: Fault management, Configuration management and Power management defines and controls the state of the system. For example, a memory module could be in the following states: faulty, failed, stand-by, ideal and off-power. Failed state is assigned to the a malfunctioned element by checking schemes, when real reason behind the malfunction and the ability to operate further is still not clarified. When reconfiguration is initiated by software, the states of hardware elements as well as state of Syndrome might be mirrored in system memory.

Without a doubt, the Syndrome is one of the most critical parts of the system. For reliability purposes, the Syndrome should be made virtually failure-free, for example by implementing three copies of the 32-bit register Syndrome connected to a voter within the processing element.

Another option that solves the complexity would be low-level hardening techniques and/or using different technologies (such as flash memory) just for the Syndrome register.

The Syndrome scheme allows to monitor configuration of memory in a pretty flexible manner, allowing the platform to adapt to different application requirements.

For aerospace applications, for example, a flight control system requires highest reliability, which is possible to achieve, for example, by using duplication for ROM and triplication spare for RAM. On the downside of brutal replication approach we are facing the efficiency problem - the available hardware resources for the program execution become much smaller, i.e. only one fourth of the total amount of available memory is used.

This also implies that only the most critical programs should run on this system, all non-safety critical programs should be moved to another system, making flexibility and efficiency even harder to achieve.

When we are acting on redundant schemes only when we need them – using a syndrome monitoring the chosen configuration allows to tolerate permanent faults by reconfiguring the memory, excluding the faulty unit and if possible including a spare one we are able to exploit a system much more efficient. A

syndrome allows to use standard classic triplicated scheme of memory or processor with much higher efficiency.

Thus the *Syndrome* allows expansion of the working states for memory and other hardware elements of the architecture.

In contrast to classic triplicated configuration with 3 working states reconfigurability of the system supported by proposed Syndrome make overall reliability much higher that known schemes. Our proposed architecture might implement and be operational in 14 different working states, like Markov reliability diagram illustrates at Fig. 5. The dotted lines illustrate toleration of malfunctions. λ and μ stand for ratio of fault and recovery respectively. For example, a modified triplicated scheme [5] combined with systems reconfiguration for malfunction tolerance [6] enables to achieve an order of five reliability boost.

Markov analysis of reliability is useful for the pre-design phase of computer systems.
It turn, real time functioning requires an implementation of reconfigurability *during the mission* making reliability requirements, performance. Therefore using a syndrome makes requirements PRE achievable.

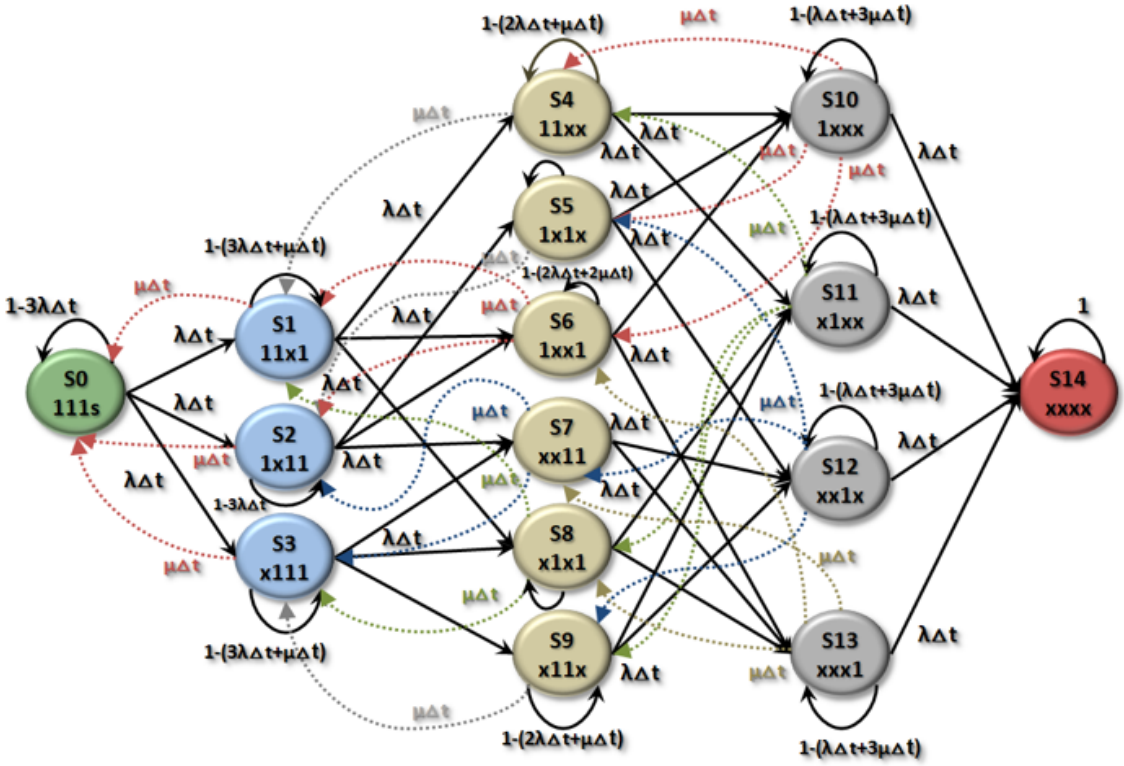


Figure 5 Reliability diagram for reconfigurable RAM using Syndrome

One of the schemes how to make memory of the system really reliable and flexible is presented in Fig. 6. The syndrome register is directly connected to the Memory Management Unit (MMU), which is an extended memory controller with reconfiguration support at runtime.

The MMU manages the connectivity of the memory, configures and reconfigures the working mode to a 16-bit single memory, 32-bit double memory with master/slave configuration or any of the memory addressing schemes available.

The Configuration and Power management flags of the syndrome describe the different states of the memory modules. Different values in the configuration area of the syndrome select the bank used and the mode.
The output memory lines of the processor determine a location within a memory bank,

whereas the Configuration and Power areas of the syndrome specify which banks are to be used and in which mode.

By using this method we can increase the independence of software/hardware configurations for the PRE- purposes. Memory addresses within the code do not need to be arranged, as code integrity is a crucial requirement for safety critical systems.

Special logic schemes -configurators as shown at Fig. 2 are physically included in the MMU.

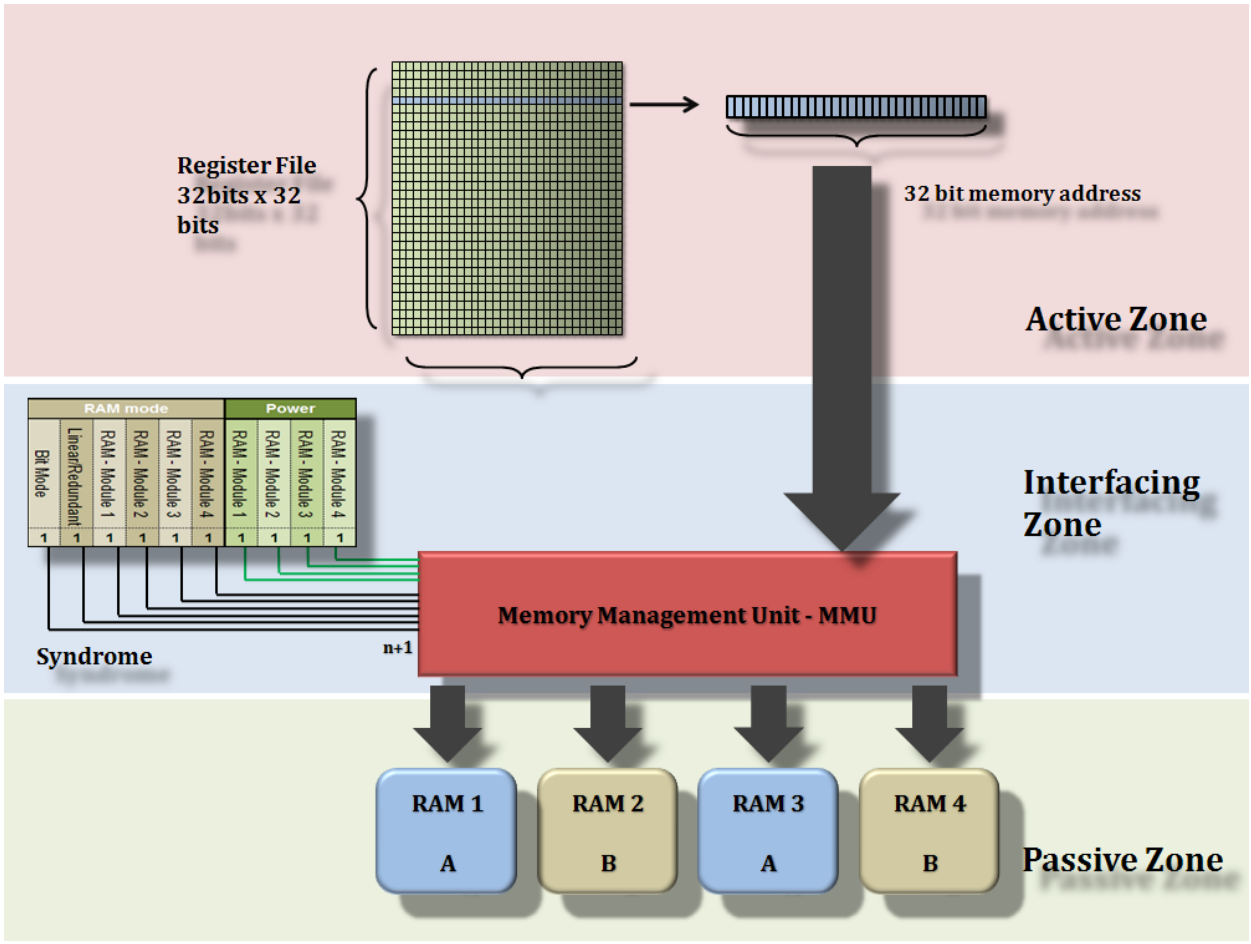


Figure 6 Syndrome use for memory control

5. Reconfigurability - executive element

Using the proposed concept and implementation of the Syndrome, one might implement reconfiguration of the system providing interconnection and dynamic inclusion or exclusion of hardware components from the working configuration.

For this, we suggest to use so-called "a T-logic inter-connector", illustrated by Figures 2, 3 and 6, an idealized concept of a hardware switch that from the system point of view behave like switch in the form of a "T".

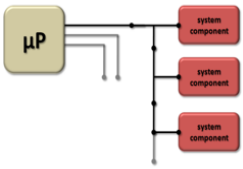
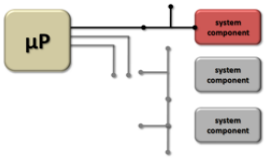
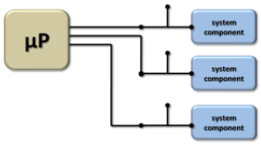
"T" can connect or disconnect hardware elements for the purposes of fault containment, power saving or performance gain. "Rotating" of "T" is virtual and used for illustrative purposes. Detailed description of T-element design is beyond the purpose of this paper for the reason or IPR.

This T-logic serves in the hardware architecture as a scheme that execute configuration scheme, defined by the software or hardware. For example, let us suppose that a fault is detected in a hardware element of a system and thus this element can't be involved in further program execution, either on a temporary or permanent basis.

It should be excluded from working configuration "until further notice".
The four T-logic inter-connectors, one per memory module, are physically contained in the T-logic Management Unit or TLMU.

Using the TLMU enables the memory to be configured and reconfigured according to all supported modes shown in (Table 1) and supports module isolation and power management.

Table 1 System configuration using T-logic scheme

Configuration	Explanation
	"T" logic connect all three components with processor. Top system component acts as leading element. The rest system elements compare the results and participate in voting. This configuration provides maximum reliability.
	This system configuration serves for maximum energy saving. In this case "T" element connects only one system component with processor, while the rest are idle.
	In this case, all three components are used for maximum hardware capacity.When performance of application is main priority this configuration fits the purpose.

6. Conclusion

- System architectures are considered having new properties such as higher efficiency in terms of performance, reliability and energy-smart functioning.
- The structural organization of computer systems is introduced as of information processing including active, passive and interfacing zones.
- System-level reconfigurability can be implemented using a new concept and implementation of hardware element called *Syndrome*; Syndrome aggregates essential information about hardware conditions.
- Functions of the Syndrome for reliability, performance and energy-smart functioning have been described and explained.
- Reconfigurability of a real-time architecture at the system level was proposed and analyzed in the context of each zone. With regards to the interfacing zone, but not limited to, we propose a new hardware element (T-logic), as a basic element or of execution of reconfiguration, making different configurations possible;
- We explained how flexibility, reliability and power-smartness can be achieved using the T-elements;
- Taking into account the memory usage has, by design, a high impact on system reliability and power consumption, reconfigurability of the passive zone has been analyzed and described with explanation of configuration control and the phases of hardware degradation.

7. References

[1] Kaegi T, Schagaev I. System software support of hardware efficiency, IT-ACS Ltd, 2013, ISBN ISBN 978-0-9575049-0-5
[2] Monkman S., Blaeser L., Schagaev I. Evolving systems, Proc. FCS'14 - ISBN #: 1-60132-270-4), Editors: Hamid R. Arabnia, George A. Gravvanis, George Jandieri, Ashu M. G. Solo, Fernando G. Tinetti, pp. 169-179, 2014. http://worldcomp-proceedings.com/proc/p2014/FCS3102.pdf
[3] Castano V., Schagaev I. Resilient computer system design, Springer 2015, ISBN 978-3-319-15068-0
[4] http://faculti.net/video?v=68
[5] Buhanova G., Schagaev I . Comparative Study of Fault Tolerant RAM Structures. Proc. IEEE Dependable Systems and Networks Conference, Guteborg, July 2001.
[6] Schagaev I. Reliability of malfunction tolerance, PP733-737. Proc. Comp science and Information Technology Conf. , IMCIT08

SESSION

INFORMATION SHARING NETWORK SYSTEMS AND SERVICES ACHIEVING HIGH DEPENDABILITY, EFFICIENCY AND USABILITY

Chair(s)

Prof. Hiroaki Nishikawa

A Concept of Community Care System and Community Information Network

Ayami Manaka[1], Akio Ogata[2], Hirohide Matsuzaka[2], Hayato Taniguchi[2], Masaya Nomoto[2], Minoru Fukuzaki[2], Hiroshi Ishii[2], Yasuhiro Nozawa[3], and Keisuke Utsu[2]

[1]Graduate Sch. of Information and Telecommunication Engineering, Tokai University, Minato City, Tokyo, Japan
[2]Sch. of Information and Telecommunication Engineering, Tokai University, Minato City, Tokyo, Japan
[3]Collaboration Project Section, Minato City, Tokyo, Japan

Abstract - *To enable the citizens to send their safety information using smartphones to the city government or its branches, we are studying and developing Community Care System (CCS). We are taking into account of real circumstances in Minato City, Tokyo, Japan, as a model case of CCS. In addition, we also developing Community Information Network (CIN), which is a network service platform to support CCS. CIN provides CCS with connectivity between citizen's smartphones and servers of city government and at the same time provides access from not only citizen but tourism to the Internet. This paper reports the concept of the CCS, the prototype development of one of CCS functions (services), the concept of CIN, and the prototype development of access point devices of CIN.*

Keywords: Community, Application, Network

1 Introduction

Japanese society is aging rapidly that results in the increase of the number of elderly people that are living alone. Traditionally, people living in a close-knit community have helped each other out. Today, such social ties are becoming weaker particularly in urban areas, which is making it difficult to confirm the well-being of elderly or physically handicapped people on a daily basis. The problem is becoming more and more serious because confirmation of their well-being depends on inefficient ways such as telephone calls or visits made by social workers who are increasingly overstretched and in short supply.

One solution of this problem is to improve the efficiency of collecting and sharing the information on the well-being of these vulnerable people on a regular basis using mobile devices such as smartphones and tablet PCs which are now widely used in Japan. Japan has a well-developed communication infrastructure. According to a report by the Ministry of Internal Affairs and Communications, the percentage of smartphone users aged 60 or older is rapidly on the rise. It was 7.3% in fiscal year 2013 compared to 3.4% in the previous year [1]. The authors expect that this percentage will continue to increase rapidly in the near future as people in their 50s and early 60s, who routinely use their smartphones for business and private purposes, join the ranks of the elderly.

In light of the above circumstances, we are developing, in collaboration with the Minato Ward Office in Tokyo, a community care system (CCS) that is designed to meet local needs for confirming the well-being of the elderly on a regular basis. An experiment is being conducted in an area in Takanawa, Minato city, Tokyo. This area was chosen as a model case because the Minato Ward has a dense population in which the proportion of elderly people is rising rapidly and will come to characterize many other localities in Japan in the near future. This paper reports the concepts of CCS and the prototype development of one of the functions (services) of CCS.

In addition, we are developing a Community Information Network (CIN) which supports CCS application by providing wireless LAN (Wi-Fi) connections, in the area. The paper also reports the concepts of CIN and the prototype development of access point (AP) devices.

The rest of the paper is organized as follows. Section 2 introduces the concept of CCS and the prototype development of the system which is one of the functions (services) of tCCS. Section 3 introduces the concept of the CIN and the prototype development of AP devices. Lastly, Section 5 summarizes this paper.

2 Community Care System, CCS

2.1 Concept of CCS

In the near future, an insufficiency of the support staffs becomes more serious due to progression of aging. The number of people who have no support person for the daily or emergency situations will be increased. We develop CCS to tackle the above problems by realizing the following functions 1) and 2).

1) Safety confirmation for elderly and/or handicapped people

The safety confirmation for elderly people and handicapped people is made by telephone and patrols by social workers. However, the ways are insufficient solutions in the metropolitan area such as Minato city. There is another way to apply services which are provided by network carriers or

communication device manufacturers. However, the installation and running costs are so expensive that the way is difficult to apply in the areas. Therefore, an alternative way which can be realized at a low cost is needed.

2) **Local information notifications by the local government:**

Local information such as daily information, event announcements, and emergency notifications are provided by message boards, handbills, and radio broadcast in general. The information provision should be realized more quickly and certainly.

2.2 Outline of the safety confirmation and notification system

We are planning to build up the CCS in step-by-step manner. As the first step, we develop the prototype of the safety confirmation and notification service system. This system is server-client based and application software is defined in each side of server and user.

This system assumes that the necessary servers are installed in the ward office and/or its branch offices and that users, i.e., elderly people, have smartphones. The administrator at the ward office or a branch office sends users a message containing an inquiry about their well-being and a list of possible answers. The user can send a response to the question by simple touch operation (selection of answer) on his/her smartphone. Then, the administrator confirms responses from the users and takes appropriate action if needed. In an actual operation, the messages will be sent regularly. Since the system facilitates the workload reduction for the ward office staffs and social workers, it can contribute to realizing the function 1) in Section 2.1. In addition to the inquiry messages, daily information, event announcements, and emergency notifications are also sent to the citizen. By sending those information by the system, to the citizen, the citizens can share the information more quickly and certainly. Hence the system can contribute to realizing the function 2) in Section 2.1.

2.3 The prototype of the safety confirmation and notification system

The prototype of the safety confirmation and notification system is configured as shown in Fig. 1. Our experimental system uses a virtual machine on Amazon Elastic Compute Cloud (Amazon EC2), which is an IaaS (Infrastructure as a Service). The OS of the server is Ubuntu 14.04 LTS. A Web server, Apache2, and a database, MySQL, run on it. The server side application is implemented using PHP (Hypertext Processer). The user needs to have a smartphone or a tablet that runs on Android (version 4.0 or higher), and also to be in an environment that allows access to the Internet. The implementation on other platforms is a further issues to be studied. The implementation of the application on other OSs will be studied in the future.

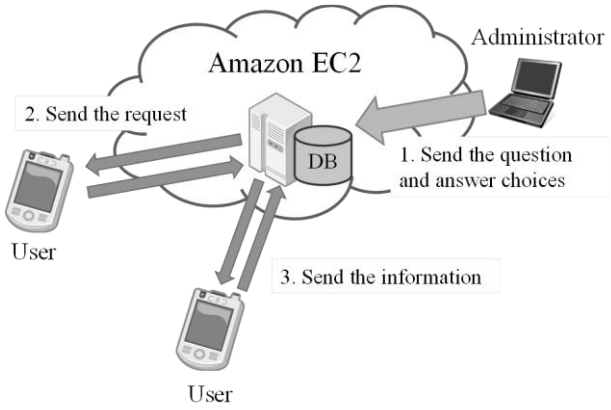


Fig. 1 System prototype configuration of the safety confirmation and notification system

2.3.1 Server side application

The administrator logs into the system by entering his/her user name and password. After that, Admin screen #1 (Fig. 2) appears. It has "Notify", "Confirm" and Register" buttons. The administrator registers users in advance on Admin screen #2 (Fig. 2). The server identifies a user terminal using the MAC address of the Wi-Fi communication device mounted on the terminal because a MAC address is unique to each terminal.

The administrator composes a request for information about the user's general condition on Admin screen #3 (Fig. 2). The administrator enters a question and possible answers on this screen and can also include information about precautions in daily life and upcoming events in the request message. When the administrator clicks the Send button, the request message is sent to user terminals.

The administrator can read user responses on Admin screen #4 (Fig. 2). The information on this screen is updated in real time as responses arrive. The list of response messages can be sorted by user name, question, answer, and response date and time.

2.3.2 Users side application

Messages requesting information about the well-being of users are sent to user terminals via the Internet. The request information is automatically displayed on the user terminal screen. The user can reply by simple touch operation because a list of possible answers to choose is displayed. For example, to the question "Please answer your health condition.", possible answers are: "Reply 1: Good", "Reply 2: Not good", "Reply 3: I need consultation / I have a question" and "Reply 4: Nearby people are in trouble ."

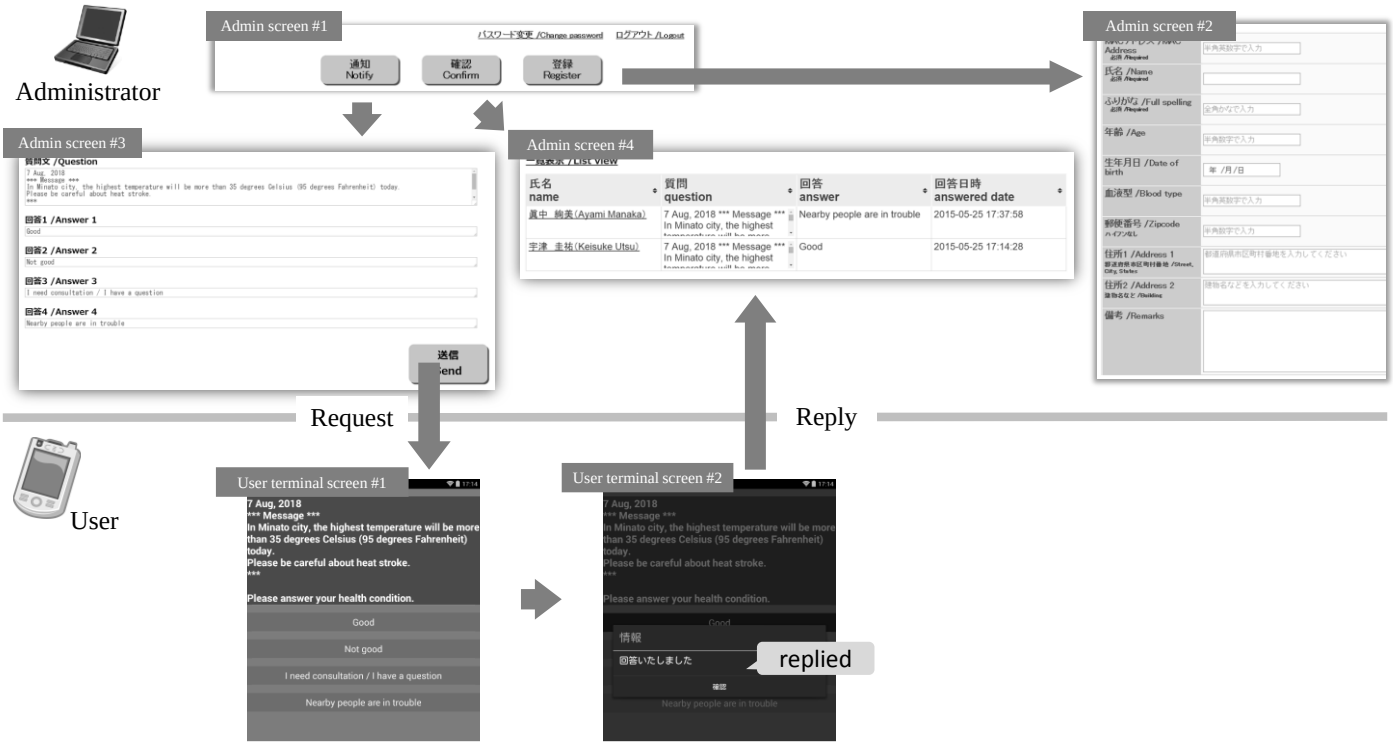


Fig. 2 Screen transition of CCS application

2.4 Example of use

Examples of messages, questions, and answer choices assumed to be sent are as follows.

Example 1: general notifications and questions

- Message: In Minato city, the highest temperature will be more than 35 degrees Celsius (95 degrees Fahrenheit) today. Please be careful about heat stroke. Please answer your health condition.
- Answer choice 1: Good
- Answer choice 2: Not good
- Answer choice 3: I need consultation / I have a question
- Answer choice 4: Nearby people are in trouble

Example 2: event announcements and questions

Message: In Minato city, x festival will be held at x park at 10:00am, Today. Please come and visit. Please answer your health condition.

- Answer choice 1: Good
- Answer choice 2: Not good
- Answer choice 3: I need consultation / I have a question
- Answer choice 4: Nearby people are in trouble

Example 3: emergency notifications and questions

- Message: We have just experienced a big earthquake in Minato City. Tsunami will not occur. Please answer your safety.
- Answer choice 1: Safe and not injured
- Answer choice 2: Minor injured but can move
- Answer choice 3: Can't move
- Answer choice 4: Nearby people are in trouble

Example 4: emergency notifications and questions

- Message: A typhoon is coming soon. Please refrain from unnecessary going out. Please answer your health condition.
- Answer choice 1: Good
- Answer choice 2: Not good
- Answer choice 3: I need consultation / I have a question
- Answer choice 4: Nearby people are in trouble

2.5 Issues to be improved

In the future, we plan to improve the following issues of the prototype system.

- Each number of people who made each choice should be visible on the administrator side.
- In particular, since a person may be identified by using information such as date of birth, there are some citizens who hesitate to register such personal information.
- The safety information of the citizen who requires daily support should be highlighted.
- The system should enable the citizen to upload optionally his present location which is obtained by GPS (Global Positioning System) on his terminal.

- Since city newsletters are published every 10 days, the inquiry and notification message should be sent with them. It can be expected that the administrator obtain the responses from the citizens more certainly.

3 Community Information Network, CIN

3.1 Needs of CIN

As locally launched communication network services, access points have been equipped to provide tourists with public Wi-Fi services in Japan [2][3]. On the other hand, as a technology, IEEE802.11s mesh network has been studied [4]. Hence, CIN will make the most of those services. We consider that there are following 2 requirements to realize the CIN.

1) The CIN makes it possible for the residents and/or tourists to be provided the internet connection service by deploying Wi-Fi APs. As a result, CIN makes it possible for the users to use the application everywhere in the city even when the infrastructures of communication carriers are unavailable due to disasters.

2) Even when the infrastructures of communication carriers are unavailable due to disasters, CIN makes it possible for the users to use the application.

3.2 Concept of CIN

To satisfy the requirements and at low costs, we are considering CIN as follows.

First let us consider the AP. In Merry Road Takanawa area, many streetlamps have been equipped by Minato-city government. An example of a streetlamp is shown in Fig.3. As an interval between any two streetlamps is around 20 meters, we assume to equip the APs on these streetlamps. Each AP is assumed to be furnished with 2 radio interfaces. The one interface runs ad hoc mode to communicate with other APs, and the other one runs infrastructure mode to communicate with user terminals in its coverage.

Next, the CIN is composed of a number of sub-areas, each of which is composed of several APs, as shown in Fig. 4. An AP in each sub-area is a gateway to other sub-areas and/or the external network (Internet). How to realize the connections among sub-areas and between the CIN and the internet needs further study. The network performance of broadcast and unicast communication among APs in for a model assuming of the CIN sub-area by network simulations are shown in [5][6].

Fig. 3 An example of a streetlamp at Merry-road Takanawa

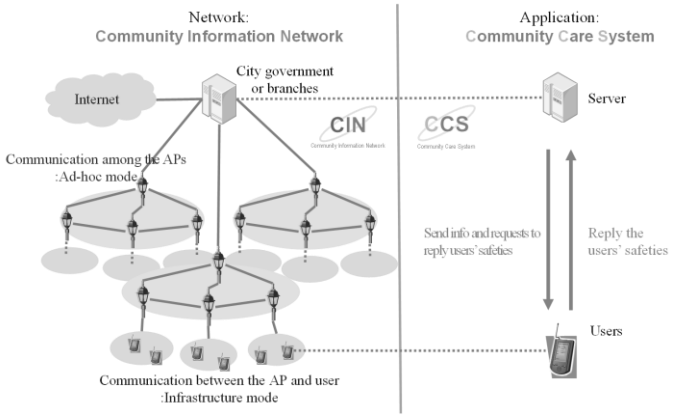


Fig.4 CIN and CCS application

3.3 Prototype development of AP

To facilitate the development of the practical APs to deploy CIN in the future, we develop and evaluates a prototype AP by use of a Raspberry Pi board [7]. The board is a low cost and single-board microcomputer and equips an ARM processor, developed by Raspberry Pi Foundation in the United Kingdom. Since the board is low price, Linux-compliant, and it runs with a low energy consumption, we can easily implement the AP functions on it.

Two wireless LAN adaptors are installed in each AP. The one is running as the ad hoc mode for the inter-AP channel (IF0) for multi-hop LAN, and the other one is running as the infrastructure mode for connections with the user terminals (IF1). This study adopts Raspberry Pi Type B as shown in Fig.5. We develop a set of 4 APs by Raspberry Pi type B boards with the spec shown in Table 1. The board is credit card sized. It has a CPU, 2 USB ports, a HDMI port, and a Wired LAN port. The OS of the board is Raspbian OS, on top of which necessary functions are implemented. Two USB connected wireless LAN adaptors are installed on the board. The adaptor for IF0 is running on the ad hoc mode and uses IEEE802.11g as the MAC layer. The adaptor for IF1 is running on the infrastructure mode by hostapd [8], and uses IEEE802.11g as the MAC layer.

A multi-hop network is constructed by the static routing as shown in Fig.6. Actually, a dynamic routing protocol should be applied to the routing. A suitable routing protocol will be adopted in the further study. In the experiment, the number of APs between the user terminals (n) is assumed to be 2, 3, or 4, i.e. the number of hops in the inter-APs channel (n-1) becomes 1, 2, or 3, For each of three cases, the throughputs are evaluated. User terminals x and y are connected to AP_0 and AP_n, respectively. The experiment is made in indoors. The distance between the adjacent APs and that between the AP and the user terminal are 10m. User terminal x generates traffics using Netperf [9]. In the measurement of the TCP throughput, the maximum volume of the load (i.e. line rate) is generated, then the throughput is calculated by the volume of the exchanged traffic. In the measurement of the UDP throughput, the maximum volume of the load is generated, then the throughput is calculated by the volume of the received packets at User terminal y.

The throughput of TCP and UDP were observed for 60 samples, the maximum 5 and minimum 5 samples were discarded, and then 50 samples were evaluated. The mesurement results are shown in Fig. 7. The deviation bars in the graphs show the standard deviations of the evaluated values. Here, the throughput values were decreased as the number of hops increased. This was due to contention resolution mechanisms in CSMA/CA (Carrier Sense Multiple Access Collision Avoidance) on the wireless LAN MAC layer.

In the future, we plan to develop the practical and cost-effective APs and also study and implement a suitable dynamic routing protocol on them. Finally we will execute field tests by deploying the APs in the town street.

Table 1 Spec of Raspberry Pi Type B board

CPU	BCM2835, 700MHz
RAN	512MBytes
USB port	2 ports
Board size	3.370 in × 2.22 in
Price	USD 35.00

Fig. 5 Raspberry Pi board

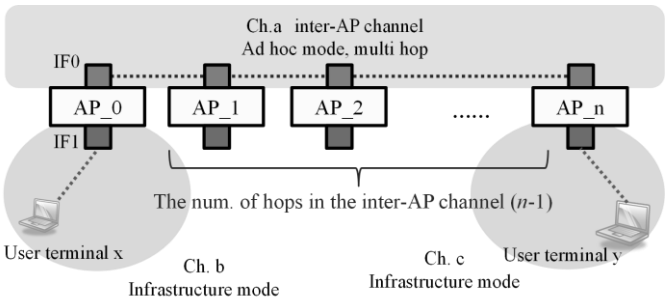


Fig. 6 Network composition of the experiment

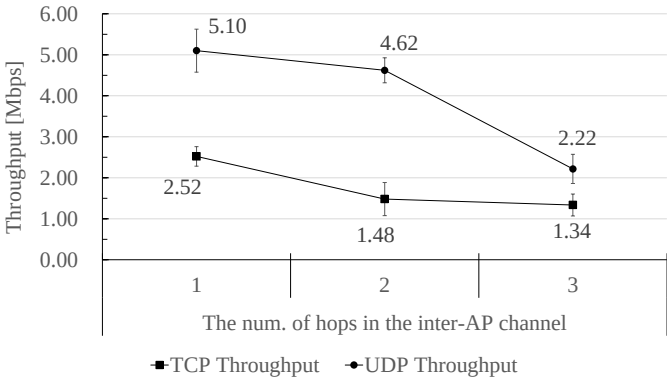


Fig. 7 Experiment result of TCP and UDPthroughput

4 Summary

We are studying to develop CCS and CIN taking the real situations in Takanawa, Minato city, Tokyo, as a model case. This paper introduced the concept of CCS and CIN. As CCS, the paper showed the development and test installation of the safety confirmation and notification system which is one of the functions of CCS. In the future, we plan to make the test installation of the system and to improve it. As CIN, the paper showed the prototype development of the AP devices. The multi-hop wireless network was configured and the throughput was evaluated in the experiment. In the future, we plan to make field tests using them.

5 Acknowledgments

This study has been supported by the following funds.

- COC (Center of Community), Ministry of Education, Culture, Sports, Science and Technology, Japan
- KAKENHI 26420372, Japan Society for the Promotion of Science.

6 References

[1] Ministry of Internal Affairs and Communications , Japan, http://www.soumu.go.jp/johotsusintokei/statistics/data/140627_1.pdf, 2014

[2] Kobe City, "KOBE Free Wi-Fi," http://www.city.kobe.lg.jp/information/press/2014/07/20140704142001.html

[3] Fukuoka City, "Fukuoka City Wi-Fi," http://www.city.fukuoka.lg.jp/wi-fi/

[4] Shiro Sakata, Akira Yamada, Hiroyuki Iizuka, Tetsuya Ito, "Trend on Wireless LAN Mesh Network," J. IEICE, Vol.92, No.10, pp.841-846, 2009

[5] Ayami Manaka, Keisuke Utsu, Chee Onn Chow, Yasuhiro Nozawa, Minoru Fukuzaki, Hiroshi Ishii, An Application of Broadcast Based Information Sharing System to A Community Information Network", MJIIT-JUC Joint International Symposium 2014, 1570022779, 2014

[6] Ayami Manaka, Tomomi Itoh, Yasuhiro Nozawa, Chee Onn Chow, Minoru Fukuzaki, Hiroshi Ishii, Keisuke Utsu, "Performance Evaluation of a Community Information Network for a Daily Life Support System", the 2015 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'15), 2015

[7] Raspberry Pi, https://www.raspberrypi.org/

[8] hostapd, http://w1.fi/hostapd/

[9] Netperf, http://www.netperf.org/netperf/NetperfPage.htm

Performance Evaluation of A Community Information Network for A Daily Life Support System

Ayami Manaka[1], Tomomi Itoh[2], Yasuhiro Nozawa[3],
Chee Onn Chow[4], Minoru Fukuzaki[2], Hiroshi Ishii[2], and Keisuke Utsu[2]
[1]Graduate Sch. of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan
[2]Sch. of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan
[3] Collaboration Project Section of Minato City, Minato City, Tokyo, Japan
[4]Faculty of Engineering, University of Malaya, Kuala Lumpur, Malaysia

Abstract - *We are studying a daily life support system for citizens which consists of Community Care System (CCS) application and Community Information Network (CIN). CIN provides CCS application with network service by use of wireless LAN (Wi-Fi) connections. The study takes into account of the actual circumstance in Minato city, Tokyo, Japan, as a model case. The CIN is composed of multiple Wi-Fi access points (APs), multi–hop links among APs, and wireless terminals under each AP. This study evaluates the network performance of broadcast and unicast communication among APs for a model of CIN by network simulations.*

Keywords: Community Information Network, Wireless LAN, Simulation

1 Introduction

We are studying a daily life support system for citizens which consists of Community Care System (CCS) application and Community Information Network (CIN) which supports the application by providing wireless LAN (Wi-Fi) connections. CCS application and CIN are assumed to be deployed in Takanawa area, Minato-city, Tokyo, Japan, as a model case [1][2]. This paper shows the performance evaluation of CIN by network simulations.

As a preliminary evaluation, we assume that the CIN is made of only "one" sub-area and that it is closed (without connection the external network). We assume two types of communications. The one is the broadcast communications, as shown in Fig.1. A public office and/or its branch office in a sub-area generates information, and it is delivered to all the nodes in the sub-areas. The other one is the unicast communication between the public office and/or its branch office and the user terminal in the subarea, as shown in Fig.2. In this paper, the performances of broadcast and unicast communication are evaluated by the network simulation.

Sections 2 and 3 show simulation evaluation on several performance items for both the broadcast and unicast communications. Lastly, Section 4 concludes the paper.

2 Broadcast communication in the CIN sub-area

Here, we consider the case where information is broadcast in the network without any specific destination. The broadcast is very useful to distribute some information over the network. As shown in Fig.1, the information generated at a city government or its branch office is firstly transferred to the APs, and then each AP distributes the information to the user terminals connecting to the AP. In addition to the above case, there are cases where the user terminals generate the information, i.e. the information is generated by unspecific terminals under the APs. Anyway, first of all, the generated information should be certainly delivered to each AP.

The content types of the information will be images, voices, and text data, whose size usually exceeds the payload size of a packet so that the information are divided into multiple packets. Therefore, even if one of the packets of the information is lost (an unreached packet), the information cannot be reassembled at the receiving side. One of the popular broadcast delivery methods; Simple flooding (SF), and its improved methods have been used usually in mobile ad hoc networks [3][4]. However, these methods cannot perform the sufficient performance such as reachability and efficiency because these method cannot complement unreached packets. On the other hand, BBISS (Broadcast Based Information Sharing System) [1] can deliver the information composed by multiple packets with high efficiency and reliability. The system can run on the wireless LAN in ad hoc mode, and it can deliver information using broadcast transmission.

The outline of BBISS is as follows. Below "send" means "broadcast" in this section. As a matter of fact, the information is generated by nodes connecting to APs. In our assumption, since the channels among APs and that for the user terminals to APs are independent, the information can be assumed to be generated by APs. First of all, the information initiator node which starts information transmission divides the information into multiple packets (Here, since this paper focuses on the communication among APs, any AP(s) will be the initiator node), and sends the packets sequentially with a

fixed time interval which is determined by *send_interval* (as Sending state in Fig.3). Then the packets are received by a node around the initiator node (as Receiving state in Fig. 4).

A node that has received all packets does not immediately relaying them but waits during a random period. During the period, the node listens to other nodes and counts the number of nodes relaying the same information that it has. If the number of relaying nodes reached to a predetermined threshold (*relay_threshold)*, the relaying by itself is canceled. The operation can save redundant relays and reduce traffic load. After the period, if the number of relaying nodes is not reached to *relay_threshold*, the node relays the information (as Sending state in Fig.3).

In the architecture, unreached packets may possibly be complemented by redundancy of broadcast transmission. In the case where the unreached packets cannot be complemented (as the right bottom node in Fig.5), a NACK (Negative Acknowledgments) based retransmission control is operated as shown in Fig.6. The number of trials to send retransmission request packet is limited to a predetermined threshold (*req_threshold*).

In the following we evaluate the performance of broadcast communication among APs by the network simulation.

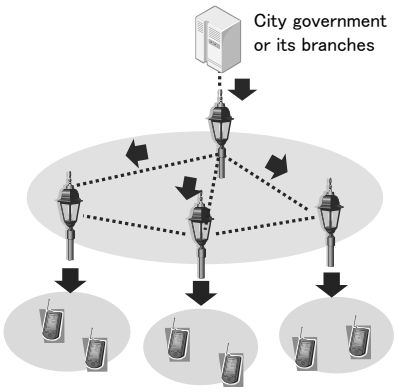


Fig. 1 Broadcast communication on the CIN sub-area

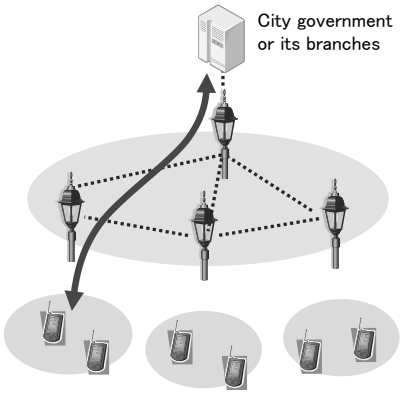


Fig.2 Unicast communication to the user terminal on the CIN sub-area

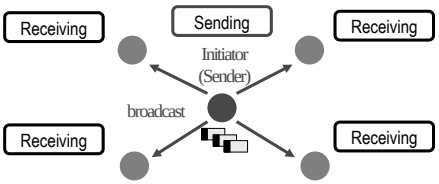


Fig.3 Sending and Receiving states in BBISS

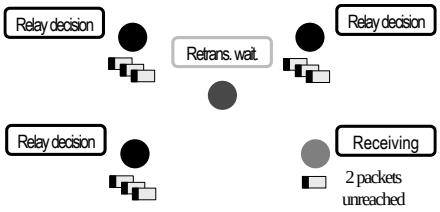


Fig.4 Relay decision state in BBISS

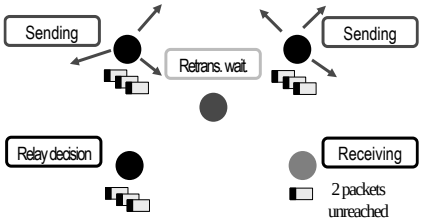


Fig.5 Relay decision and Sending states in BBISS

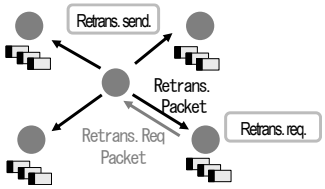


Fig.6 Retransmission operation for complementing unreached packets in BBISS

2.1 Condition of the simulation

We use OPNET Modeler 17.5 as a network simulator [5]. The simulation area size can cover Merry Road Takanawa Street. The APs are assumed to be equipped there covering the street in about 100m intervals, as shown in Fig. 7. The radio communication range (transmission radius) is 150m. The MAC layer protocol is IEEE802.11a, and the data rate is 54Mbps.

AP1, 6, and 8 are in one hop radio range of AP0. AP2 and 7 are in the two-hop range from AP0. AP3 is reachable with three hops from AP0. AP4 and 5 are four hops away from AP0. The radio channel for inter-APs and that for the AP to user terminals are assumed not to interfere each other. In the simulation, traffic is assumed to be generated from the AP.

Fig. 7 Assumption of the APs' placement

The simulated information transfer methods are SF and BBISS, and the performances of them are compared. For SF, we set up two parameter sets: SF-A and SF-B are simulated. As SF-A, *send_interval* is set 0.032s, and the random waiting time for the packet relaying is randomly selected from the range of (0.032, 0.320)s. As SF-B, *send_interval*: 0.032s, and the random waiting time: in the range of (0.064, 0.640)s. For BBISS simulation, *send_interval* is set 0.032s, and the random waiting time for the information relaying is chosen from the range of (0.032, 0.320)s. The other parameters for BBISS are set according to [6], that is, *relay_threshold* is set 2, and *req_threshold* is set 3.

As described before we assume information is generated by APs. The numbers of APs generating the information simultaneously (source APs) are 1 (AP0, in Fig.7), 2 (AP0 and AP2), or 4 (AP0, AP2, AP4, and AP7). The larger the number is, the heavier the traffic load is. The data size of the information is 100kByte, and *payload_size* (the payload size of each packet) is 1024 Byte, i.e. one information is composed of 100 packets. The number of information transmission is 10.

2.2 Evaluated items

The following 2 items are evaluated.

(i)The number of information receiving APs

The number of APs that has received information successfully is calculated for each generated information. Here, the source APs are not included. Then, the numbers for all generated information are averaged. The larger number is, the better performance is.

(ii) Information delivery time [s]

The time between the information is generated at the source AP and received at the other APs (when all the packets belonging to the same information are received) is calculated. The AP could not receive the whole information are eliminated. The times for all receiving APs are averaged. The smaller the value is, the better the performance is.

2.3 Simulation results and discussions

(i)The number of information receiving APs

The simulation result is shown in Fig. 8. The deviation bar in the graph means the standard deviation. In SF, both SF-A and SF-B showed that as the number of source APs increased, the number of information receiving APs decreased due to data frame collisions. On the other hand, the numbers of information receiving APs are almost 8 regardless of the number of source APs. In other words, BBISS can deliver the information to almost all the APs. Since the AP relays the information just after the AP receives whole information, the traffic load is lighter than SF.

(ii) Information delivery time [s]

The simulation result is shown in Fig. 9. The deviation bar in the graph means the standard deviation of the delivery time for each generated information (the case which has no difference of the delivery time is eliminated). In SF, SF-A showed about 3s, and SF-B showed about 6s. On the other hand, BBISS showed 7-9s, that was 1-3s longer than SF-B. Here, in the case that the number of source APs was 4, the delivery time was longer than those of 1 and 2 cases. It was because there were some unreached packets and the retransmissions were operated. Therefore, the delivery time for the latter case was lengthened.

The simulation results can be discussed as follows. In the broadcast communication in CIN, the generated information should be certainly delivered to each AP, i.e. the most important metric for the delivery methods is the reachability. The result showed that BBISS was proved to perform higher information reachability regardless of the instantaneous traffic load. Therefore, BBISS has higher applicability for the broadcast information delivery among the APs than SF. However, the delivery time for BBISS was longer than that for SF. The delivery time shortening of BBISS is a further issue.

The simulation environment in this paper assumed to have rather small number of APs, or the low node density. Our previous paper, however, shows that BBISS in the high node density (with high redundancy) can achieve better delivery performance than that in the low node density. We have to study the optimum number and placement of the APs in the future. In case where we adopt broadcast transfer method in CIN, we can conclude that BBISS is the appropriate candidate.

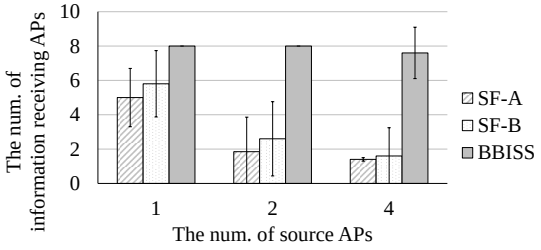


Fig. 8 Simulation result for the average number of information receiving APs

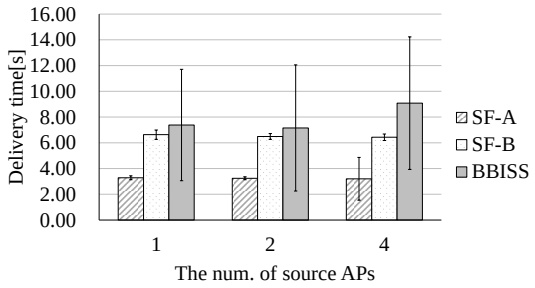


Fig. 9 Simulation result for the average delivery time

3 Unicast communication in the CIN sub-area

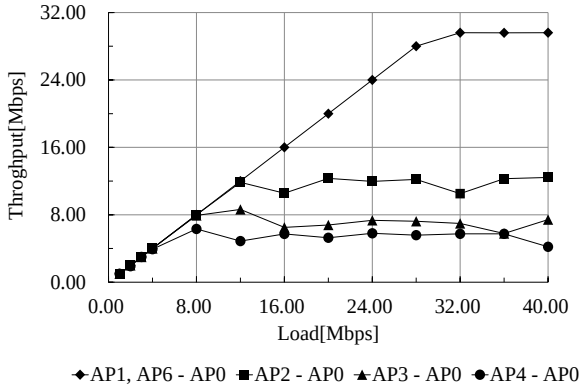
In the section, we evaluate throughput of unicast communication in the CIN sub-area. APs' placement is the same as Section 2.1. We assume that any unicast communication is made between AP0 and other APs, that is, the gateway node is AP0, which is located in Takanawa police station. The routing protocol for multi-hop communications among the APs is Optimized Link State Routing (OLSR) [7] that is pre-installedon the simulator. However, the appropriate routing protocol among the APs is our further study.

3.1 Evaluation 1: Evaluation of the UDP maximum throughput among the APs

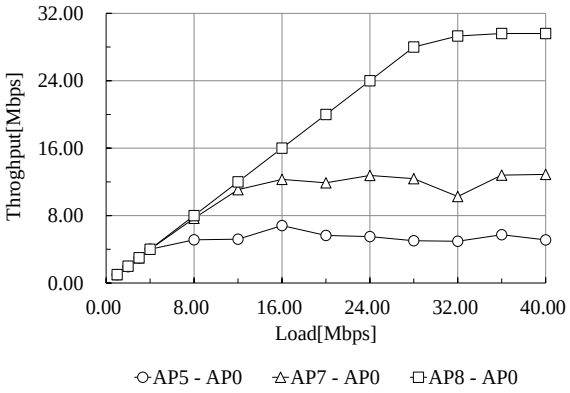
We evaluate the UDP maximum throughput between AP0 and each AP. No background traffic is assumed. In the evaluation, each AP (AP1, 2, ..., or 8) generates the traffic load to AP0 by constant bit rate (CBR) with the bit rates 1.00, 2.00, 3.00, 4.00, 8.00, …, or 40.00Mbps. The packet size is 1400Byte which assumes a normal packet size of about Web browsing using HTTP (Hyper Text Transfer Protocol).

The evaluation result is shown in Fig.10. To understand the figure clearly, the throughputs between AP0 and each of AP1, 2, 3, 4, and 6 are shown in Fig.10 (a). The throughputs between AP0 and each of AP5, 7, 8 are shown in Fig.10 (b). The throughput is about 29Mbps between AP0 and each of AP1, 6, and 8 which are in one-hop neighboring area of AP0. The throughput is about 12Mbps between AP and each of AP2 and 7 which are two-hop neighboring area of AP0. The throughput is about 7Mbps between AP0 and AP3 that is three hops away from AP0. Then, the throughput between AP0 and AP4 that is four hops away from AP0 is about 5Mbps.

The evaluation results show that the network model covering Merry Road Takanawa area can perform at least 5Mbps throughput. We can guess the comparable throughput can be expected for the comparable scale with our model.



(a) Throughput between AP0 and AP{1, 2, 3, 4, 6}



(b) Throughput between AP0 and AP{5, 7, 8}

Fig.10 Evaluation 1: UDP maximum throughput between AP0 and each of AP1-8

3.2 Evaluation 2: Evaluation of the HTTP communication performance

To know the characteristics of actual communication service, we evaluate the HTTP communication performance from the user terminals under the APs. As mentioned before, since the radio channel for inter-APs and that for the AP to user terminals are assumed not to interfere each other, HTTP request is assumed to be generated from the AP, i.e. each AP is a assumed to be the user terminal(s). A web server is assumed to be AP0.

Simulation duration is 600 seconds. During the simulation, each user terminal sends the web page download requests to the web server based on a built-in communication model

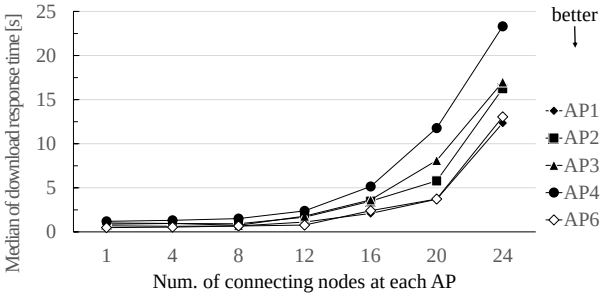
named "Heavy Browsing", which is shown in Table 1. The time interval of the download request transmissions is based on exponential distribution (average $1 / \lambda = 60$ [s]). The number of user terminals under each AP is set 1, 4, 8, …, or 24. Since the total number of APs excluding the AP0 is 8 and we assume same number of terminals exist per AP, the number of user terminals in the entire network is 8, 32, 64, ..., or 192.

We evaluate the expected download response time at a user terminal. Here, the download response time means the time between a HTTP request packets is generated and the requested contents is downloaded completely at the AP. The download response time is observed in each AP. The median value of download response time of all communications made during the simulation (we used the median value in order to eliminate the effect of specifically large value). The evaluation result is shown in Fig.11. To show the results clearly, the results for AP1, 2, 3, 4, and 6 are shown in Fig.11 (a), and those for AP5, 7, and 8 are shown in Fig.11 (b). In the cases for the number of user terminal was smaller than 16 under each AP, download response time was expected within 5 seconds in any APs. On the other hand, in the cases for the number of the user terminal under the AP was more than 16, the download response time was drastically increased.

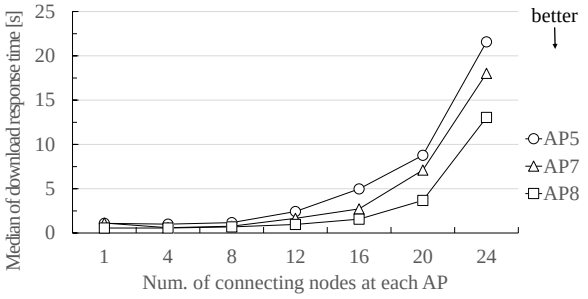
Below, the expected HTTP throughputs against the number of connecting nodes under each AP are evaluated. The throughput is calculated as total amount of successfully received web page data divided by the total download response time at the AP. To show the results clearly, the throughputs observed in the terminals under AP1, 2, 3, 4, and 6 are shown in Fig.12 (a), and those under the AP5, 7, and 8 are shown in Fig.12 (b). The results showed that as the number of user terminals under the AP increased the expected throughput in the user terminal decreased.

Table 1 Contents included in a web page downloaded by an user terminal for "Heavy Browsing" model

Contents	Number of data
Data of 1000Byte, assuming text data, etc.	1
Data size of uniform distribution in the range of (500, 2000)Byte, assuming a medium-sized image	5
Data size of uniform distribution in the range of (10000, 350000)Byte, assuming a short video	2

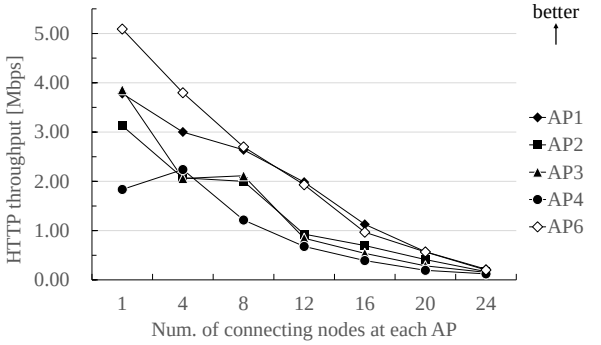


(a) AP1, … , 4, and 6

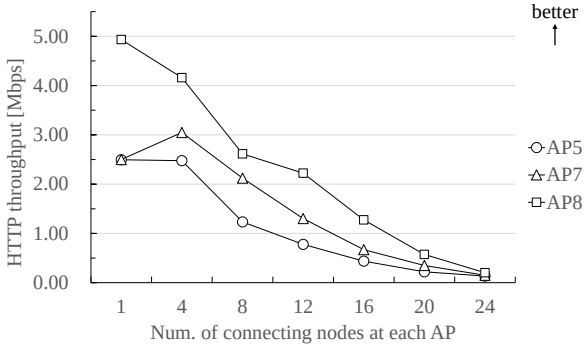


(b) AP5, 7, and 8

Fig.11 Evaluation 2: HTTP download response time



(a) AP1, 2, 3, 4, and 6



(b) AP5, 7, and 8

Fig.12 Evaluation 2: expected HTTP throughput at user terminals

The simulation result of the number of pages downloaded successfully in the network is shown in Fig.13 for reference. As the number of user terminals under each AP increased, the number of pages successfully downloaded was supposed to increase linearly. However, in the case for the number of communication nodes under each AP was more than 16, the number of pages successfully downloaded did not increase linearly. Finally, in the case for the number of communication nodes under each AP was more than 16, the number of pages successfully downloaded decreased, therefore the network falls into the congestion.

Moreover, we can guess the comparable HTTP throughput and download response time can be expected for the comparable scale with our model (comparable network scale with this study and similar number of the user terminals under each AP).

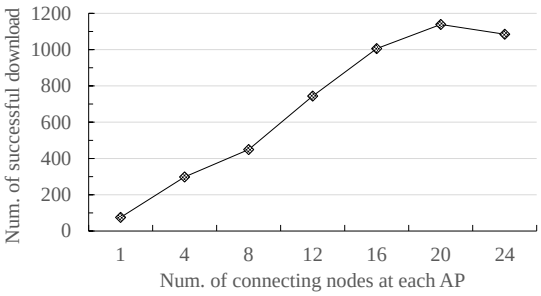


Fig.13 Evaluation 2: Number of successful download

3.3 Evaluation 3: Evaluation of the throughput of uniform loads to each AP

To know the fairness among each pair of AP0 and other AP and maximum throughput when multiple connections are set up simultaneously, we study the throughput when the UDP load is uniformly given among each AP pair. The uniform UDP load of Constant Bit Rate (CBR) is generated from AP0 (gateway AP) to each AP or from each AP to AP0 and then the throughput is evaluated per an AP pair.

The evaluation result is shown in Fig.14. The transmission from AP0 to each AP is described as "down" (assuming downloading from AP0). Also the transmission from each AP to AP0 is described as "up" (assuming upload to AP0). The dotted line in the figure means the ideal throughput, which is the line of load = throughput. The evaluation result showed that the ideal throughput was achieved in the cases where the load was 1.2Mbps or less in both cases for "up" and "down". In other words, the network model can fairly perform 1.2Mbps throughput to each AP. In addition, the similar throughputs can be expected on a network, which has the similar network scale in this study and has the similar number of the user terminals under each AP.

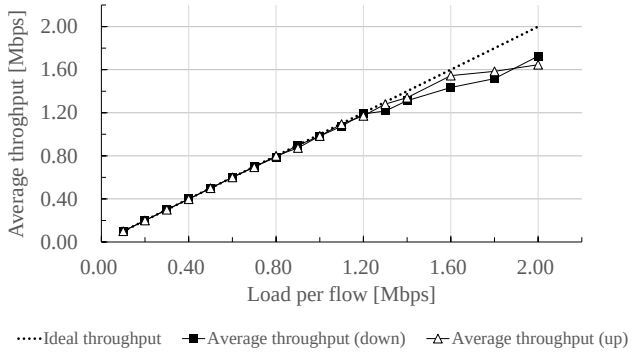


Fig.14 Throughput for each AP against an uniform UDP load for APs

3.4 Discussions

The performance for the unicast communication in CIN is evaluated in this section. The simulation results shows that we can expect the performance as follows.

- Evaluation 1: the network model can perform at least 5Mbps for the UDP maximum throughput between AP0 and each AP.
- Evaluation 2: the network model may fall into congestion when there are greater than 16 user terminals browsing by HTTP heavily under each AP.
- Evaluation 3: the network model can fairly perform 1.2Mbps for the UDP throughput to each AP

We can guess the comparable throughput can be expected to perform on a network, which has the comparable scale with our model.

4 Conclusion

This paper discussed Community Information Network (CIN) which is assumed to be deployed in Merry Load Takanawa Street, Minato-city, Tokyo, Japan, as a model case. The performance are evaluated for both broadcast communications using BBISS among APs and the unicast communications by the network simulations.

As a result, for broadcast communications among APs, The result showed that BBISS was proved to perform higher information reachability regardless of the instantaneous traffic load. Therefore, BBISS has higher applicability for the broadcast information delivery among the APs than SF. However, the delivery time for BBISS was longer than that for SF. The delivery time shortening of BBISS is a further issue.

For the unicast communications, we discussed the expected communication performance of the communication among

APs. The fisrt evaluation, related to the UDP maximum throughput among the APs, showed that the network model can perform at least 5Mbps for the UDP maximum throughput between AP0 and each AP. The second evaluation, related to the HTTP communication performance, showed that the network model may fall into congestion when there are greater than 16 user terminals browsing by HTTP heavily under each AP. The third evaluation, related to the throughput of uniform loads to each AP, showed that the network model can fairly perform 1.2Mbps for the UDP throughput to each AP. We can guess the comparable throughput can be expected to perform on a network, which has the comparable scale with the simulated model. Those results gave a good guide principles for designing the real CIN.

In the future, we plan to develop actual CIN environment and CCS over it in Takanawa area. For the purpose, we will evaluate the performance of actually implemented CIN by installing the AP device in Takanawa area.

5 Acknowledgments

This study has been supported by the following funds.

- COC (Center of Community), Ministry of Education, Culture, Sports, Science and Technology, Japan

- The Science Research Promotion Fund, The Promotion and Mutual Aid Corporation for Private Schools of JapanThis study has been supported by KAKENHI 26420372, Japan Society for the Promotion of Science.

6 References

[1] Ayami Manaka, Keisuke Utsu, Chee Onn Chow, Yasuhiro Nozawa, Minoru Fukuzaki, Hiroshi Ishii, "An Application of Broadcast Based Information Sharing System to A Community Information Network", MJIIT-JUC Joint International Symposium 2014 (MJIIS2014), 1570022779, Malaysia, 2014

[2] Ayami Manaka, Akio Ogata, Hirohide Matsuzaka, Hayato Taniguchi, Masaya Nomoto, Minoru Fukuzaki, Hiroshi Ishii, Yasuhiro Nozawa, and Keisuke Utsu, "A Concept of Community Care System and Community Information Network", the 2015 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'15), 2015

[3] Brad Williams, Tracy Camp, "Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks," Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 194-205, 2002

[4] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, Jang-Pinig-Sheu, "The Broadcast Problem in a Mobile Ad Hoc Network," Wireless Networks Volume 8, Springer, Kluwer Academic Publishers, pp. 153-167, 2002

[5] The network simulator OPNET, http://www.riverbed.com/products/performance-management-control/opnet.html

[6] Sayuri Wada, Hiroshi Ishii, Hiroaki Nishikawa and Keisuke Utsu, "An Optimization Study on Broadcast Based Information Sharing System (BBISS)", the 2014 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'14) , pp.485-489, 2014

[7] T.Clausen, P,Jacquet, "Optimized Link State Routing Protocol (OLSR)", Request for Comments: 3626, http://www.ietf.org/rfc/rfc3626.txt

A Study on Secure Communication Method Using Secret Sharing Schemes over MANET

Kei Kobayashi[1], Yosuke Totani[2], Keisuke Utsu[2], and Hiroshi Ishii[2]
[1]Graduate School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan
[2]School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan

Abstract – *A Mobile Ad-hoc Network (MANET) has been studied as a technology for building infrastructure independent and autonomously distributed controlled network in the disaster situation* [1]. *However, MANET has many issues. One of the issues to be solved in MANET is security problem. For example, MANET is vulnerable to network sniffing because it is using a wireless communication that can be heard by others and existing security approaches are difficult to be applied to it. In this paper we propose a novel communication method that cannot easily be subjected to network sniffing even in the aspect of security. In addition, we evaluate the proposed method and show its effectiveness.*

Keywords: Mobile Ad-hoc Network, Secret Sharing Schemes, Secure Communication

1 Introduction

A MANET is one of the methods to build a communication in the disaster situation by mobile device (node) such as smart phone or rap top PC. If node needs to send a data packets to another nodes outside its radio range, the data packet is relayed by neighboring nodes within a range. However, MANET has many issues should be solved such as security problems [2]. For example, MANET is vulnerable to network sniffing because it is using a wireless communication and existing security approaches are difficult to be applied in MANET. In this paper we propose a secure communication method that cannot easily be subjected to network sniffing over MANET by using secret sharing schemes.

Section 2 describes relevant studies. In section 3, we explain about our proposed method. Section 4 shows evaluation of the proposed method through network simulation, its effectiveness. Section 5 concludes this paper.

2 Relevant studies

2.1 Simple flooding

Simple flooding is the simplest communication method in the MANET. In this method, a transmission node broadcasts a data packet. If other node than the transmission node receives the packet for the first time, it broadcasts the packets for next hop nodes. This is used for data and key transmission in our approach.

2.2 Secret Sharing Schemes

Secret sharing schemes are methods by which a dealer distributes shares to parties such that only authorized subsets of parties can reconstruct the secret. A typical example of secret sharing scheme is the (k,n) threshold scheme [3][4]. In this scheme, a person who generates a secret splits it into n shares and distributes shares to n members. The secret can be reconstructed if someone of members gathers k shares among n.

3 Proposed method

As described in section I, since MANET uses open radio channel and shall rely on intermediate relay nodes, it is difficult to establish secure communication. To resolve this problem, this paper proposes a secure communication method using secret sharing scheme over MANET. In this section, the details of the proposal is explained. Before entering the detailed explanation, we show the essence of our proposal.

The source node generates a secret information to be sent to the destination. By use of (k,n) threshold scheme, the source node splits the secret into multiple (n) shares and put each share into different data packet. Here, to avoid malusers from easily sniffing and decoding the share, our proposal obfuscates the share by use of HELLO information (node ID, location and sent time) and send it to selected members who will relay the share to the destination. If the destination node collects at least "k" shares from the member nodes, it can reconstruct the secret information. Fig. 1 shows the outline of our proposal.

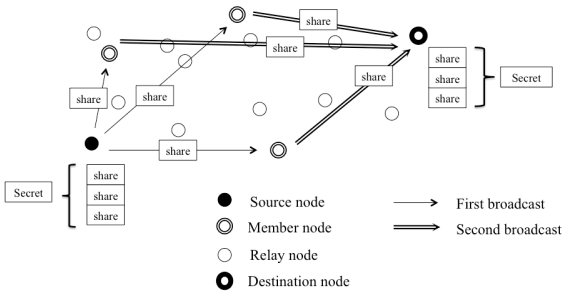


Fig 1. Outline of proposed method

Details of the proposal are shown below

3.1 Procedure of HELLO exchange

Each node exchanges HELLO packets with neighboring nodes to inform node ID, node location, and time stamp of sending. The HELLO penetrates in the network according to the HELLO TTL value. Each node stores HELLO packets submitted by it self in the send HELLO buffer. If the buffer becomes full, oldest HELLO is discarded. Each node stores HELLO packets received from neighboring nodes in the receive HELLO buffer. If the buffer becomes full, oldest HELLO is discarded. We assume that these HELLO packets are broadcast every 5 seconds and its TTL value is 1 or 2. HELLO packets have following format. The format is contained in the payload of layer 3 packet with broadcast address.

Normal data packet has also the packet type as shown in Fig.2.

packet_type	node_ID	location_data	hello_time	ttl	check_sum

packet_type : Type of the packet (0:HELLO packets 1:data packet)
node_ID : Node ID of the source node
location_data : Location information of the node
hello_time : Time stamp when the HELLO packet is generated

Fig. 2 HELLO packet format

3.2 Procedure in source node

3.2.1 Generation of share data

A source node generates data (secret information) to be transmitted to the destination node. It fragments the secret information into n pieces by using (k,n) threshold scheme.

3.2.2 Selection of member nodes

The source node selects n nodes (member nodes) among nodes from which the source node has received HELLO packets by seeking in the receive HELLO buffer

3.2.3 Share obfuscation

Source node obfuscates each share before storing it in the packet payload to prevent inappropriate nodes (non-members) from restoring shares by use of HELLO information of the member node. The obfuscation algorithm is shown below.

$$h(nodeID_{mi}: location\_data_{mi}: hello\_time_{mi}) \oplus \{h(nodeID_{mi}): share\_type: nodeID_s: nodeID_d: share\}$$

Table 1. The description of variables

$NodeID_{mi}$	A node ID of a member node i (sent by HELLO)
$Location\_data_{mi}$	A location information of a member node i (extracted from one of HELLOs sent by node i)
$Hello\_time_{mi}$	A time stamp when HELLO from member node i is generated (extracted from the same HELLO as that of $Location\_data_{mi}$ the selected)
Share_type	A type of a share (0:source → members, 1:members → destination)
$NodeID_s$	A node ID of a source
$NodeID_d$	A node ID of a destination
Share	Concatenation of share itself and the value of k and n.

The algorithm is explained as follows. The source node selects a HELLO message sent by a member node in the receive buffer and extracts its node ID, location data and hello tome. Then the source executes the operation shown above and get the obfuscated share for a member node. And the operation is made for other (m-1) member node.

3.2.4 Transmission of the obfuscated share data

The source node creates m packets each of which carries each obfuscated share data and then broadcasts these packets to the network area. At this time, the value of the *share_type* field of these packets set to 1 to indicate that the packets are data packets. The destination address of the packet layer is "broadcast address". Fig 3 shows the packet format of share data packets.

packet_type	ttl	check_sum	Share_data

Fig.3 Share data packet format

3.3 Procedure of Member nodes

3.3.1 Decoding an obfuscated payload

When a member node receives a packet containing obfuscated share with *packet_type* field = 1, the node knows the packet is a data packet in secure communication and try to decode it. Since the share is obfuscated by HELLO information of a member node, member node refers to own send HELLO buffer, picks up a HELLO and extracts node ID, location and time which are sent by itself.

Then it calculates the hash value of concatenation of node ID, location and time and executes exclusive-OR with the received packet. If the first part (20 bytes from the beginning if using of sha1 hash algorithm [5]) of exclusive OR result matches the hash value of its own ID, the node knows the packet is submitted to itself and picks up the share data from the packet. If not, the node selects the next HELLO from its send HELLO buffer and executes the same calculation in a brute force for decoding the packet.

Lastly if none of its HELLOs can decode the packet, the node knows the packet is submitted to other nodes and discards it or rebroadcasts it to neighboring nodes according to TTL

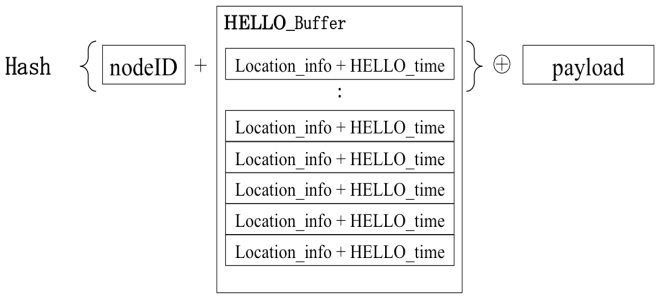


Fig 4. Image of decoding an obfuscated payload

Member nodes which can decodes share refer to *share_type* field. If *share_type* field is 0, the nodes know the share is submitted to itself and move onto next procedure

3.3.2 Broadcasting obfuscated share packets

The member node submits the share to the destination node if the node can decode it section 3.3.1. At this time, the process of obfuscating is executed by use of HELLO information from the destination node in same manner as section 3.2.3. But if the member node never receive HELLO packets from the destination node or does not remain it (receive HELLO buffer overflow), the member node cannot broadcast the share to the destination node. In the case that member nodes broadcast to the destination node, *share_type* field is set to 1.

3.4 Procedure of destination node

3.4.1 Decoding of obfuscated payloads

When the destination node receives obfuscated data packet, the node checks *share_type* field of the packet. If the *share_type* field is 1, the destination node tries to decode the payload as with 3.3.1. The exact calculation is

shown below. After the destination node decode the share data, it stores the share and waits for another share packet.

$$h(nodeID_d : location\_data_d : hello\_time_d)$$
$$\oplus \{h(nodeID_d) : share\_type : nodeID_s : nodeID_d : share\}$$

Table 2. The description of variables

$NodeID_d$	A node ID of a destination node (sent by HELLO)
$Location\_data_d$	A location information of a destination node (extracted from one of HELLOs sent by destination)
$Hello\_time_d$	A time stamp when HELLO from destination node is generated (extracted from the same HELLO as that of the selected $Location\_data_d$)
$Share\_type$	A type of a share (0:source → members, 1:members → destination)
$NodeID_s$	A node ID of a source node
$NodeID_d$	A node ID of a destination node
$Share$	Concatenation of share itself and the value of k and n.

3.4.2 Restoration the secret information

If more than *k* number of share data packets arrives to destination node, the destination node can restore the secret information by using *k* shares.

3.5 Effects of obfuscation

A member node can restore a share by use of his node ID, location data and hello time in the algorithm shown in section 3-2-2. Suppose the node has m buffer to store m combinations of location data and hello time. Then the member node needs *m/2* times computations in average (half size of the buffer). However, for non-member that tries to sniff and restore members' shares, he needs more computation. First of all, he must assume a member node ID and search his memory storing location data and hello time of the member node. In this case, he needs *m/2* calculations in average same as member nodes. He must try *N/2* times in average for different node IDs. Moreover, he must get at least *k* shares. Hence, the non-member node needs *(N/2)*(m/2)*k* calculations.

$$C_c = \frac{k \cdot \frac{N}{2} \cdot \frac{m}{2}}{\frac{m}{2}} = k \cdot \frac{N}{2}$$

where, C_c=complexity of computation of our proposal,
N = total number of nodes

4 Evaluation

Here, we evaluate our proposal and show results. The simulation conditions are shown in Table 3.

Table 3. Simulation condition

Total number of nodes	100
Network size	500×500 [m]
Radio coverage	250 [m]
Simulated time	60 [sec]
Moving velocity	1~10 [m/s]
Hash algorithm	SHA1
Buffer size	50~200
HELLO TTL	1~2
Data Packet TTL	7
(k,n)	(3,5) or (4,5)

4.1 Reachability of secure communication

Figure 5 and 6 shows the Reachability of secure communication when values of k are 3 and 4. The vertical axis indicates the percentage of number of secret information that are successfully received by the destination to the number of sent secrets by the source. And horizontal axis indicates buffer size of nodes. Except the region where the memory size is small, our approach achieves rather high reachability.

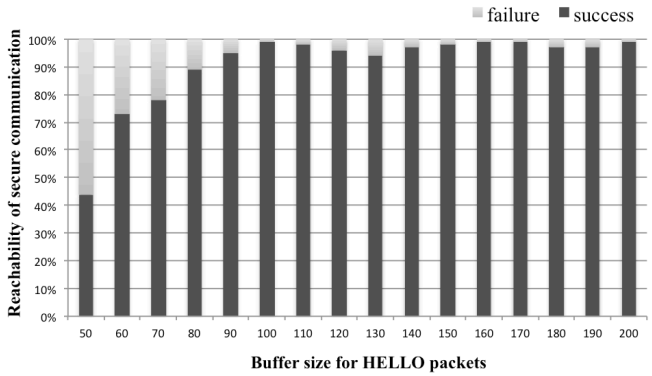


Fig 5. Reachability of secure communication (k = 3)

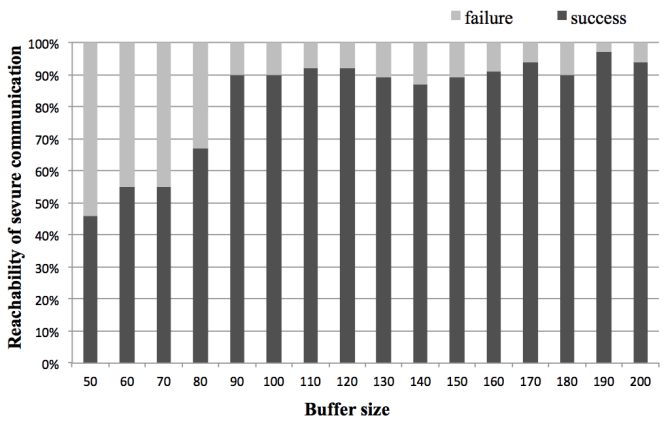


Fig 6. Reachability of secure communication (k =4)

4.2 Number of non-member nodes that are capable of restoring a secret information

Figure 7 and 8 shows the number of non-member nodes that are capable of restoring a secret information by trying brute force attack. The vertical axis indicates number of nodes that succeeds restoring and horizontal axis indicates buffer size of nodes. The results show that it is very difficult to sniff and restore the secret.

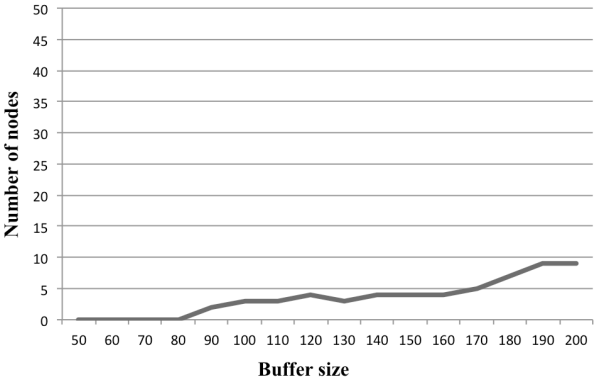


Fig 7. Number of non-member nodes that are capable of restoring a secret information (k = 3)

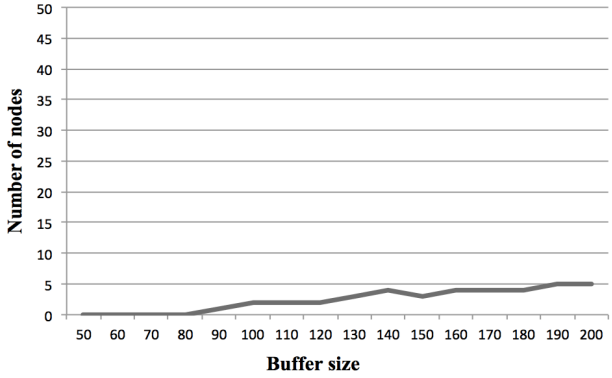


Fig 8. Number of non-member nodes that are capable of restoring a secret information (k = 4)

4.3 Differences of computational complexity between legitimate nodes and non-member nodes

Figure 9 shows differences of computational complexity between legitimate nodes and non-member nodes. For example, in case that number of nodes are 100, our proposal achieves inappropriate nodes needs 200 ~ 500 times of computational complexity compared with legitimate nodes.

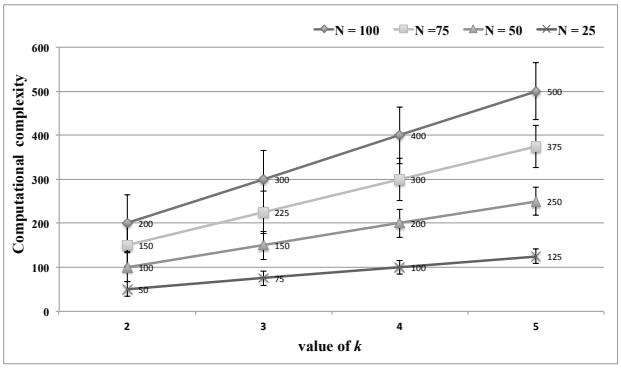


Fig 9. Differences of computational complexity between legitimate nodes and inappropriate nodes

5 Conclusions

This paper has proposed a secure communication method that cannot easily be subjected to network sniffing even in the aspect of security. And we evaluate the proposed method and show its effectiveness. Through simulation evaluation, our proposed method maintains high reachability and high difficulty of sniffing by non-member nodes. Further study is needed to realize more improvement in reachability of secure communication and to clarify how to decrease the number of non-member nodes that are capable of restoring secret information.

6 Acknowledgment

This research is partly conducted as JSPS Grants-in-Aid for Scientific Research 26430372.

7 References

[1] Ken'ichi Mase, Shiro Sakata, "AdHoc Networks and Mesh Networks", corona publishing co., LTD, vol. 1 2007

[2] fa Priyanka Goyal, Vinti Parmar and Rahul Rishi, n legitimate nodes and inappropriate nodescause it is using a wireless communication and also putational Engineering & Management, Vol. 11, January (2011)

[3] Shamir, A., "How to share a secret", Comm. Assoc, Comput, Mach., vol22, no.11, pp612-613, Nov, 1979

[4] Amos Beimel, "Secert-Sharing Schemes: A Survey" IWCC (2011)

[5] D. Eastlake 3rd, P. Jones "US Secure Hash Algorithm 1 (SHA1)" http://tools.ietf.org/rfc/rfc3174.txt, (September 2001)

Efficient Location-aided Route Discovery Mechanism for Ad-hoc networks

P. Phoummavong[1], K. Utsu[2], H. Nishikawa[3], and H. Ishii[2]
[1] Graduate School of Science and Technology, Tokai University, Takanawa, Japan
[2] School of Information and Telecommunication Engineering, Tokai University, Takanawa, Japan
[3] Graduate School of Systems and Information Engineering, University of Tsukuba, Ibaraki, Japan

Abstract - *The location-based routing protocols of Mobile Ad hoc Networks (MANETs) are very important and useful in terms of energy saving and prolonging network lifetime, especially in emergency situations without an IP address. Previous approaches do not work efficiently or well. They may produce a large overhead of packets, which affects their scalability. Moreover, the reachability from source to destination is low in existing methods because route discovery fails and long end-to-end delays may occur due to inefficient route discovery. To solve these problems, we propose route discovery based on location-aided two-hop neighbor information. In this paper, we introduce a pair of approaches to enhance route discovery. The first approach considers the minimum average distance to the destination from two-hop neighbors of the source, and the second approach considers the relation between the source-neighbors' distances from the destination and the number of two-hop neighbor nodes from the source. Furthermore, we compare our approaches with existing algorithms through computer simulation and analysis. The simulation results show that our approaches can achieve higher performance than existing algorithms.*

Keywords: location aided routing; two-hop neighbor information; relay nodes; ad hoc networks

1 Introduction

A mobile ad hoc network (MANET) [1] is deployed by a group of independent mobile nodes without pre-installed infrastructure. Recently, therefore, MANETs have been widely used to support various organizations, including industry, education, military, and medical and emergency services.

Usually MANET nodes use a topology-based routing protocol, which assumes that each node is assigned a unique IP-address. However, considering the deployment of MANET in the face of a recent or ongoing disaster such as an earthquake or tsunami, we see that IP address assignment is difficult and a topology base routing protocol is not suitable. Hence, this paper proposes a location-based routing protocol that can operate in a disaster area without specific IP addresses. Usually, a location-based routing protocol is designed for a mobile ad hoc network by using the global positioning system (GPS), and location services are used instead of IP-addresses. However, in such networks, the power supply of mobile nodes is limited, so routing activities should not consume more than the minimum necessary overhead of packets in order to avoid degrading the ability to send data from source (*S*) to destination (*D*). Moreover, the decisions of the forwarders are the critical mechanism that is used to relay data packets when route discovery is enabled. Many different location-based routing protocols have been proposed. Before discussing our proposed mechanism, let us describe existing approaches and crucial problems.

First, Simple Flooding (SF) [2] is considered as a reference point. SF is a simple system that forwards data packets to all neighbor nodes. This approach guarantees that the data packet will reach the destination (perfect 'reachability'), but it requires a great deal of overhead.

Second is Greedy Perimeter Stateless Routing (GPSR) [3],[4]. This algorithm has two modes 1) greedy forwarding mode (GF), and 2) perimeter forwarding mode. When GF mode fails, the perimeter mode is invoked. However when the network is large, the perimeter mode might give a very bad path.

The next approach is the Geocast Adaptive Mesh Environment for Routing (GAMER) [5]. GAMER uses a hybrid of the greedy and flooding protocols. This approach guarantees the data packets will reach the destination, because it uses the link duration of the feedback at each node to determine the appropriate direction for forwarding. However GAMER creates a redundancy of relay nodes in the forwarding area and a long end-to-end delay. Moreover, GAMER cannot complete the relay of data packets to *D* when a forwarder does not have neighbor nodes located in the same direction as *D*.

Finally, let us consider the Location-Aided Routing protocol (ILAR) [6]. ILAR uses a baseline that is a straight line between *S* and *D* for route discovery. When a route request packet is broadcast, the forwarder that is located in a request zone based on the baseline is chosen as the next forwarder. However ILAR has the same problem as GAMER.

If a forwarder lacks neighbor nodes in the requested zone, no route is found.

The problems of existing location-based routing protocols (SF, GPSR, GAMER, ILAR) can be summed up as: redundant packet transmission, low reachability, and long end-to-end delay. To solve these problems, we propose two algorithms to improve the choice of forwarders in route discovery based on location-aided two-hop neighbor information. The first algorithm considers the minimum average distances between two-hop neighbors of *S* and *D* (CMAD). The second algorithm considers the relation between the distances of neighbors of the source to *D* and the number of neighbor nodes of neighbors (two hops from *S*) (CRDN). By improving the choice of forwarder, our proposals can reduce the amount of overhead, increase reachability, discover optimal path routes and reduce end-to-end delay.

The rest of this paper is organized as follows. Section 2 proposes route discovery based on location-aided two-hop information and our algorithms. Section 3 presents the performance evaluation and results. Section 4 concludes this paper.

2 Location-Aided Route Discovery based on Tow-hop Neighbor Information and Proposed Algorithms

2.1 Sharing the information on two-hop neighbors

In this section, we introduce a route discovery based on location-aided two-hop information. The Geocast algorithm, which uses two-hop information [7], has been proposed for static wireless sensor networks. The proposal shows how to optimize the number of forwarders by using two-hop information in forwarding the data packet to the multicast region. However, this algorithm is designed for a wireless sensor network, and the source node selects the forwarder with the largest number of neighbors and the greatest distance from *S*. To enhance this algorithm, we improved the method for choosing the forwarding node by using the relational information from each node. To set the scene, we assume that a node is deployed in free space without loss of communication. Then all the mobile nodes exchange *hello* messages by broadcasting to their one-hop neighbors. The initial *hello* message contains the location of the node. After receiving the first *hello*s, every node knows the identities, locations and density of its one-hop neighbors.

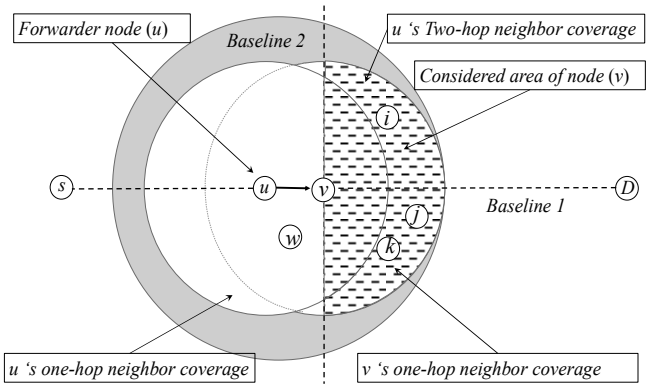


Fig. 1. Forwarder *u* considers one-hop information of *v*

Next, every node broadcasts the second *hello* message, which contains that information on its one-hop neighbors. After that, every node knows the identity and density of its two-hop neighbors (see Fig.1). Then all the nodes operate in the normal mode, in which each node periodically broadcasts the *hello* message to its one-hop neighbors. In the next step, we assume communication between node *S* and *D*, focusing on forwarder *u*, which needs to relay a packet to node *D*. Before forwarder *u* selects the next relay node, forwarder *u* must make a selection by computing the progress of nodes *v* and *w*. Let us consider node *v,* for which forwarder node *u* has the information about one-hop neighbors of node *v* (nodes *i, j* and *k*) that are located in the consideration area (to the right-hand side of the vertical line at node *v* or in the right half of *v*'s radio region). The consideration is made for the right hand side of the vertical line (baseline 2) crossing baseline 1 at node *v* in Fig. 1 (same direction to node *D*). Similarly, node *w* also is taken into consideration by forwarder *u*.

Let's define the notations and parameters used are as follows:

- $n(u)$: the number of one-hop neighbors of u.
- v_x ($x \in \{1, 2, 3,\ldots, n(u)\}$): a one-hop neighbor of sender node u
- $v_{x,y}$ ($y \in \{1, 2, 3,\ldots, n(v_x)\}$):a one-hop neighbor of v_x (two-hop neighbor of node S).
- $n(v_x)$: the number of a one-hop neighbor of v_x .
- $d(v_x, D)$: the distance from v_x to D.
- $d(v_{x,y}, D)$: the distance from $v_{x,y}$ to D.

A baseline 1 is set between v_x and the destination D. Then a base line 2 is drawn at 90 degree angles to baseline 1 at v_x. The two-hop neighbor nodes of v_x to be considered as next relay nodes are located in the right hand side of baseline 2 in Fig.2 and Fig.3.

2.2 CMAD

The first approach we propose is optimizing the decision of a forwarder in route discovery based on two-hop neighbor information based on the minimum average distances to *D* from two-hop neighbors of *u*.

We can calculate the average distance from two-hop neighbors of *u* to *D*. To calculate the average distance from two-hop neighbors of *S*, we determine the average distance to *D* from two-hop neighbors of node *u* by using (1).

$$d_{avg}(v_x) = \frac{1}{n(v_{xy})} \sum_{y=1}^{n(v_{xy})} d(v_{x,y}, D) \quad (1)$$

In (1), where $d_{avg}(v_x)$ represents the average distance to *D* as shown in Fig. 2. Thus node v_x does not include node *u* when calculating the average distance, because node *u* is outside the considered area. Node *u* chooses the neighbor node that has the minimum average distance (2) as the next sender and sends a RREQ (Route REQuest).

$$Select = \min_{\forall i} \{d_{avg}(v_x)\} \quad (2)$$

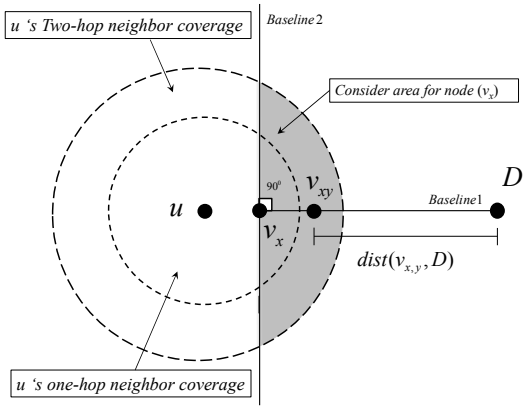


Fig. 2. CMAD calculates the average distance of two-hop neighbors of *u* to *D*

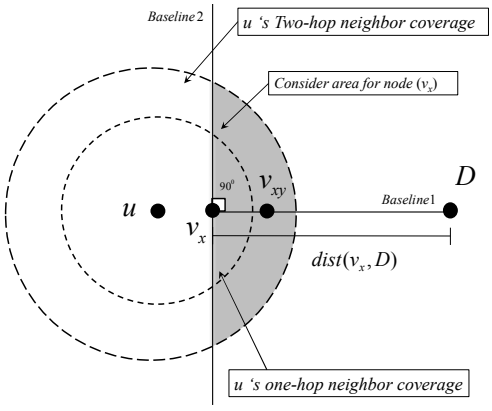


Fig. 3. CRDN: forwarder *u* considers one-hop information of *v*

2.3 CRDN

The second proposal, we consider the probability index or the relation between the distance to *D* from neighbors of *u* and the number of neighbor nodes of neighbors of *u*. Just as in the first method (CMAD), a sender collects two-hop neighbor information, but this differs in the fact that *u* uses the distances from one-hop neighbors (v_x) to *D* and their number of neighbors as shown in Fig. 3.

We find the probability (distance between node v_x and *D*) divided by the sum of $d(v_x,D)$ for all x (x=1,2,... $n(u)$) (total of distances from neighbor nodes of *u* to *D*), Let d_t be the sum of $d(v_x,D)$ for all $x \in \{1, 2, 3,\ldots, n(u)\}$. Whichever neighbor node of *u* is closest to *D* will get the lowest probability as (4).

$$d_t = \sum_{x=1}^{n(u)} d(v_x, D) \quad (3)$$

$$p_{avg}(v_x) = \frac{d(v_x, D)}{d_t} \quad (4)$$

As the next step, we find the probability by considering the number of neighbors of each neighbor node of v_x. Let n_t be the summation of the $n(v_x)$ for all the *x* as (5). The probability index $p_n(v_x)$ is calculated using (6), which considers the number of neighbors of each neighbor node of v_x. The more $p_n(v_x)$ is, the more likely v_x is selected.

$$n_t = \sum_{x=1}^{n(u)} n(v_x) \quad (5)$$

$$p_n(v_x) = \frac{n(v_x)}{n_t} \quad (6)$$

So if a neighbor of *u* (e.g., v_x) has the lowest number of neighbors, it will have the highest probability. Finally, from (4) and (6), we get a probability that is a combination of the probabilities of distance and the probability regarding the neighbor node v_x.

Definition (Probability Independent Events): $p_{avg}(v_x)$ and $p_n(v_x)$ are considered mutually independent since the number of neighbors and their distances are independent of each other. Hence, we can get (7).

$$Pr(v_x) = (1 - p_{avg}(v_x)) p_n(v_x) \quad (7)$$

Subsequently, node *u* can select the next neighbor node if the node has the highest probability $Pr(v_x)$.

3 Performance and Evaluation

Here, we analyze the performance of our proposals (CMAD, CRDN) by using a simulation to compare them with existing algorithms GAMER and ILAR in the static mesh network node. We assume all the nodes are static (no mobility), and each node is uniformly distributed in a rectangular area. The overhead (measured in packets), the packet delivery ratio, the average number of hops and the end-to-end delay are used as the performance metrics.

3.1 Assumption and network topology

In the static network topology, we show the simulation parameters in Table1. For each topology we assume a position for *S* is randomly selected 100 times for each topology of each number of nodes.

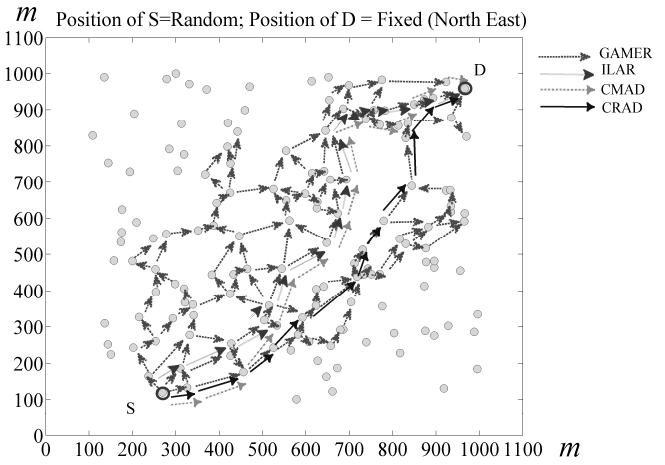


Fig. 4. An example of network topology with *S* selected randomly

Table 1. Simulation Paramiter

Simulation Parameters	Value
Topology size	1000 m x 1000 m
Number of nodes	30, 50, 100 and 150
Transmission range	250 m
Mac protocol	IEEE 802.11b
Packet size	512 bytes
Time intervals broadcast messages	0.1s
Pause time	2s
Ad hoc routing protocol	GAMER, ILAR, CMAD, CRDN
Simulation time	900 s
Simulation trials	1000 Topologies for each density

3.2 Evaluation and results

In this section, we show the performance results from the simulation. Fig.4 shows an example route determined by each algorithm and our proposal.

The first result shows the number of overhead packets. The second result shows the average number of hops of relay nodes. Then, we show the packet delivery ratio, i.e. the ratio of arrived packets to all sent packets, and finally the average end-to-end delay.

3.2.1 The overhead packets:

When counting the overhead packets, we included all combinations of packets, the nature of which depends on the algorithm in use, i.e. *flooding route discovery*, *route request*, *route response*, *two-hop beacon exchange* message, *hello* message, *location information query* and *location information response*.

Fig. 5 shows the overheads of GAMER, ILAR and our approach CMAD and CRDN with different number of nodes. GAMER tries to minimize the number of transmitted nodes by determining the flooding area. However, the number of overheads in GAMER is greater than in ILAR and our algorithms CMAD and CRDN. The next approach is ILAR, which is better than GAMER because it determines the expected zone of *D* before it relays packets. The expected zone is advantageous when used to find *D*. However, the overhead of ILAR is also large because it sends many overhead packets into the expected zone. In contrast, in CMAD and CRDN, the overhead of packets increases very slowly and they are smaller than those used in the two previous algorithms. Computing only average distances, CMAD is not well suited to choose a relay node. CDRN is better than CMAD because it uses probability to select relay nodes.

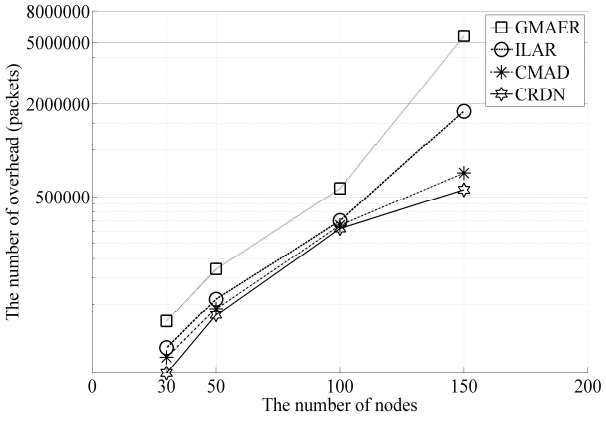


Fig. 5. Number of overhead packets vs. number of nodes

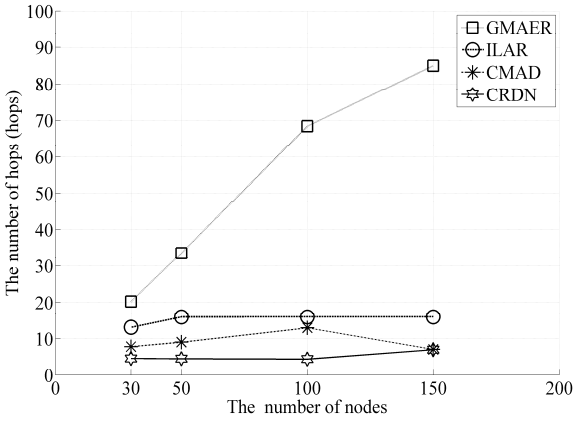


Fig. 6. Number of hops vs. number of node

3.2.2 The number of Hops:

Fig. 6 shows how the number of hops of relay nodes varies with the number of nodes. The path length with GAMER is larger than with ILAR because each node relays packets throughout a forwarding area (directional flooding). The next approach is ILAR, which optimizes the average path length better than ALARM, but the average path length is increased in the multicast region or expected zone. CRDN gives shorter path lengths than the two previous algorithms.

3.2.3 The packet delivery ratio:

The next result is the packet delivery ratio versus the number of nodes. To calculate the packet delivery ratio [8], we use (8).

$$p_{rx} = \frac{N_{rx}}{N_{tx}} = \frac{N_{tx} - N_{drop}}{N_{tx}} \tag{8}$$

Where p_{rx} represents the packet delivery ratio; N_{rx} is the number of packets successfully delivered to *D,* N_{tx} is the number of packets sent from *S*, and N_{drop} represents the number of packets dropped.

Fig. 7 plots the packet delivery ratio versus the number of nodes when *S* is chosen randomly. GAMER, ILAR, CMAD and CRDN all have very high ratios, that are 98%-100%, which means the packets are successfully received by *D* for all the cases of node numbers. CMAD scores lower than CRDN because the forwarder cannot relay the packet when no neighbors are located in same direction as the destination. However, Fig. 7 shows that GAMER does better in all number of nodes but GMAER consumes much overhead packets in previous performance.

3.2.4 The end-to-end delay:

Next we consider the average end-to-end delay of data packets [9], which we can compute using (9).

$$T_{delay} = \frac{\sum_{i=1}^{N} (t_{r(i)} - t_{s(i)})}{N_{rx}} \tag{9}$$

Where *N* is the total number of packets delivered to *D*, $t_{r(i)}$ represents the time when a packet arrives at *D*, and $t_{s(i)}$ represents the time when the packet was sent by *S.* We assume that the delay includes all the delays in route discovery, such as queuing. Fig. 8 shows the end-to-end delay. GAMER has high delay because the algorithm always needs to perform route discovery before sending a data packet. ILAR is better than GAMER because it encounters route discovery delay only at the first stage. However CMAD and CRDN are faster than ILAR because they dos not need to determine the expectation zone to find *D*, and CRDN is the fastest.

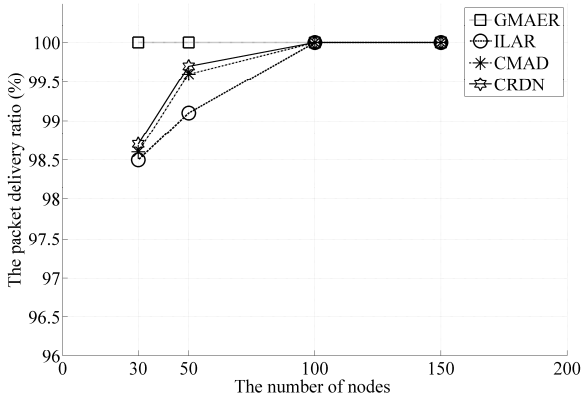


Fig. 7. Packet delivery ratio vs. number of nodes

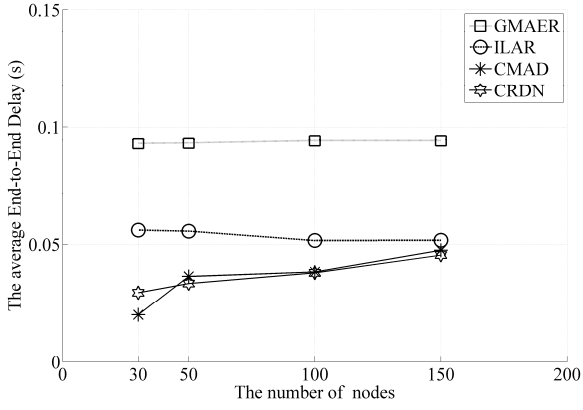


Fig. 8. Average end-to-end delay vs. bumber of nodes

4 Conclusions

In this paper, we address the problem of efficient data transmission in a mobile ad hoc network. Selecting the best next node among neighbor nodes is very difficult in existing location-aided routing algorithms, requiring an increase in reachability, and reductions in overhead, delay, and number of hops. To overcome these problems, we proposed location-aided route discovery algorithms (CMAD and CRDN) based on two-hop neighbor information about an ad hoc network inspired by geographic routing. Simulation results show how our proposals select the optimum relay nodes and improve several aspects of performance with different number of nodes and network topologies. Specifically, CRDN has the best performance in terms of low overhead, high delivery ratio, small number of hops and low latency, compared with existing methods. Future studies will perform more detailed evaluations, including mobility and power consumption.

5 Acknowledgment

This work is partly supported by JSPS KAKENHI Grant Number 26420372.

6 References

[1] S. Misra, I. Woungang and S. C. Misra, "Guide to Wireless ad Hoc Networks", *British Library Cataloguing in Publication Data, Springer-Verlag* London Limited, 2009.

[2] S. Y. Ni, Y. C. Tseng, Y. S. Chen and J. P. Sheu, "The Broadcast Strom in a Mobile Ad Hoc Network," *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, ACM New York, USA, pp. 151–162, 1999.

[3] B. Krap and H.T. Kung, "GPSR: Greedy perimeter stateless routing for wireless network," *in MobiCom '00*, New York, USA: ACM, 2000, pp. 243-254.

[4] C. Maihofer, "A Survey of Geocast Routing Protocols," IEEE Communication Surveys and Tutorial, Vol. 6, Issue. 2, pp. 32-42, 2004.

[5] T. Camp and Y. Liu, "An adaptive mesh-based protocol of geocast routing", Journal of Parallel and Distributed computing, pp. 196-213, October 2002.

[6] G. C. Wang and S. M. Wang "An Efficient Location-Aided Routing Protocol for Mobile Ad Hoc Networks," *Parallel and Distributed Systems, Proceedings. 11th International Conference*, Vol. 1, pp. 335-341, July. 2005

[7] Y. C. Shim and H. S. Kang, "An Efficient Geocast Algorithm using 2-hop Neigbour Knowledge in Sensor Networks", *International journal of latest trends in computing,*Vol.2, 2011.

[8] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks", *in Proc. ACM SenSys'03*, Los Angeles, CA, USA, Nov. 5–7, 2003.

[9] R. Groenevelt, P. Nain and G. Koole," The message delay in mobile ad hoc networks" *Elsevier's Performance Evolution, Science Direct*, August, 2005.

Highly-Dependable and Long-Lifetime Data-Driven Networking Processor with Energy Assurance Capability

Shuji SANNOMIYA and Hiroaki NISHIKAWA
Graduate School of Systems and Information Engineering, University of Tsukuba,
Tsukuba Science City, Ibaraki, Japan

Abstract— *In addition to thrifty power consumption, failure detection is one of the crucial issues to exploit large wireless network systems realized by battery-operated wireless devices while keeping system maintenance cost acceptable. CUE, a data-driven networking processor, exploits passive and on-demand (data-driven) processing manner exhaustively to its circuit implementation level and thus its power consumption is reduced as low as essentially required. This essential power consumption also results in strong correlation between consumption current and processing load of the CUE. By virtue of this strong correlation, failures changing consumption current can be detected by comparing run-time consumption current with a current range estimated with a given typical processing load beforehand. In this paper, a parallelized pipeline with shortcut to realize load distribution is proposed to smooth the run-time processing load in order to assure the estimation on the current range, and its effectiveness is proven by circuit simulation.*

Keywords: data-driven processor, real-time multiprocessing, self-timed pipeline, sensor network, machine-to-machine, internet of things

1. Introduction

Wireless network systems composed by battery-operated wireless devices can be easily installed into existing constructions and infrastructures and thus they are one of promising technologies to realize convenience and safety for both human life and social infrastructure, compared with wired systems. To exploit such systems largely, not only long-lifetime but also high dependability are indispensable because the power budget of battery is strictly limited and doubtful devices make system maintenance cost unacceptable for large scale systems.

In order to realize highly dependable and long-lifetime devices, the authors have studied a series of data-driven networking processor, named CUE [1], which can provide the wireless devices with not only ultra-low-power feature but also unique observability resulting in autonomous detection of the doubtful devices. The CUE is an embodiment of data-driven principle by which operation execution is initiated on the arrival of input data as long as computational resources (i.e. pipeline stages) are available. This passive operation execution makes it possible to realize real-time multiprocessing indispensable for networking under time-constraints defined by communication protocols without extrinsic program-execution overheads such as context switching and interrupt handling resulting in power dissipation.

The data-driven principle is also realized in the circuit implementation of the CUE by using self-timed elastic pipeline in which each pipeline stage autonomously transfers valid data based on local negotiation between adjacent pipeline stages. As a result of this local data transfer, pipeline stages with valid data are exclusively driven in the CUE, and thus switching power concentrates into the processing of valid data naturally while the other empty pipeline stages are powered off to reduce leakage power [2]. Moreover, the supply-voltage of the self-timed pipeline without any global-clock can be scaled anytime without suspends, and thus the power consumption can be reduced as low as essentially required by using run-time voltage scaling technique [2]. The ultra-low-power feature of the CUE has already been demonstrated in our previous studies [3].

In addition to the ultra-low-power consumption, the essential power consumption of the CUE provides direct observability on the processing load through consumption current because the power in the CUE is consumed only to processing data and thus the consumption current has strong correlation with the number of processing data (i.e. processing load). By virtue of this strong correlation, consumption current in operation is foresaw under a given typical processing load, and thus the energy which dominates the battery-operated lifetime of system devices can be estimated before the installation of the devices. This foreseeability of the lifetime leads to the drastic reduction of the system maintenance costs. For instance, the schedule of the replacement of battery can be designed and optimized before the system installation, and thus the working for battery replacement each time the battery runs down becomes no longer required.

Moreover, the foreseeable consumption current makes it possible to detect the failures of devices autonomously after the system installation. This is because unexpected failures resulting in the change of the consumption current can be detected by measuring the consumption current and comparing the measured result with the amount of consumption current estimated before operation. For instance, the malfunction of the peripheral units such as sensors changes the processing load of the CUE and this change can be detected via the

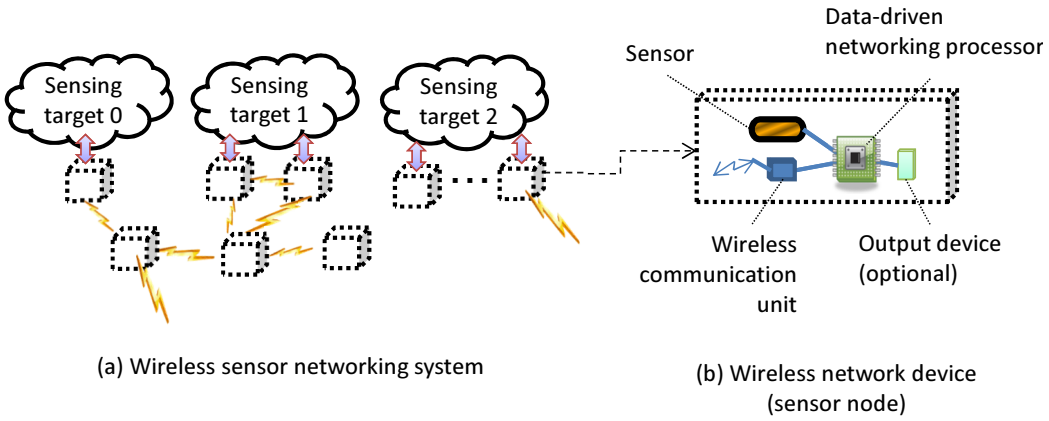


Fig. 1: An example: data-driven wireless sensor networking sytem.

consumption current of the CUE, and a wireless network device which transfers unusual data changes the processing load of the CUE in its circumjacent devices, and this change can be also detected via consumption current. This autonomous detection of the failures becomes indispensable in the future wireless network systems because it becomes especially difficult to find out a device with failure in large scale systems especially in the era of Trillion Sensor Universe advocated by Dr. Janusz Bryzek.

To assure the estimation of the consumption current in operation, unpredictable changes of the consumption current should be minimized. In this paper, a pipeline structure implementing the CUE is designed to reduce the dynamic change of the consumption current. The number of operations executed concurrently may change according to the data arrival timing even when processing the same number of data. In the CUE, the supply-voltage is regulated according to the number of operations executed concurrently to realize the ultra-low-power consumption. On the other hand, the regulation of the supply-voltage may change the consumption current significantly because the consumption current is in proportion to the square of voltage. Unfortunately, it is difficult to predict the arrival timing of all data precisely. Under this practical constraint, to realize both the ultra-low-power consumption and faithful estimation of the consumption current in operation, a parallelized pipeline with shortcut is proposed to realize load distribution at pipeline stage level for smoothing the number of operations executed concurrently. The effectiveness of the proposed pipeline is evaluated through a concrete protocol processing.

2. Data-driven networking processor

In this section, the necessity of both ultra-low-power consumption and failure detection is discussed with an explanation of data-driven wireless sensor networking system which is a wireless network system. Moreover, requirements on data-driven networking processor are discussed.

2.1 Data-driven wireless sensor networking system

Wireless sensor networking systems provide a variety of services such as security, infrastructure monitoring, and disaster prevention, by networking sensors spread on sensing targets as shown in figure 1(a). Although the wireless sensor networking systems can be easily installed because of the absence of wiring, system maintenance cost to keep the wireless devices alive and healthy may increase in large scaled systems due to the replacement or charge of discharged batteries and checking of the operation of every device. Therefore the key to realize large scale wireless networking systems is to reduce the frequency of the battery replacement/charge and the work for operation checking as low as possible.

As illustrated in figure 1(b), data-driven wireless sensor networking system realizes ultra-low-power consumption to save the batteries drastically and failure detection to eliminate the need for operation checking, by introducing ULP-DDCMP (Ultra-Low-Power Data-Driven Chip Multiprocessor) [4]. As shown in figure 2, the ULP-DDCMP is a chip multiprocessor version of the CUE, and it is realized by interconnecting a ultra-low-power version of the CUE, named ULP-CUE by using a token router which is a switch based multi-stage interconnection network whose switch is realized by merge (M) stages which accept data from two preceding stages in order of arrival and transfers the accepted data to a succeeding stage and branch (B) stages which transfer each data to one of two succeeding stages selectively.

Figure 3 shows the ULP-CUE's circular pipeline which is indispensable to naturally realize the iteration of operation execution in which operation result is transferred to the input of the succeeding operation. The circular pipeline of the ULP-CUE consists of matching memory (MM) to detect the arrival of operands, program storage (PS) to fetch operations, functional processing unit (FP) to execute the operations

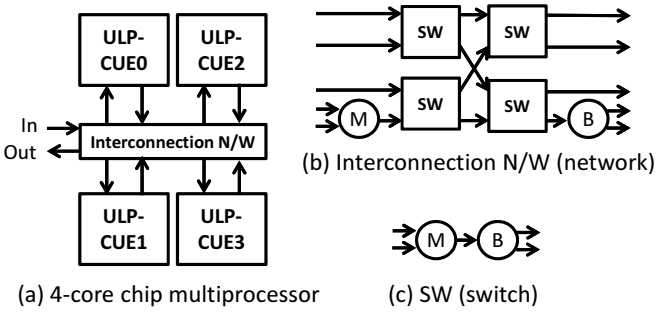


Fig. 2: Data-driven networking chip multiprocessor (an example of 4-core).

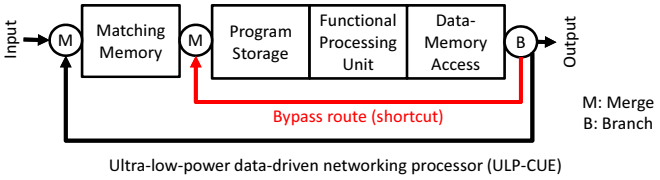


Fig. 3: Circular pipeline to realize ULP-CUE.

and memory access (MA) to read and write data. With this structure, the concurrent operations of target programs can be naturally exploited over the circular pipeline as long as the pipeline stage is available, as a result of data-driven program execution, i.e. no context switching and scheduling are required. Moreover, the circular pipeline is optimized for protocol handling whose operations are mainly unary, by providing shortcut to bypass the MM when unary operations are executed [4].

To realize the passive and on-demand processing to the circuit implementation, the whole stages are realized by using ultra-low-power self-timed elastic pipeline (ULP-STP). In the ULP-STP, only pipeline stages with valid data are driven exclusively as a consequence of the localized data transfer called handshake. Figure 4 shows the structure of the ULP-STP in which each stage consists of a data-latch (DL), functional logic (FL) and transfer control unit (C). The ULP-STP is a kind of asynchronous bundled data pipelines, and it employs four-phased handshake [5]. Based on the four-phased handshake, the valid data in the STP are transferred between adjacent stages, as follows.

- Reset: After the assertion of the reset signal, the C negates both its send signal representing transfer request and ack signal representing acknowledge.
- The C asserts its ack signal after its send signal is asserted.
- After the assertion of the ack signal, the preceding C negates its send signal.
- After the negation of the send signal, the C asserts both its gate open signal (cp) and its send signal and it negates concurrently its ack signal, only if the ack signal is negated. As a result, the data is latched in the stage to which the C belongs.
- The succeeding C repeats the above steps similarly to the C.

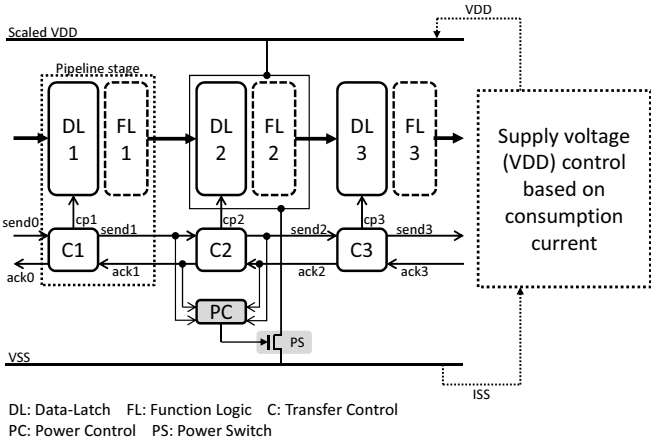


Fig. 4: Self-timed elastic pipeline with run-time voltage scaling and power gating.

This handshake concentrates dynamic consumption current into the pipeline stages with valid data naturally while the power control and power switch power off the empty pipeline stages to reduce the leakage current through the empty stages. Moreover, the signal propagation delay of the DL, FL and C are changed at equal rate according to the supply-voltage, and thus the supply-voltage of the ULP-STP can be scaled at run-time while the rate of change of the voltage is moderate enough to guarantee the transistor switching, i.e., the throughput and processing time of the ULP-DDCMP can be changed during the execution of target programs.

The ultra-low-power consumption of the ULP-DDCMP realized by the ULP-STP has already been demonstrated [3].

2.2 Requirement for energy assurance

Figure 5 shows the measured values of the throughput and current consumption of the ULP-CUE in a prototype of the ULP-DDCMP. As a proof of the essential power consumption of the ULP-DDCMP, both the throughput and current consumption increase when the occupancy rate of the ULP-STP increases, i.e. they increase when the amount of processing load increases. Moreover, the consumption current has strong correlation with the processing load. This strong correlation makes it possible to estimate consumption current in operation ($I[A]$) according to the processing load in operation beforehand by using a system level simulator [6] and to calculate battery-operated lifetime ($T[sec.] = W/I$) of the wireless networking devices by using the I and a given ampere-hour capacity ($W[Ah]$) before the start of the operation of the devices. This predictability on the lifetime leads to the economical design of the battery replacement/charge schedules.

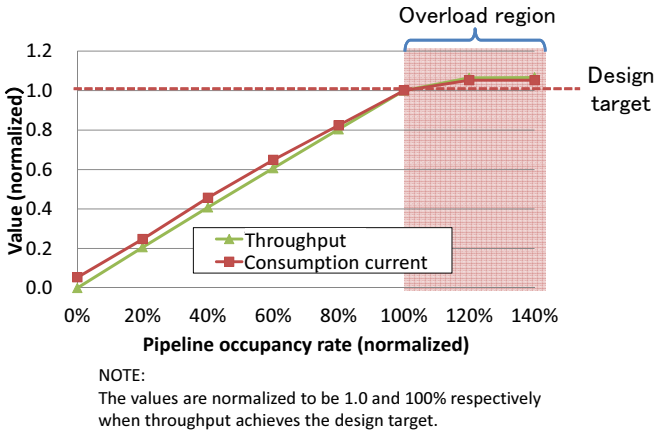


Fig. 5: Direct correlation between throughput and consumption current.

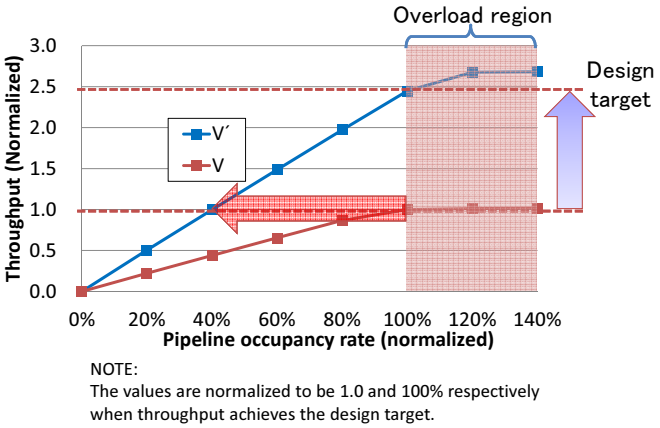


Fig. 6: Run-time scaling of throughput design target.

On the other hand, the processing load may change in operation due to failures such as malfunction of the sensors and also the consumption current may change due to the change of the processing load as well as operating environment changes such as the change of temperature. Therefore, the actual battery-operated lifetime may differ from the calculation. Fortunately, by virtue of the strong correlation, such failures and contingencies in the data-driven wireless networking system can be detected by measuring the consumption current in operation. In particular, ampere-hour capacity during a certain time $t[sec.]$, which is denoted by $W_t[Ah]$, can be calculated as explained above, and discharge capacity ($d[Ah]$) which is the production of the measured run-time consumption current within the t and t is compared to the W_t. The deviance of the d from the the range of the W_t means that the failures or contingencies occur. In operation, devices whose d becomes out of the W_t inform the failures/contingencies occurrence to the other devices or the center of management with a request to settle the failures/contingencies. This autonomous detection

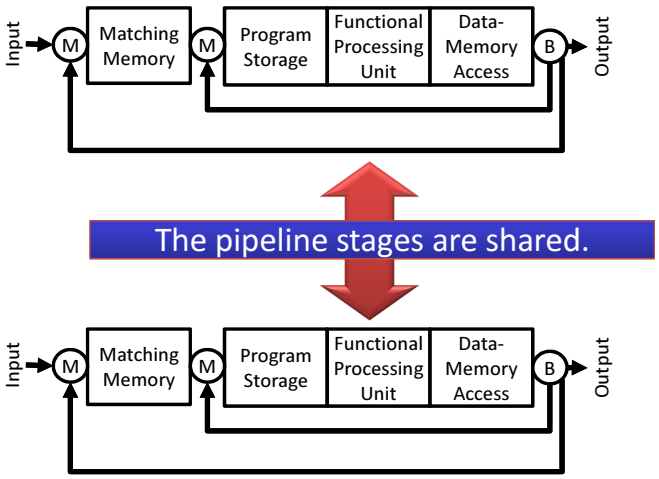


Fig. 7: Load distribution between two parallel pipelines.

of the failures leads to the elimination of the health checking operations.

In addition to these energy assurance features, the ULP-DDCMP provides a natural tolerance against the processing load fluctuation. As shown in the figure 5, as a result of the handshake, the throughput is kept at a maximum value without any additional controls when the pipeline occupancy rate exceeds the design target and reaches overload region temporarily. However, the processing time increases when the pipeline occupancy rate is in the overload region because data transfer at some pipeline stages is postponed until the succeeding stages completes the data processing and becomes empty. To avoid this processing time increase, as shown in figure 6, the supply-voltage is controlled according to the pipeline occupancy rate observed through the consumption current and it is increased to speed-up the ULP-DDCMP before the observed pipeline occupancy rate falls into the overload region.

This run-time supply-voltage control may change the consumption current widely because the consumption current is in proportion to the square of the supply-voltage. In order to preserve the consumption current in operation within the estimated range, dynamic processing load fluctuation should be suppressed as possible from the viewpoint of the energy assurance of ULP-DDCMP.

3. Fine-grain load distribution to enhance elasticity

In this section, the pipeline structure of the ULP-DDCMP is redesigned to suppress the dynamic processing load fluctuation. In chip multiprocessor structures, processing cores share the processing load among them, and usually a set of data is the unit of processing load distribution. For instance, a packet is a unit to be assigned to one of the processing cores in protocol handling. On the other hand, the actual

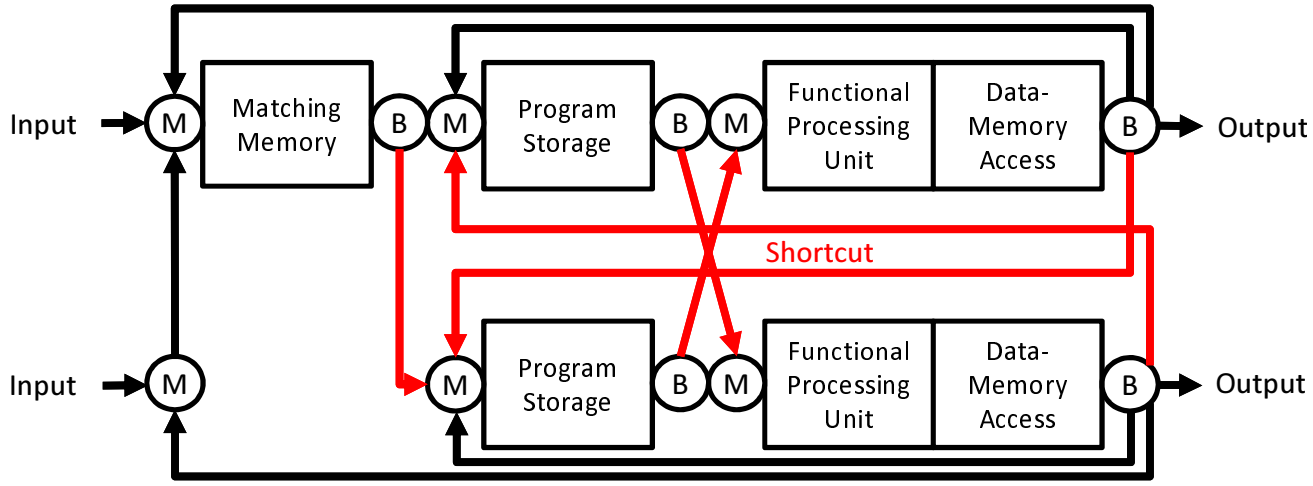


Fig. 8: ULP-CUE-P.

processing load of a unit may differ due to the amount and value of the assigned data. To smooth the processing load fluctuation finely, deeper load distribution should be realized.

3.1 Utilization of ULP-STP's elasticity

Although the packet handling time is in the order of hundreds $\mu sec.$ according to the actual demonstration experiments [4], it is practically difficult to estimate the arrival timing of all packets in the hundreds $\mu sec.$ order, and thus the number of packets processed concurrently may differ from the estimated value. Fortunately, in the ULP-STP, the number of operations executed concurrently is temporally smoothed as a result of the handshake and thus the pipeline occupancy rate may be kept before the overload region when the number of packets processed concurrently changes. However, this elasticity of the ULP-STP is available when the number of packets processed is within a certain value, and therefore the run-time load distribution is necessary to keep the elasticity of every ULP-STP available.

From the viewpoint of the fair sharing of the processing load among ULP-CUE's, the processing load should be finely distributed, i.e. operation level load distribution is preferable instead of conventional packet level load distribution. However such fine grain load distribution results in the increase of both power consumption and processing time due to the intensive usage of the interconnection network among the ULP-CUE's because every data transferred among the ULP-CUE's goes through the interconnection network shown in the figure 2. In this paper, the pipeline of the ULP-DDCMP is restructured by integrating the operation execution pipeline and the interconnection network to realize operation level load distribution without overheads.

3.2 Parallelized self-timed elastic pipeline with shortcut

The pipeline occupancy rate of each processing core, ULP-CUE, of the ULP-DDCMP is determined by the number of operations executable concurrently, and thus sharing the operations among the ULP-CUE's fairly results in the reduction of the pipeline occupancy rate of each ULP-CUE. That is, the fair sharing of operations suppresses the occurrence of the situation where the pipeline occupancy rate falls into the overload region. To realize such fair sharing of operations, the pipeline structure should be redesigned to avoid the intensive usage of the MM, PS, FP, MA of a specific processing core, as shown in figure 7.

To realize completely fair sharing of operations, all of operations fetched in the PS should be assigned among all of the FP, MA and MM fairly. In this paper, sharing is studied among two ULP-CUE's as a minimum configuration. To make it possible to assign any of operations to both ULP-CUE's, the PS's store the same target program. The operations fetched in the PS's are distributed in round-robin fashion between two FP's to realize fair sharing. To realize this distribution, branch (B) stages whose output is transferred to alternately upper path and lower path and merge (M) stages to merge the outputs from the B's are added to provide shortcut for load distribution between two parallel circular pipelines. As for the MM, the output data is distributed in round-robin fashion as well as the fetched operations.

Figure 8 illustrates the redesigned pipeline named ULP-CUE-P (ULP-CUE with parallelized circular operation execution pipeline). In the ULP-CUE-P, the MA's share one physical memory realized by 2-port RAM to preserve coherency of stored data. Moreover, the MM's are unified as one because binary operations are a minority in protocol handling. For instance, the binary operations occupy only

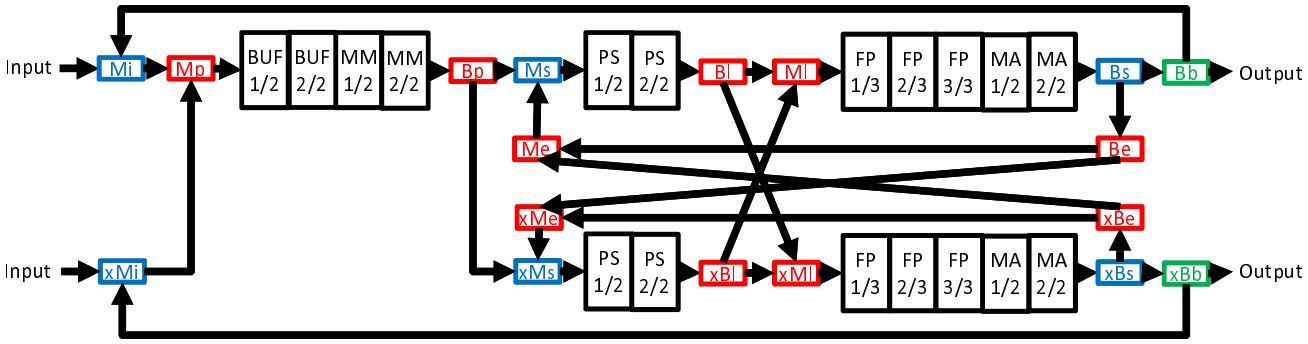


Fig. 9: Designed circuit of ULP-CUE-P.

approximately 20% of UDP/IP handling program. With the proposed pipeline structure, the operations are distributed fairly between two parallel pipelines without transferring the interconnection network between processing cores which are essentially unnecessary to execute target operations.

3.3 Evaluation on load distribution

To evaluate the processing load distribution capability of the proposed ULP-CUE-P, the circuit of the ULP-CUE-P is simulated. The circuit of the ULP-CUE-P is described at RTL (Register Transfer Level) by utilizing the RTL description and pipeline tact information of the ULP-DDCMP. The circuit size of the ULP-CUE-P is comparable to that of the two ULP-CUE's, and thus the simulation result of the ULP-CUE-P is compared with that of the two ULP's.

To realize the proposed ULP-CUE-P, the merge and branch stages described blow are added in addition to the MM, PS, FP, MA and MM.

- Mi: a merge stage to accept input data
- Bs: a branch stage to realize the shortcut
- Ms: a merge stage to realize the shortcut
- Bb: a branch stage for binary operation and output
- Bp: a branch stage for distributing the binary operations
- Be: a branch stage for distributing the unary operations
- Mp: a merge stage for binary operations
- Me: a merge stage for unary operations
- Bl: a branch stage for sharing operations
- Ml: a merge stage for sharing operations

Figure 9 shows the circuit of the ULP-CUE-P. The branch and merge stages of the lower pipeline are labeled with "x" to distinguish those of the upper pipeline. The ULP-CUE-P has only one MM, and thus a buffer stage named BUF is added to compensate one of two MM's existing in the conventional two ULP-CUE's. This is because the reduction of the number of pipeline stages increases the pipeline occupancy rate against the same processing load. To annotate the signal propagation delay in the ULP-DDCMP to the described circuit, the pipeline tact extracted through the circuit simulation of the post-layout circuit of the prototype LSI of the ULP-DDCMP which is shown in figure 10 is set to every pipeline stage of the described circuit.

In the simulation, the conventional ULP-CUE is realized by disabling the branch stages, Bp, Bl and Be for load distribution. To confirm that the processing time increase due to the processing load fluctuation is suppressed by the ULP-CUE-P, the processing time of a protocol handling program is measured through the simulation. As the protocol handling program, a data-driven program of UDP/IP handling is used because its connection-less packet transfer results in low-power consumption indispensable in battery-operated wireless devices and thus it is one of the protocols expected to be used in the wireless ad hoc networking systems. Figure 11 shows the measured results of ULP-CUE-P and two ULP-CUE's. As shown in the result, the processing time increase in the ULP-CUE-P (proposed) is within approximately 4% when three packets are processed simultaneously while approximately 18% processing time increases in the two ULP-CUE's (conventional). This is a proof that the processing load fluctuation is suppressed by the proposed pipeline structure.

4. Conclusions

This paper describes a parallelized self-timed pipeline with shortcut to exploit the energy assurance capability of the data-driven networking processor CUE, in order to preserve the ultra-low-power consumption of the CUE and to assure the estimation of the CUE's energy and consumption current in operation. The proposed pipeline is evaluated through an RTL simulation with UDP/IP handling and it is proven that the processing load fluctuation resulting in the change of the consumption current in operation can be suppressed by the operation level load distribution realized by the proposed circuit.

The proposed circuit is an implementation of the operation level load distribution for two circular pipelines (processing cores), and its scalability for 4 or more pipelines should be further studied because the multiport memory required to preserve the stored data coherency is cannot be scaled without overhead. On the other hand, the consumption

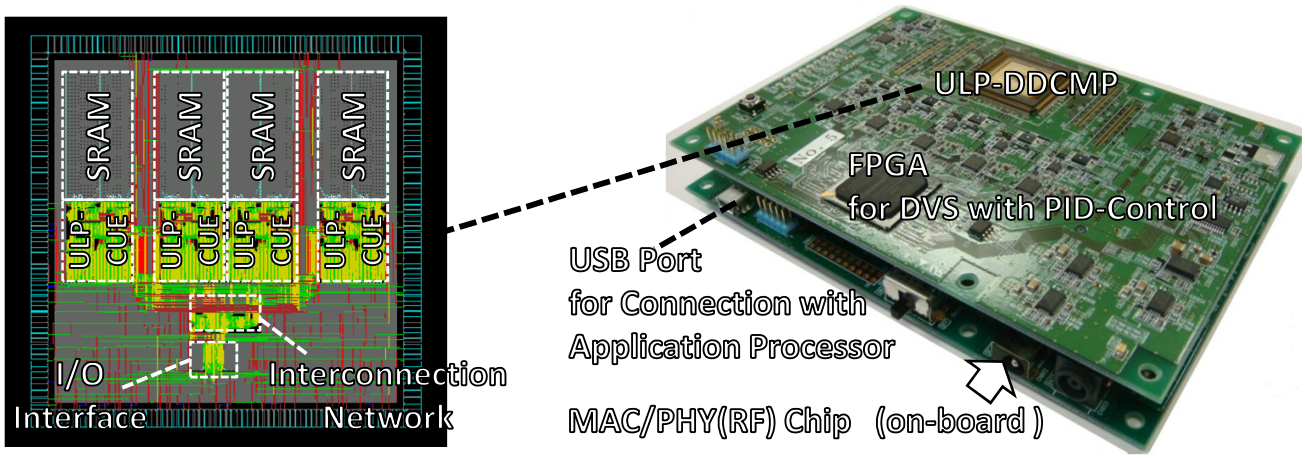


Fig. 10: Prototype of data-driven networking chip multiprocessor.

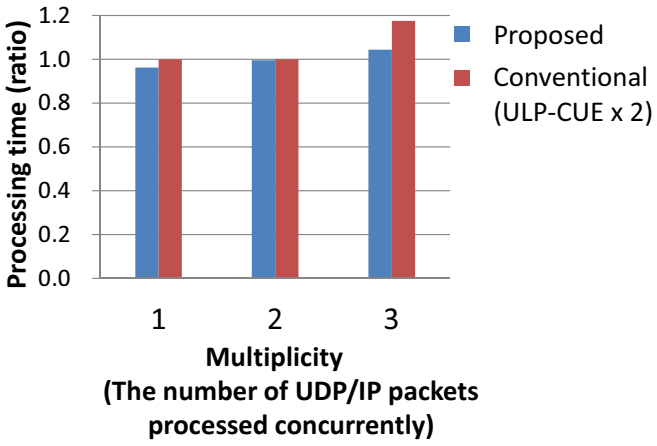


Fig. 11: Processing time of UDP/IP protocol handling.

current on a program can be easily estimated and observed if the program is allocated to a processing cores group different from the other groups which execute the other programs in target application, and the proposed circuit may be sufficient to become one of the processing cores groups, i.e. the proposed circuit can be the unit of the program allocation in chip multiprocessor. To examine this possibility and demonstrate the practical effectiveness of the proposed pipeline architecture, the proposed circuit is now being evaluated through a system level simulation [6] of an actual data-driven wireless sensor networking system. The result of the further evaluation will be presented at the conference.

Acknowledgement

Although it is impossible to give credit individually to all those who organized and supported our project, the authors would like to express their sincere appreciation to all the colleagues in the project.

This research work was supported in part by START program (Program for Creating Start-ups from Advanced Research and Technology) of MEXT (Ministry of Education, Culture, Sports, Science and Technology) and IS program of Semiconductor Technology Academic Research Center (STARC). The CAD tools for the evaluation in this work is supported by VDEC (VLSI Design and Education Center), the University of Tokyo in collaboration with Synopsys, Inc.

References

[1] Hiroaki Nishikawa, "Design Philosophy of a Networking-Oriented Data-Driven Processor: CUE," IEICE Transactions on Electronics, Vol.E89-C No.3, pp.221-229, Mar. 2006.

[2] Kei Miyagi, Shuji Sannomiya, Makoto Iwata, and Hiroaki Nishikawa, "Low-Powered Self-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling," in Proc. of PDPTA, pp.618-624, July 2013.

[3] Kazuhiro Aoki, Hiroshi Ishii, Makoto Iwata, and Hiroaki Nishikawa, "A Comprehensive Evaluation of ULP-DDNS by Platform Simulator," in Proc. of PDPTA, pp.445-451, July 2012.

[4] Shuji Sannomiya, Kazuhiro Aoki, Makoto Iwata, and Hiroaki Nishikawa, "Power-Performance Verification of Ultra-Low-Power Data-Driven Networking Processor: ULP-CUE," in Proc. of PDPTA, pp.465-471, July 2012.

[5] C. J. Myers, "Asynchronous circuit design," Univ. of Utah John Wiley & Sons, Inc., 2001.

[6] Kazuhiro Aoki, Shuji Sannomiya, and Hiroaki Nishikawa, "Data-Driven Sensor Networking System Simulator," in Proc. of PDPTA, July 2015.(to be published)

Data-Driven Sensor Networking System Simulator

Kazuhiro Aoki[1], **Shuji Sannomiya**[2] **and Hiroaki Nishikawa**[2]
[1]Information Infrastructure Laboratory, Inc., Tsukuba Science City, Ibaraki, JAPAN
[2]Faculty of Engineering, Information and Systems, University of Tsukuba,
Tsukuba Science City, Ibaraki, JAPAN

Abstract— *This paper describes an implementation of data-driven sensor networking system simulator which is essential to evaluate robust sensor networking system. Robustness and self-diagnostic ability are important to realize sensor networking system because it is difficult to maintain many sensors in sensor network such as IoT. For keeping sensor networking system available, it is crucial issue to decrease power consumption in sensor networking system. We proposed Ultra-Low-Power Data-Driven Networking System (ULP-DDNS) and evaluated that ULP-DDNS achieved 1/180 power consumption to conventional network system in ad hoc network. Furthermore, keeping constant load scheme and self-diagnostic scheme which resolves trouble sensor by monitoring sensors each other have been studied in order to realize robust data-driven sensor networking system. In this paper, we propose data-driven sensor networking system simulator which can evaluate the effects of keeping constant load scheme and self-diagnostic scheme include behavior of network communication as extension of data-driven platform simulator.*

Keywords: data-driven, sensor, networking, low-power, simulator

1. Introduction

IoT (Internet of Things) is broadly discussed as an interesting application of Internet [1]. Sensor network is essential to implement IoT because sensors can be interface between Internet and various things such as IC tag [2]. Then, it is necessary for sensor network to be robust because a lot of sensors must be kept available at all times. However, maintenance of many sensors is difficult if they are fragile and short life.

About lifetime, sensor network is mostly wireless network. Therefore, it is necessary to study ultra-low-power scheme for sensor network because power consumption is generally crucial issue [3]-[5]. And it is desirable to detect trouble of sensors by themselves in order to keep network available. Thus, self-diagnostic scheme of sensor network should be studied.

The authors have been studying an implementation of Ultra-Low-Power Data-Driven Networking System (ULP-DDNS) [6]. ULP-DDNS project has been aiming at development of data-driven networking system which can achieve ultra-low-power consumption. And we have evaluated that ULP-DDNS can achieve 1/180 power consumption against the present system.

ULP-DDNS project have applied mobile ad hoc network [7] to ULP-DDNS. Ad hoc network is an infrastructureless network and is a group of wireless devices that organize themselves in a mesh topology to find routes and relay packets from the hardware platform through the network layer to application. The authors have proposed Load-aware Dynamic Counter-based Flooding (LDCF) which is a flooding scheme to reduce traffic for ultra-low-power [8]. And Ultra-Low-Power Data-Driven Chip MultiProcessor (ULP-DDCMP) which is our data-driven chip multiprocessor has been applied to networking platform in order to reduce power consumption [9].

The authors have started research about data-driven sensor networking system based on ULP-DDNS. We aim at applying data-driven sensor networking system to security service and monitoring infrastructures such as bridge and tunnel because it is difficult to monitor at all times by manual. It is essential to evaluate power consumption as well as performance in the whole data-driven sensor networking system. Furthermore, we have proposed keeping constant load scheme of protocol handling and self-diagnostic scheme in order to realize long-lifetime of sensor networking system [10]. Data-driven sensor networking system simulator is the simulator which evaluates the effects in keeping constant load scheme and self-diagnostic scheme.

In this paper, the authors introduces ULP-DDNS on which data-driven sensor networking system is based. And we discuss about keeping constant load scheme and self-diagnostic scheme which utilize features of ULP-DDCMP. Furthermore, this paper describes about data-driven sensor networking system simulator which is added network simulation function to data-driven platform simulator. We also estimate the precision of simulation on the data-driven sen-

sor networking system simulator. Finally, this paper shows current status of this study and discusses about application of data-driven sensor networking system.

2. Data-Driven Sensor Networking System

This section reports data-driven sensor networking system by introduction of ULP-DDNS. Fig. 1 shows layer of data-driven sensor networking system. Data-driven sensor networking system is ULP-DDNS whose concrete application is defined. Data-driven sensor networking system has some platforms connected by network. Some platforms are based on ULP-DDCMP. ULP-DDCMP is chip multiprocessor which has 4 processor core. And, the processor cores are based on self-timed pipeline which is controlled by handshake between pipeline stages. Handshake control is the VLSI implementation by data-driven principle. Protocol handling and flooding scheme are implementation on ULP-DDCMP for ultra-low-power. On the data-driven sensor networking system, we have studied data-driven implementation of sensor applications.

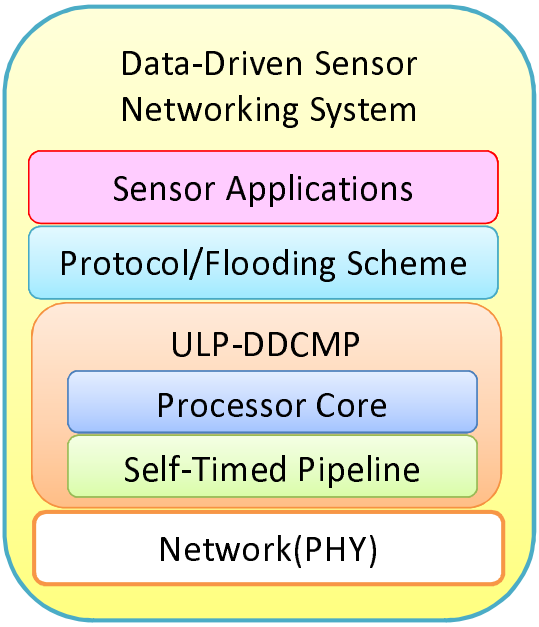


Fig. 1: Configuration of Data-Driven Sensor Networking System

2.1 Ultra-Low-Power Data-Driven Networking System

ULP-DDCMP is realized by data-driven VLSI implementation as shown in Sec. 2. Ultra-low-power features of ULP-DDCMP are produced from utilization of characteristics in the data-driven chip multiprocessor. For example, it is popular to utilize dynamic voltage scaling (DVS) and power gating (PG) for low power consumption of VLSI [11], [12]. The authors then proposed more effective utilization of them by using characteristics of data-driven VLSI implementation. Our DVS by PID controller realizes scaling on work by utilizing handshake control of self-timed pipeline. And, fine-grain power gating is implemented by utilizing self-timed pipeline [13].

Furthermore, the authors studied mobile ad hoc network as an applicable network architecture to disaster situation. In disaster situation, it is essential for network to work with low power consumption. And, effective information discovery is also important. At a same time, effective secure communication is needed [14]. They should be realized under the effective data transfer on ad hoc network. We have proposed these schemes in ultra low power consumption and we have evaluated effects of these schemes in reducing traffic and power consumption. The authors also studied load-aware dynamic counter-based flooding (LDCF) for streaming data. LDCF can achieve very low traffic and high reachability for streaming. As a result, the authors evaluated that ULP-DDNS achieved 1/180 power consumption to conventional network system in ad hoc network.

2.2 Self-Diagnostic Scheme on Data-Driven Networking Platform

This section refers to self-diagnostic scheme on ULP-DDNS platform which is based on ULP-DDCMP. ULP-DDCMP has characteristics which are linearity between power consumption and load as well as ultra-low-power consumption. Therefore, the authors have studied load-avoiding scheme which detects and avoids overload by monitoring power consumption because power consumption is proportional to load of ULP-DDCMP [9]. Besides, we have proposed keeping load constant scheme of protocol handling. And we have also studied lifetime estimation by monitoring power consumption as an application of the scheme. Furthermore, autonomous load distribution of data-driven networking processor for high energy efficiency has been studying [15].

As another application of the load-avoiding scheme, we have proposed self-diagnostic scheme on data-driven networking platform. Usually, sum of event detection by sensors shows a tendency because many events are apt to occur periodically. On the other hand, the sum of event is proportional to the load on data-driven sensor networking

system platform because the sum of event is equal to the sum of input to ULP-DDCMP. Besides, ULP-DDCMP can be monitored its own load by measuring power consumption because power consumption is proportional to load of ULP-DDCMP. Therefore, we can estimate the amount of load of ULP-DDCMP in each platform when sum of event detected by each sensor is periodical.

Then, ULP-DDCMP can detect trouble of the sensor by monitoring extraordinary power consumption which shows unusual input from the sensor. Thus, ULP-DDCMP can notify users of trouble of the sensor. The authors have started evaluating the effectiveness of the self-diagnostic scheme [10].

3. An Implementation of Data-Driven Sensor Networking System Simulator

3.1 Data-Driven Platform Simulator to Evaluate ULP-DDNS Platform

This subsection describes an implementation of data-driven platform simulator to evaluate power consumption of ULP-DDNS platform which applies ULP-DDCMP to networking processor [16]. Data-driven platform simulator has workspace which is a container of configuration of the platform. The workspace has following information.

- Chip multiprocessor (CMP)
- Router
  - Pipeline stages
  - Connection between pipeline stages
- Processor core in CMP
  - Pipeline stages
  - Connection between pipeline stages
- Chip specification
  - Ratio(power, area)
  - Voltage specification
- Routing table
- Schedule table
- Parameters of PID controller

Data-driven platform simulator records events which are sending/receiving packet and change voltage. A packet is the implementation of a token in data-driven principle. Data-driven platform simulator outputs time and power of each event as a event data. Data-driven platform simulator has an event scheduler. The event scheduler loads schedule table which is set packet flow by user. Data-driven platform simulator repeats following step until schedule in scheduler is finished.

- The simulator progress time until most immediate event time.
- The simulator process the event and add new event to scheduler as needed.

The events are mainly sending/receiving packet and change voltage in the schedule. this subsection explains transition of packet on self-timed pipelines which compose ULP-DDCMP. Fig. 2 is an example of self-timed pipeline configuration including mergence of two pipeline stages: $S0$ and $S1$. Fig. 3 shows sequence chart of handshaking on the configuration in shown Fig. 2. Self-timed Pipeline sends packets by handshaking between pipelines. In Fig. 3, pipeline stage $S1$ firstly receives a packet and sends acknowledge(ACK) signal to former pipeline stage to handshake with the pipeline. And $S0$ also receives another packet and sends ACK signal to former pipeline stage. Furthermore, $S1$ sends SEND signal to $S2$ which is following and merging stage of $S0$ and $S1$. Then, status of $S2$ is standby, therefore $S1$ immediately sends the packet which exist on $S1$ to $S2$, and $S2$ sends ACK signal to $S1$. Next, $S0$ sends SEND signal to $S2$, but status of $S2$ is active because $S2$ already received the packet from $S1$ and didn't send the packet to $S3$ yet. Therefore, $S0$ keeps another packet until status of $S2$ become standby. When $S2$ sends SEND signal and the packet to $S3$ and $S2$ receives ACK signal from $S3$, $S0$ sends another packet to $S2$ and $S2$ sends ACK signal to $S0$. Besides, $S3$ sends the packet to following stage, and receives ACK signal from following stage. Then, $S2$ can send another packet to $S3$ because status of $S3$ become standby. Fig. 3 shows "ACK time" and "SEND time". "ACK time" is time from receiving send signal to sending ack signal on the same stage. On the other hand, "SEND time" is time from receiving a packet to sending the packet on the same stage. When status of the pipeline stage is standby, a packet is received by the pipeline stage as soon as receiving signal. Data-driven platform simulator simulates the behavior of packet on the pipeline stages by scheduling to send SEND/ACK signal including latency in voltage at that time.

Data-driven platform simulator can simulate voltage control by PID controller. Table 1 shows parameters for voltage control simulation. The simulator samples power consumption and voltage of chip multiprocessor core as an input. And the simulator output voltage which is calculated by PID controller. Fig. 4 shows PID controller. Table 2 explains functions in Fig. 4.

When input sampling time is $t_i = i \cdot T_{pidin}$, consumption current I is shown following equation:

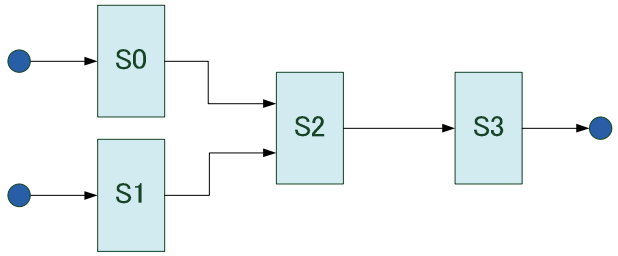


Fig. 2: Topology of Merge on Self-Timed Pipeline

$$I(t_i) = \frac{P(t_i)}{V(t_i)}$$

When target voltage V_{sp} is

$$V_{sp}(t_i) = min\{V_{MAX}, max\{V_{MIN}, \alpha I(t_i) + \beta\}\}$$

voltage deviation e is shown following equation:

$$e(t_i) = \begin{cases} 0, & i = 0 \\ V_{sp}(t_i) - V(t_i), & i > 0 \end{cases}$$

Then, the output value of PID controller V_{MV} is shown following equation:

$$V_{MP}(t_i) = K_p e(t_i) + K_i \sum_{x=1}^{i} e(t_x) + K_d(e(t_i) - e(t_{i-1}))$$

$$\tilde{V}_{MV}(t_i + T_{pid}) = min\{V_{step}, \quad max\{-V_{step}, V_{MP}(t_i)\}\}$$

$$V_{MV}(t) = \begin{cases} 0, & 0 \le t < t_0 + T_{pid} \\ \tilde{V}_{MV}(t_i + T_{pid}), & t_i + T_{pid} \le t < t_{i+1} + T_{pid} \end{cases}$$

Then, output voltage V_{PV} is shown following equation when output sampling time is $t_j = j \cdot T_{pidout}$:

$$\tilde{V}_{PV}(t_j) = \begin{cases} V_{MIN}, & j = 0 \\ V_{PV}(t_{j-1}) + V_{MV}(t_j), & j > 0 \end{cases}$$

$$V_{PV}(t) = \tilde{V}_{PV}(tj), \qquad t_j \le t < t_{j+1}$$

3.2 Extension of Data-Driven Platform Simulator to Evaluate Networking System

This subsection describes about extension of data-driven platform simulator to evaluate sensor networking system. The authors named the extended simulator "Data-driven sensor networking system simulator". Data-driven sensor networking system simulator adds the information of platform network and network routing table to the workspace as the simulator configuration. Fig. 5 shows an example of platform network.

Table 1: Parameters for Voltage Control

Parameter	*Unit*	*Definition*
T_{pidin}	psec.	Sampling interval of input current
T_{pidout}	psec.	Sampling interval of output current
T_{pid}	psec.	Latency of PID controller
K_p		Gain of proportion in PID controller
K_i		Gain of integral in PID controller
K_d		Gain of derivation in PID controller
α		Coefficient for target voltage
β		Constant for target voltage
V_{MIN}	V	Minimum voltage in output voltage
V_{MAX}	V	Maximum voltage in output voltage
V_{step}	V	Maximum amount of variation by PID controller

Table 2: Functions of Voltage Calculation in PID Controller

Function	*Definition*
$P(t)$	Function of power consumption in time t
$V(t)$	Function of voltage in time t
$I(t)$	Function of current in time t
$V_{SP}(t)$	Function of target voltage in time t
$V_{MV}(t)$	Function of output value in time t
$V_{PV}(t)$	Function of output voltage in time t

Platform is defined by CMP, chip specification, routing table, and parameters of PID controller in the simulator. And, connection between platforms is defined by selection of two connected platforms. Then, the connection has communication latency. The communication latency is symmetrical between connected platforms, and can be set by user in simulation. Data-driven sensor networking system simulator defines routing information of platform network in the network routing table. Some routing can be defined in the network routing table. Network packet flow is defined in the routing information. Users selects following items in each hop.

- Destination platform of the current hop
- Input network packet in the current hop

Elements of input network packet must be linked to available pipeline packets which are defined in the routing table of destination platform. Therefore, network routing can be defined by pipeline packets. In the network routing table, input of platform is defined receiving from out of network in first hop, and output of platform is defined sending to out of network in last hop.

The authors estimates the precision of power consumption which is calculated by data-driven sensor networking system simulator. Time slice of the simulator is 1 $psec.$ In the configuration, SEND time and ACK time which are defined is $O(nsec.)$, and latency of network routing is $O(msec.)$.

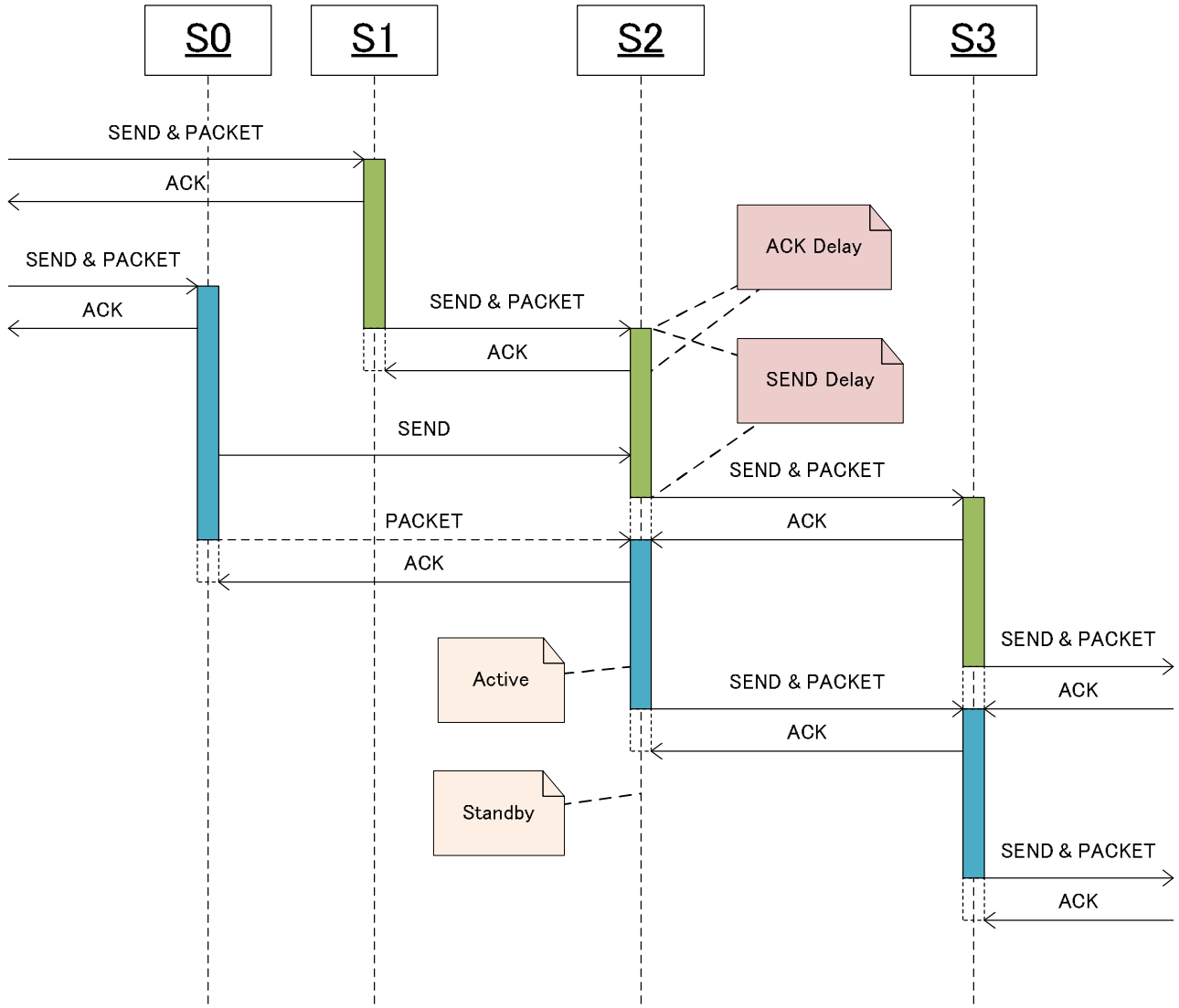


Fig. 3: Sequence of Self-Timed Elastic Pipeline

On the other hand, the order of power is $O(mW)$ in the simulator. Therefore, the simulator can derives from power consumption in $O(mWsec.)$ at least. This precision is enough to estimate lifetime of sensor networking system because sensor networking system usually works for more than a few years. The authors have been studying lifetime prediction of data-driven sensor networking system platform to schedule battery replacement or charge economically [10]. And the lifetime prediction scheme will be evaluated the precision and the validity with data-driven sensor networking system simulator.

4. Conclusion

This paper firstly introduced ULP-DDNS on which data-driven sensor networking system is based. And we discuss about keeping constant load scheme and self-diagnostic scheme which utilize features of ULP-DDCMP. Furthermore, this paper described about data-driven platform simulator, and data-driven sensor networking system simulator which is added platform network and network routing table to data-driven platform simulator. Besides, we estimated the precision of simulation on the data-driven sensor networking system simulator.

The authors have been studying applications of data-driven sensor networking system. Our colleagues have stud-

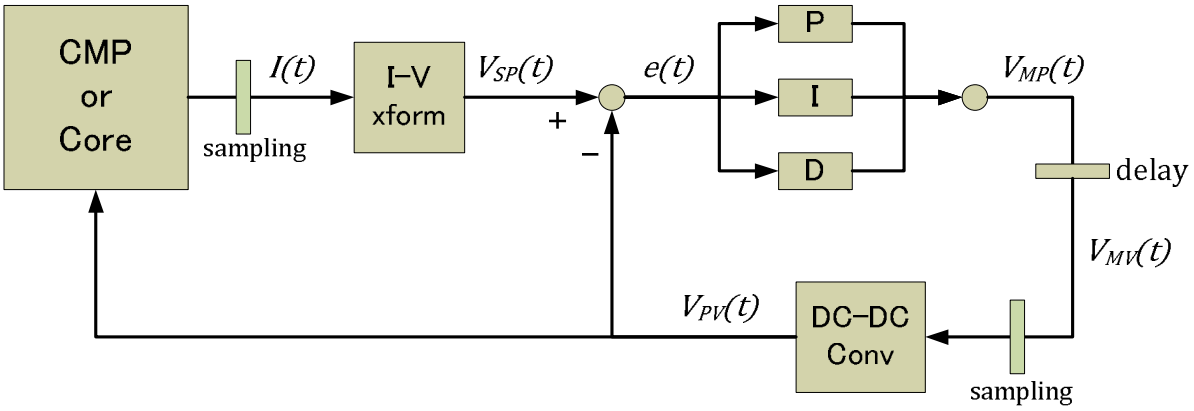


Fig. 4: Configuration of PID Control on Data-Driven Platform Simulator

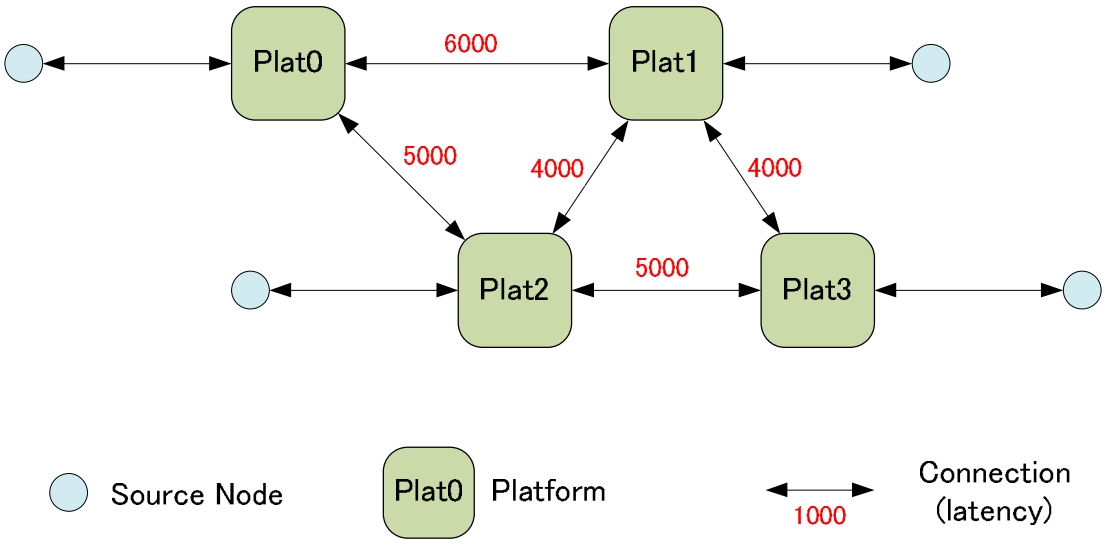


Fig. 5: Configuration of Network Topology on Data-Driven Sensor Networking System Simulator

ied various communication systems, and they have evaluated schemes for security service and ad hoc network [17]-[20]. Our colleagues have also studied self-timed pipeline circuit for Fast Fourier Transform (FFT) because FFT is important application to process image from sensor such as image sensor [21]. On the other hand, the authors think data-driven sensor networking system is suitable for the application which require robustness and longlife to sensor networking system. We will apply the system to monitoring infrastructure and machines in factories because it is essential for the sensor networking system to be robustness and longlife.

Acknowledgments

Although it is impossible to give credit individually to all those who organized and supported the CUE project and the ULP-DDNS project, the authors would like to express their sincere appreciation to all the colleagues in the project.

The CUE project and the ULP-DDNS project are partially supported by Program for Creating STart-ups from Advanced Research and Technology (START) and Core Research for Evolutional Science and Technology (CREST), Japan Science and Technology Agency, Strategic Information and Communications R&D Promotion Programme (SCOPE), Ministry of Internal Affairs and Communications, Japan, and the Grants-in-Aid for Scientific Research of Japan Society for the Promotion of Science and Semiconductor

Technology Academic Research Center (STARC). And, this work is supported by VLSI Design and Education Center(VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Journal of Future Generation Computer Systems, Vol.29, Issue 7, pp.1645–1660, Sep. 2013

[2] C. Alcaraz, P. Najera, J. Lopez, and R. Roman, "Wireless Sensor Networks and the Internet of Things: Do We Need a Complete Integration? ," 1st Int'l Workshop on the Security of the Internet of Things (SecIoT'10), CD-ROM, Dec. 2010.

[3] T. Inagaki and S. Ishihara, "HGAF: A Power Saving Scheme for Wireless Sensor Networks," IPSJ Journal Vol.50, No.10, pp. 2520–2531, Oct. 2009.

[4] A. Keshavarz-Haddad and R. Riedi, "Bounds on the benefit of network coding: Throughput and energy saving in wireless networks," IEEE INFOCOM 2008, Phoenix, Arizona, USA, pp. 376–384, April 2008.

[5] L. Sukyoung, K. Laeyoung, and K. Hojin, "MIPv6-Based Power Saving Scheme in Integrated WLAN and Cellular Networks," IEICE transactions on communications Vol. E90-B, No. 10, pp. 2780–2783, Oct. 2007.

[6] Hiroaki Nishikawa, Hiroshi Ishii, and Makoto Iwata, "Collaborative Research Project on Ultra-Low-Power Data-Driven Networking System," Proc. of the 2008 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 697–703, July 2008.

[7] Jie Wu, Ivan Stojmenovic, "Ad Hoc Networks," IEEE Computer, Vol.37, No.2, pp.29–31, Feb. 2004.

[8] Keisuke Utsu, Hiroaki Nishikawa, and Hiroshi Ishii, "Broadcast Voice Streaming by Load-aware Flooding over Ad Hoc Networks achieving Reduction of Traffic and Power Consumption," Proc. of the 2011 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp.455–461, July 2011.

[9] Shuji Sannomiya, Yukikuni Nishida, Makoto Iwata, and Hiroaki Nishikawa, "An Overload-Free Data-Driven Ultra-Low-Power Networking Platform Architecture, " Proc. of the 2013 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp.604–610, July 2013.

[10] Shuji Sannomiya and Hiroaki Nishikawa, "Highly-Dependable and Long-Lifetime Data-Driven Networking Processor with Energy Assurance Capability," Proc. of the 2015 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, PDP7009, July 2015.

[11] Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen, "Low Power CMOS Digital Design," IEEE Trans. on Solid-state Circuits., vol. 27, No. 4, pp.473–483, Apr. 1992.

[12] Shin-ichiro Mutoh, Satoshi Shigematsu, Yoshinori Gotoh, and Shinsuke Konaka, "Design Method of MTCMOS Power Switch for Low-Voltage High-Speed LSIs," Proc. of Asia and South Pacific Design Automation Conference, Hong Kong, pp.113–116, Jan. 1999.

[13] Kei Miyagi, Shuji Sannomiya, Makoto Iwata, and Hiroaki Nishikawa, "Low-Powered Self-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling," Proc. of the 2013 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp.618–624, July 2013.

[14] Hiroshi Ishii, Keisuke Utsu and Hiroaki Nishikawa "Integrated Evaluation on Effectiveness of ULP-DDNS Networking Layer," Proc. of the 2012 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp.452–457, July 2012.

[15] Shuji Sannomiya and Hiroaki Nishikawa, "Energy Efficient Data-Driven Networking Processor with Autonomous Load Distribution Capability," Proc. of the 2014 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp.514–520, July 2014.

[16] Kazuhiro Aoki, Shuji Sannomiya, Makoto Iwata, Hiroshi Ishii, and Hiroaki Nishikawa, "An Implementation of Platform Simulator for Congestion-Free Ultra-Low-Power Data-Driven Networking System," Proc. of the 2013 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp.611–617, July 2013.

[17] Ayami Manaka, Akio Ogata, Hirohide Matsuzaka, Hayato Taniguchi, Masaya Nomoto, Yasuhiro Nozawa, Minoru Fukuzaki, Hiroshi Ishii, and Keisuke Utsu, "A Concept of Community Care System and Community Information Network," Proc. of the 2015 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, PDP7005, July 2015.

[18] Ayami Manaka, Tomomi Itoh, Yasuhiro Nozawa, Chee Onn Chow, Minoru Fukuzaki, Hiroshi Ishii, and Keisuke Utsu, "Performance Evaluation of a Community Information Network for a Daily Life Support System," Proc. of the 2015 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, PDP7006, July 2015.

[19] Kei Kobayashi, Yosuke Totani, Keisuke Utsu, and Hiroshi Ishii, "A Study on Secure Communication Method Using Secret Sharing Schemes over MANET," Proc. of the 2015 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, PDP7007, July 2015.

[20] Phonepadith Phounmmavong, Keisuke Utsu, Hiroaki Nishikawa, and Hiroshi Ishii, "Efficient Location-aided Route Discovery Mechanism for Ad-hoc networks," Proc. of the 2015 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, PDP7008, July 2015.

[21] Norifumi Uno and Makoto Iwata, "Self-Timed MM-FFT Circuit and its Performance Evaluation," Proc. of the 2015 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, PDP7011, July 2015.

Self-Timed MM-FFT Circuit and its Performance Evaluation

Norifumi UNO and Makoto IWATA
Graduate School of Engineering, Kochi University of Technology,
Kami, Kochi, 782-8502 Japan

Abstract— *For realizing future heterogeneous wireless communication systems, flexible and powerful fast Fourier transform (FFT) circuit is indispensable. This paper presents a basic configuration of multimode and multichannel FFT (MM-FFT) circuit based on self-timed pipeline (STP). The proposed circuit is designed by using 65 nm CMOS standard cells and evaluated in terms of diverse conditions.*

The evaluation results show that the proposed circuits perform under the ideal pipeline efficiency as long as the target FFT has enough parallelism for processing resource. Furthermore, it is revealed that the proposed MM-FFT circuit can support present heterogeneous wireless network applications, e.g., 2.4 G sample/s for wireless personal area network (WPAN), 2-channel 20 M sample/s for wireless local area network (WLAN), and 2-channel 40 M sample/s for mobile broadband wireless access (MBWA).

Keywords: heterogeneous wireless network, multimode and multichannel, FFT, self-timed pipeline

1. Introduction

The amount of data traffic on wireless network has been increasing exponentially in recent years. Meanwhile, usage of smart wireless devices has widely spread. To accommodate such huge traffic and provide uniform user experience in ubiquitous environment, heterogeneous wireless network (HetNet) has been investigated [1].

Our research project aims to establish a self-timed pipeline (STP) implementation for the dependable wireless systems (DWS) [2] supporting multimode and multiband interfaces like HetNet. Since the self-timed pipeline (STP) circuit inherently has a clockless passive operation mode [3], [4], it can flexibly process any combination of signal streams even if they are sampled at different frequencies.

The fast Fourier transform (FFT) is one of heaviest tasks in wireless modem. In the heterogeneous wireless network, FFT circuits are required to be high speed and low power consumption but also flexible with heterogeneous wireless channels. Although multimode FFT circuit with flexible radix configuration has been proposed in [5], [6], and so on. They are dedicated to single channel stream, i.e., they cannot cope with multichannel signal streams at the same time. In [7], a basic circular pipeline circuit for multimode and multichannel FFT (MM-FFT) based on STP has been proposed. It can operate multiple input signal streams because of asynchronous handshake behavior of the STP. Even if their input signal streams are sampled at different rate, this circuit accepts and processes them together without any strict scheduling of processing time slots. However, pipeline efficiency (occupancy) of this circuit was not optimized so that it resulted in some degradation of pipeline throughput. Suppose if the pipeline efficiency can be maximized, the maximum performance was limited by the transistor speed. Therefore, some parallelization technique is necessary for faster wireless channel [8].

This paper proposes a basic self-timed MM-FFT circuit as a modified version of previous proposal and discusses its pipeline efficiency for every mode of FFT points. The paper also proposes an expansion scheme of MM-FFT circuit for faster wireless channel and discusses its efficiency in multimode and multichannel configuration. After that, the performance of the proposed circuit is evaluated based on 65 nm CMOS circuit design.

2. MM-FFT with Single-Circular STP

2.1 Basic Scheme of MM-FFT

Basic scheme of multimode and multichannel FFT is organized to execute multiple butterfly operations in the pipelined manner. As shown in Figure 1, it is constructed by a single circular STP and consists of merge stage, buffer memory stage, reconfigurable butterfly operation stage, and ID handler stage. Each data set within all pipeline stages is distinguished by its identifier $ID(ch, step, btf)$, where ch denotes an identifier of input channel, $step$ denotes a recursive step to which the butterfly operation belongs, and btf denotes an identifier of a set of butterfly operations within the step. In principle, since all the necessary parameters of FFT such as the number of FFT points, indices of twiddle factors, and so on can be derived from $ID(ch, step, btf)$, this scheme can accept multimode and multichannel input streams where every stream can be calculated in different mode, i.e., the number of FFT points or sampling rate.

In this scheme, the merge stage receives N_{ch} input data from a channel ch consecutively. The input data reaching to the buffer memory stage is written in a place associated with index i of the packet. The buffer memory stage reads a set of r input data which includes multiple sets of radix-r butterfly operands from dual r-way memory banks at the same time, their corresponding twiddle factors are read from TF lookup memory. The butterfly operation stage can be reconfigured by adapting to a required radix r_{req}. In this stage, so-called

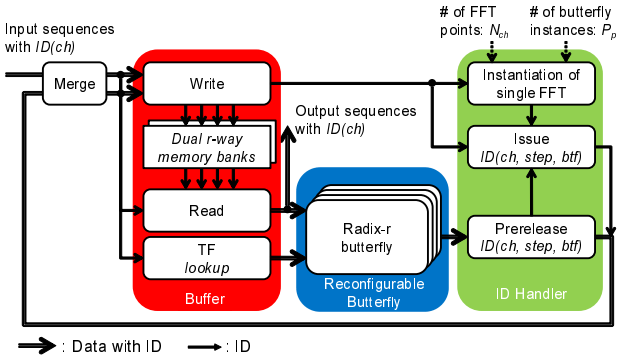


Fig. 1: Basic scheme of MM-FFT.

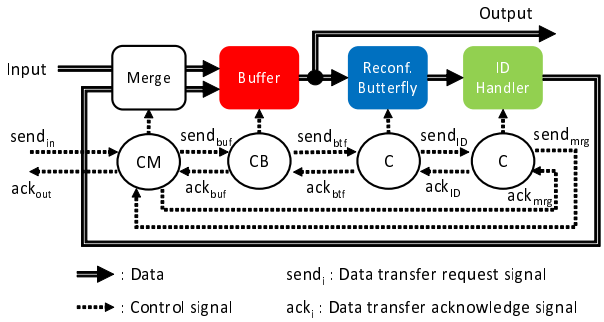


Fig. 2: STP-based MM-FFT circuit.

data-parallel execution of multiple butterfly operations can be realized. The number of butterfly operations executable in parallel is decided by r/r_{req}. For example, if the butterfly operation stage composed of 7 multipliers, i.e., $r = 8$, four radix-2 butterfly operations can be calculated in parallel, where r_{req} is 2 and r/r_{req} is 4. In the context of parallel processing, r can be associated with the degree of data parallelism P_d. After the butterfly operations, sets of their resultant data will be intermediately stored at the buffer memory stage. During this, the ID handler stage issues a new identifier of next butterfly operation set as long as pipeline processing resource is available. In the scheme, this pipeline processing resource is abstracted as the degree of pipeline parallelism P_p.

If each pipeline processing stage in this scheme autonomously behaves along with the processing requirement receiving from its neighbor stages, the scheme must accept different sampling rate signals and process them in the pipeline. Therefore, a basic idea of an STP implementation for MM-FFT was proposed in [7]. Figure 2 shows an overview of an STP-based MM-FFT circuit. In this implementation, every pipeline stage is basically controlled by a self-timed data-transfer control circuit C which handshakes among adjacent C elements by using both data transfer request signal $send_i$ and acknowledge signal ack_i. CM based on the basic C in the figure is a merging control circuit from two pipelines and CB is a branching control circuit to two pipelines.

However, previous STP implementation could not maximize its pipeline efficiency so that its performance was limited. This was because the conventional merging control circuit CM employed in the implementation arbitrated two input data sets under the first-come-first-serviced (FCFS) basis and sequentially transferred them in the prioritized order. This means that the pipeline throughput of the resultant data of butterfly operations is degraded in almost half while input data arrive at the merge stage.

In order to solve this throughput degradation issue in the STP-based MM-FFT, a generalized C element proposed in [4] is introduced in this paper. This C element circuit supports all possible interaction between two pipelines so that two sets of data flowing from two pipelines can be packed into a set of data and it is transferred only if two sets of input data arrive at the stage simultaneously. Otherwise, this C circuit behaves as a conventional merging control circuit with the FCFS basis.

2.2 Efficiency of Basic MM-FFT circuit

Basic handshake timing in the STP is shown in Figure 3. In case of no congestion in the pipeline, pipeline throughput and efficiency on the STP are defined as $1/(T_f + T_r)$ and $T_f/(T_f + T_r)$ respectively, where T_f denotes forwarding latency required to transfer valid data from one stage to its neighbor stage and T_r backward latency which is enough to hold the data at the stage [3]. Thus, the maximum number of butterfly instances executable within the STP can be represented as $(S * T_f)/(T_f + T_r)$, where S denotes the number of pipeline stages composing MM-FFT. Since it indicates the potential pipeline processing resource, the total number of P_p assigned for each channel must not be exceeded.

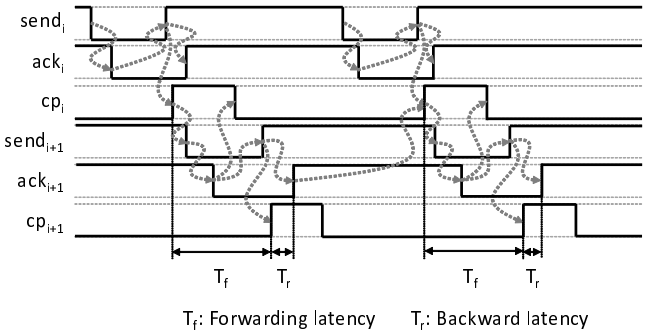


Fig. 3: Timing chart of self-timed pipeline.

To simplify the efficiency analysis of the basic self-timed MM-FFT circuit, we suppose a single stream of input signals. If the number of butterfly operation executable in parallel is enough to consume the pipeline processing

resource, the processing time of N_{ch}-point FFT is estimated as following equation:

$$T_{FFT} = \frac{N_{ch}\lceil \log_r N_{ch} \rceil}{rP_p} ST_f \quad (1)$$

where $\lceil \log_r N_{ch} \rceil / r$ indicates the number of steps for calculating N_{ch}-point FFT, $N_{ch}\lceil \log_r N_{ch} \rceil / r$ is the number of butterfly operation sets for the FFT, and ST_f expresses one circulation time when a set of data flows in the STP ring.

However, there are some data dependencies between butterfly operations belonging to neighbor steps. In order to guarantee these data dependencies, some pipeline stalls occur. For example, in the case of 512-point FFT, r=8, and P_p=16, 8 stalls occur. Figure 4 illustrates the pipelined execution timing of radix-8 butterfly operations. As shown in this figure, the 56-th resultant data in the first step is produced in the last 8-th butterfly operation so that the first butterfly operation in the second step has to wait for it. In this case, the number of butterfly operation executable in parallel is limited to 8. This indicates that 8 stalls occur in spite of available pipeline parallelism P_p=16.

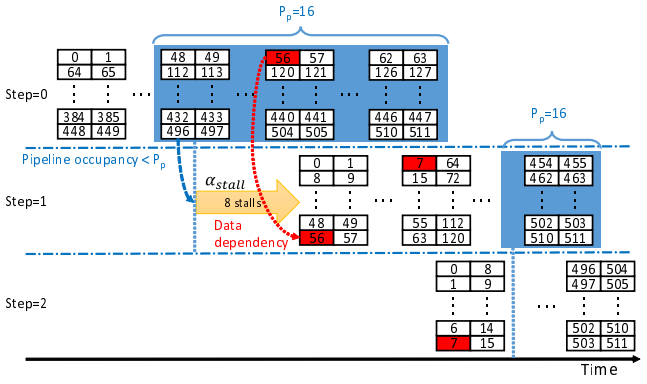


Fig. 4: Timing chart of single MM-FFT (N_{ch}=512).

In such case, the processing time of FFT is lengthened by those pipeline stalls so that Equation (1) should be modified as follow:

$$T_{FFT} = (\frac{N_{ch}\lceil \log_r N_{ch} \rceil}{rP_p} + \frac{\alpha_{stall}}{P_p}) ST_f \quad (2)$$

Therefore, the throughput of the STP-based MM-FFT circuit can be estimated by equation (3).

$$Throughput = \frac{N_{ch}}{(\frac{N_{ch}}{r}\lceil \log_r N_{ch} \rceil + \alpha_{stall})\frac{ST_f}{P_p}} \quad (3)$$

This equation indicates that it is important to choose an optimal combination of radixes in order to achieve the maximum throughput with minimizing pipeline stalls. Therefore, we introduce following constraints for the radix combination:

1) to minimize the number of recursive FFT steps.
2) to minimize pipeline stalls.

Based on these constraints, an optimal combination of radixes for a specific FFT point, r, and P_p can be designed as shown in Table 1.

Table 1: Radix combination of MM-FFT (r=8, P_p=16).

# of FFT points	Radix	Efficiency
2048	8-8-8-4	1
1024	8-8-4-4	1
512	8-8-8	0.96
256	8-8-4	1
128	8-4-4	0.77

If any pipeline stall does not occur, the normalized efficiency will become 1. Otherwise, the efficiency will be less than 1. In case of 512 points FFT, 8 stalls occur so that the efficiency will be degraded to 0.96. In case of 128 points FFT, the number of radix-8 butterfly operations is 16 and 14 stalls happen so that the steady pipeline processing cannot realized. In other cases, the proposed MM-FFT scheme could work well without any pipeline stall and achieve the maximum efficiency.

If the MM-FFT module has to accept multichannel signal streams, the logical pipeline parallelism P_p should be appropriately allocated to each channel within the physical pipeline processing resource. This allocation task can be conducted like the Table 1. If the physical pipeline resource is not enough to accept all of the required input channels, the throughput of the MM-FFT will be required to be improved further as discussed in the following section.

3. Extended Configuration of MM-FFT

The basic MM-FFT circuit proposed in the previous section has an upper limit of the throughput because the throughput of STP is fixed by the circuitry parameters of the STP, T_f and T_r under the same CMOS process condition. In order to achieve higher throughput than that of the single MM-FFT circuit, it is necessary to introduce multiple MM-FFT circuit modules. Therefore, in this section, an extended configuration of MM-FFT is proposed to utilize macroscopic parallelism inherent in FFT's recursive calculation in terms of temporal and spatial parallelism. Furthermore, the performance of the proposed MM-FFT configuration is discussed with its throughput and pipeline efficiency under the multimode environment.

3.1 Extended MM-FFT

FFT is recursively calculated by butterfly operations. Butterfly operations belonging to a step can be executed in concurrent. Therefore, FFT calculation structure can be divided into both temporal and spatial directions. In this viewpoint, the minimum configuration of multiple MM-FFT circuit modules can be organized as 2×2 MM-FFT. Figure 5 shows a block diagram of the 2×2 MM-FFT.

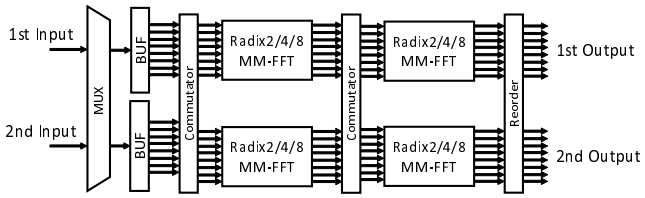


Fig. 5: Block diagram of 2×2 MM-FFT.

It consists of a multiplexer, two serial/parallel buffers, the commutator of former and latter stages, four basic MM-FFT modules, and a reorder module. At first, the multiplexer receives two streams of input signals and transfers them to either upper or lower input buffer. Each of them de-serializes input signals, i.e., it packs r input signals into a packed data. The packed data is transferred to the commutator. The commutator of former or latter stage produces a set of butterfly operands and sends it to either upper or lower MM-FFT module. The commutator of the former stages passes even and odd input data to the former stages. Another commutator passes first and last part data to the latter stages. Four MM-FFT modules execute a set of butterfly operations as well as the basic MM-FFT. Finally, the resultant data are reordered at the last stage. Thus, the 2×2 MM-FFT circuit achieves about four times higher throughput compared to that of the basic single MM-FFT circuit. But, this can be realized in the ideal case. In the following subsection, the efficiency of the proposed MM-FFT configuration will be discussed in terms of various conditions.

3.2 Efficiency of Extended MM-FFT

At first, we consider the efficiency of 1×1 MM-FFT circuit shown in Figure 6. It utilizes only temporal parallelism, i.e., pipelined parallelism inherent in the FFT recursive calculation structure to improve the throughput. In this configuration, the commutator between the former and latter stages can be omitted because the buffer stage within the basic MM-FFT plays the same role.

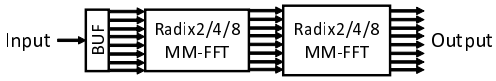


Fig. 6: Block diagram of 1×1 MM-FFT.

In addition to the constraints on the radix combination for the basic MM-FFT, the following constraints should be adopted to allocate the FFT recursive structure into two MM-FFT modules.

1) to minimize the total number of recursive FFT steps.
2) to equally allocate these steps to two MM-FFT modules.
3) to minimize pipeline stalls.
4) to maximize non-strict execution between two MM-FFT modules, i.e., to minimize the total processing time of FFT.

The additional constraints 2) and 4) are related to improve macroscopic pipeline throughput. It is noted that the achievable throughput is not proportional to the inverse of the total FFT processing time. The equation (3) is applied to each MM-FFT module and the lower throughput within them decides the total throughput of the 1×1 MM-FFT circuit.

According to these constraints, the radix combinations can be designed as listed in the Table 2. The table also shows the efficiency of the 1×1 MM-FFT circuit in case of $P_p = 16$ and $r = 8$ and which MM-FFT module, former (F) or latter (L), governs the total throughput.

Table 2: Radix combination of 1×1 MM-FFT.

# of FFT points	Radix		Efficiency
	Former	Latter	F/L
2048	8-8	4-8	F/L=1
1024	8-8	2-8	F/L=1
512	8	8-8	L=1
256	8	4-8	L=1
128	8	2-8	L=0.97

In this configuration, even if there are strict data dependencies between two MM-FFT modules, any pipeline stall will happen at all. This is because the latter MM-FFT has the buffer stage which absorbs the waiting time for operand availability and this latency can be hidden by the pipeline processing of the previous set of input signals. As a result, the efficiency of the latter MM-FFT can be kept at 1. Therefore, in case of 512 points FFT, the efficiency becomes 1 although it is 0.96 in the single MM-FFT circuit.

According to the above discussions, we will consider the efficiency of the 2×2 MM-FFT circuit shown in Figure 5, which utilizes the spatial parallelism of the FFT structure. For example, the 2×2 MM-FFT with 512 point FFT, r=8, and P_p=16 behaves as shown in Figure 7.

In this configuration, butterfly operations in a step is divided into half and each of them is allocated to the upper or lower MM-FFT modules. Therefore, a stream of input data is separated into odd and even signal sequences in front of the former stage. Furthermore, the commutator between the former and latter MM-FFT modules redistributes the intermediate data to the following upper and lower MM-FFT modules by classifying them into the first half and the last half of N_{ch} data set.

Because the number of butterfly operations per step allocated to each MM-FFT module is reduced in half, the pipelined instances executable at the same time also become half. Compared with the case of the 1×1 MM-FFT, the risk of pipeline stalls increases. Fortunately, in case of 512 points FFT, there is no pipeline stalls as shown in the timing chart. However, in case of 256 and 128 points FFT, the pipeline stalls increase slightly.

In order to apply Equation (3) to the multiple MM-FFT configurations, it should be modified as following equation.

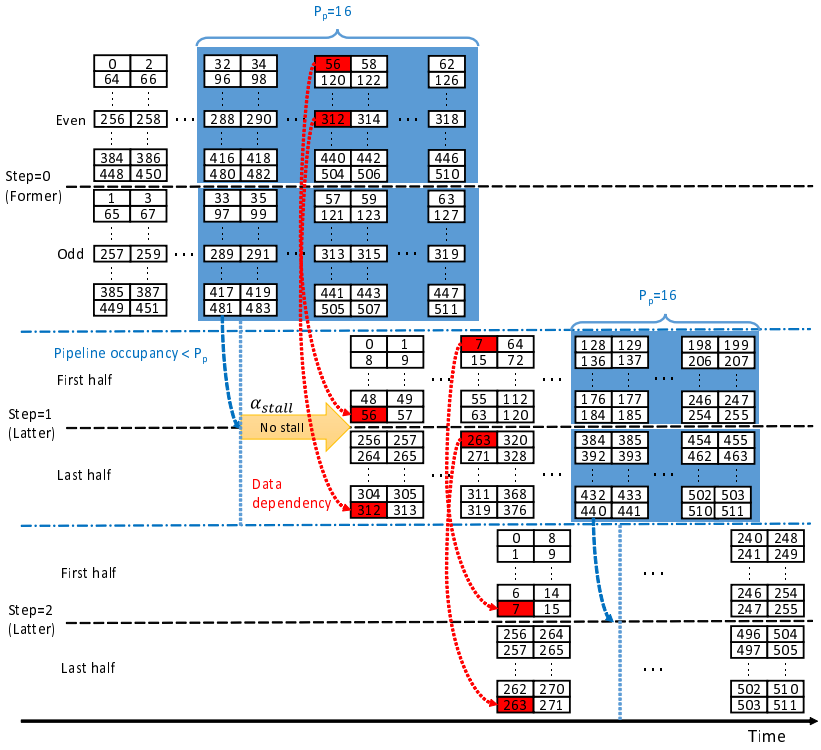


Fig. 7: Timing chart of 2×2 MM-FFT (N_{ch}=512, r=8).

$$Throughput = \frac{N_{ch}}{(\frac{N_{ch}}{rP_s}\lceil \log_r N_{ch} \rceil + \alpha_{stall})\frac{ST_f}{P_p}} \quad (4)$$

where P_s denotes the degree of spatial parallelism.

Based on the above discussion, Table 3 summarizes the radix combination and the efficiency of the 2×2 MM-FFT circuit.

Table 3: Radix combination of 2×2 MM-FFT.

# of FFT points	Radix		Efficiency
	Former	Latter	F/L
2048	8-8	4-8	F/L=1
1024	8-8	2-8	F/L=1
512	8	8-8	L=1
256	8	4-8	L=0.91
128	8	2-8	L=0.94

In case of 128 points FFT, the number of radix-8 butterfly operations at a step is 16 and thus only 8 butterfly operations per step are executed in the pipeline. This means that its parallelism is not enough for P_p=16. In cases of over 256 points, the parallelism of the FFT is enough for P_p=16 and the efficiency becomes 1 even if the butterfly operations are distributed to the upper and lower MM-FFT modules. Hence, it is necessary to decide pipeline parallelisms depending on the number of points N_{ch}.

Although Table 3 shows the optimal radix combination only for the single input channel, other radix combinations may be better for multiple input channels in the heterogeneous wireless communication systems. Therefore, the radix combination for each channel should be dynamically determined depending on the required sampling rate and the number of FFT points. Needless to say, the proposed 2×2 MM-FFT can be reorganized flexibly and dynamically. For example, Figure 8 shows a reconfigured 2×2 MM-FFT accepting two heterogeneous input channels.

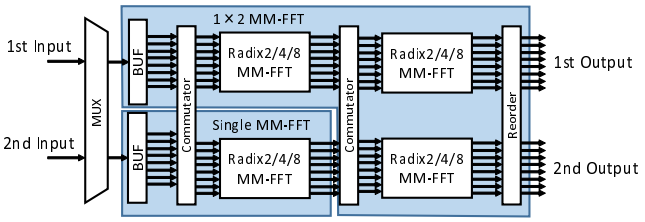


Fig. 8: Reconfiguration of 2×2 MM-FFT.

In such a case, P_p and the butterfly operations at each step should be appropriately allocated not only to achieve higher efficiency and but also to balance the pipeline processing load on each MM-FFT module. These dynamic scheduling issues should be studied further by taking account of the trade-off between optimality and overhead of scheduling schemes.

4. Evaluation

For the preliminary evaluation, an MM-FFT circuit with a single circular STP was designed using a 65 nm CMOS standard cell library. The specifications of this circuit are shown in Table 4. The designed STP circuit was described by Verilog-HDL and synthesized by Design Compiler, Synopsys Inc.

Table 4: Specification of the designed MM-FFT circuit.

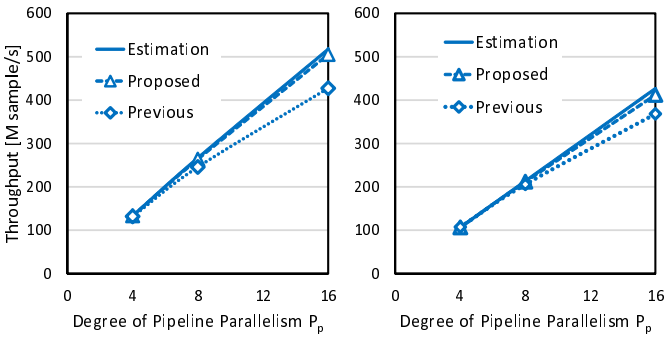
Process		65 nm CMOS
# of FFT points		64-1024
Radix		4
Pipeline parallelism P_p		1-16
Word length	Real	16
	Imaginary	16
# of STP stages S		30

Table 5 shows total cell area of single MM-FFT circuit. In the designed circuit, the signal data and twiddle factors are stored in SRAM modules: data mem. and TF mem. The cell area of memory accounts for 70 % of the total cell area.

Table 5: Total cell area of the synthesized MM-FFT circuit.

	Logic	data mem.	TF mem.
cell/bit	25865 cells	64K bit	24K bit
area [mm^2]	0.15	0.37	0.05

As for the performance of single MM-FFT circuit, RTL simulation result revealed that equation (3) could estimate the throughput as shown in Figure 9.



(a) N_{ch}=256 (b) N_{ch}=1024

Fig. 9: Performance of single MM-FFT (r=4, T_f=1ns).

In this figure, the horizontal axis denotes the degree of pipeline parallelism P_p and vertical axis indicates the maximum sampling rate (throughput) which can be achieved by the designed circuit. The throughput estimated by equation (3) is drawn by the solid line and the measured throughput of the designed circuit is drawn by the dotted line. The triangular denotes the measured throughput of the proposed circuit introducing the generalized C element. The diamond denotes the measured throughput of the previous circuit with the conventional merge control circuit. As seen in Figure 9, the throughput of the previous circuit is degraded due to the arbitrative merge control. On the contrary, the proposed MM-FFT circuit nearly achieves the theoretical throughput of ideal condition. These results imply that equation (3) estimates the actual throughput well.

Since the actual circuit design and simulation proves accuracy of the equations defined in the paper, achievable throughput of the 2×2 MM-FFT is estimated based on equation (4) as shown in Table 6.

Table 6: Pipeline efficiency and throughput of MM-FFT.

# of FFT points	2×2 Efficiency F/L	2×2 Throughput [G sample/s]	Single Throughput [G sample/s]
2048	F/L=1	4.27	1.07
1024	F/L=1	4.07	1.07
512	L=1	4.27	1.37
256	L=0.91	3.90	1.42
128	L=0.94	2.01	1.10

Table 6 shows if the pipeline efficiency is maximum value (pipeline efficiency=1), the extended MM-FFT circuit achieves the upper limit throughput of ideal condition. Since this circuit is designed to assume data transfer time T_f is 1 ns and eight data is simultaneously input, the maximum input rate is limited up to 8.0 G sample/s in the case of single input stream. Throughput of 2×2 MM-FFT circuit is 4 times faster than single MM-FFT circuit in case of N_{ch}=512, 1024, and 2048.

As for the practical evaluation of 2×2 MM-FFT circuit, the throughput corresponding to P_p is estimated for a HetNet application shown in Table 7. Figure 10 shows the estimation results for this application condition.

Table 7: Example specifications of heterogeneous air interfaces.

Air interface (Example)	WPAN (IEEE802.11ad)	WLAN (IEEE802.11n)	MBWA (LTE)
# of FFT points	512	2048	128
Sampling rate [G sample/s]	2.4	0.04	0.02
# of channels	1	2	2

In this figure, the horizontal axis denotes the degree of pipeline parallelism P_p and the vertical axis indicates the maximum sampling rate which can be accepted by the designed circuit. According to Figure 10, the 2×2 MM-FFT circuit is able to perform this configuration of HetNet applications. In both cases of P_p=16 and =8 at WLAN, the throughput is similar. In order to reduce the pipeline processing resource, P_p=16 should not be chosen for WLAN. Accordingly, the proposed 2×2 MM-FFT circuit

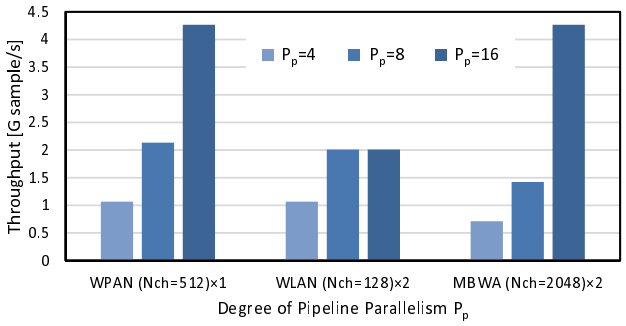


Fig. 10: 2×2 MM-FFT performance of heterogeneous wireless channels.

can share its pipeline processing resource with required wireless channels by controlling each P_p.

5. Conclusions

In this paper, a basic scheme of multimode and multichannel FFT (MM-FFT) circuit with a single circular self-timed pipeline (STP) is proposed. Furthermore, an extended configuration of multiple MM-FFT modules is proposed to improve the throughput. The performance of these proposed circuits are evaluated based on 65 nm CMOS circuit design. The evaluation results show that the proposed circuits perform under the ideal pipeline efficiency as long as the target FFT has enough parallelism for processing resource.

Furthermore, the extended configuration of MM-FFT circuit based on STP will be feasible to the required performance for current heterogeneous wireless communication, e.g., 2.4 G sample/s for WPAN, 2-channel 20 M sample/s for WLAN, 2-channel 40 M sample/s for MBWA. This evaluated condition is assumed for a typical case of HetNet so that benchmark evaluation in case of other usage scenario should be further studied.

Since smart wireless devices and terminals have to operate at low power consumption, the designed circuit must cooperate with typical low power techniques, e.g., dynamic voltage scaling (DVS) [9] and power gating (PG) [10]. Since the degree of pipeline parallelism P_p in the designed circuit can be adjusted along with dynamically-scaled supply voltage, the circuit could contribute to lower its power consumption. Quantitative evaluation on such energy efficiency will be reported in other article.

Acknowledgement

Although it is impossible to give credit individually to all those who organized and supported our project, the authors would like to express their sincere appreciation to all the colleagues in the project.

This research work was supported in part by Core Research for Evolutional Science and Technology (CREST), Japan Science and Technology Agency (JST). The circuit design work was supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

References

[1] R. Q. Hu, Y. Qian, S. Kota, and G. Giambene, "HetNets - A New Paradigm for Increasing Cellular Capacity and Coverage," IEEE Commun. Mag., Vol. 18, No. 3, pp. 8–9, June 2011.

[2] K. Tsubouchi, S. Kameda, and N. Suematsu, "Dependable Air," IEICE Trans. Electron., Vol. J95-C, No. 12, pp. 450–459, Dec. 2012 (in Japanese).

[3] H. Terada, M. Iwata, and S. Miyata, "DDMP's: Self-Timed Super-Pipelined Data-Driven Multimedia Processors," Proc. IEEE, Vol. 87, No. 2, pp. 282–296, Feb. 1999.

[4] K. Komatsu, S. Sannomiya, M. Iwata, H. Terada, S. Kameda, and K. Tsubouchi, "Interacting Self-Timed Pipelines and Elementary Coupling Control Modules," IEICE Trans. Fundamentals, Vol. E92-A, No. 7, pp. 1642–1651, Jul. 2009.

[5] S. N. Tang, C. H. Liao, and T. Y. Chang, "An Area- and Enegy-Efficient Multimode FFT Processor for WPAN/WLAN/WMAN Systems," IEEE J. Solid-State Circuits, vol. 47, No .6, pp. 1419–1435, June 2012.

[6] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined Radix-2^k Feedforward FFT Architectures," IEEE Trans. VLSI Sys., Vol. 21, No. 1, pp. 23–32, Jan. 2013.

[7] R. Taguchi, H. Ohiso, K. Mendori, K. Miyagi, and M. Iwata, "Self-Timed Single Circular Pipeline for Multiple FFTs," Proc. The 2013 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 625–630, July 2013.

[8] N. Uno, R Taguchi, and M. Iwata, "Spatial Parallelization of Self-Timed FFT Circuit," Proc. The 2014 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 521–527, July 2014.

[9] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, "Low-Powered Self-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling," Proc. The 2013 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 618–624, July 2013.

[10] K. Miyagi, M. Iwata, S. Sannomiya, and H. Nishikawa, "Self-Timed Pipeline with Fine Grain Power Gating and Its Evaluation," IEICE Trans. on Fundamentals, Vol. J97-A, No. 8, pp.554–564, Aug. 2014 (in Japanese).

SESSION

CSTAW 2015 - 7TH HIGH PERFORMANCE COMPUTING FOR COMPLEX SYSTEMS, THEORY AND APPLICATIONS WORKSHOP

Chair(s)

Dr. Lou D'Alotto
Dr. Georgios Sirakoulis
Dr. William Spataro
Dr. Giuseppe A. Trunfio

Accelerating Lava Flows Simulations with GPGPU and OpenCL

A. De Rango, M. Macrì, D. D'Ambrosio, W. Spataro
Department of Mathematics and Computer Science, University of Calabria, Italy

Abstract - *The introduction of the GPU (graphics processing units) has marked a revolution in the field of Parallel Computing allowing to achieve computational performance unimaginable until a few years ago. Widely adopted in the Scientific Computing Field, this hardware has proven to be extremely reliable and suitable to simulate Cellular Automata (CA) models for modeling complex systems whose evolution can be described in terms of local interactions. This paper presents an effective implementation of a well-known numerical model for simulating lava flows on Graphical Processing Units (GPU) based on the OpenCL (Open Computing Language) standard. Carried out experiments show that significant performance improvements in terms of speedup are achieved, adopting also some original optimizations strategies, confirming the validity of OpenCL and both low-cost and high-end graphics hardware as an alternative to expensive solutions for the simulation of CA models.*

Keywords: Cellular Automata, GPGPU, OpenCL, Parallel Software Tools, Modeling and Simulation.

1 Introduction

Numerical models are adopted in High Performance Computing (HPC) ([12]) for solving complex equation systems which rule the dynamics of complex systems as, for instance, a lava flow or a forest fire. In recent years, the introduction of the GPU (graphics processing units) has marked a revolution in the field of Parallel Computing allowing to achieve computational performance unimaginable until a few years ago. Nevertheless, GPU applications to the important field of Computational Fluid Dynamics (CFD) are increasing both for quantity and quality among the Scientific Community (e.g., [23], [11]).

With GPGPU (General Purpose computing with GPU) it is possible to obtain computational performances of a theoretical order of teraflops (thousands of megaflops), still characterized by production costs that are extremely low compared to classical parallel systems. GPGPU adopts use of the GPU for operations different from graphics rendering, for which these devices were originally designed. This method has been particularly widespread in 2007 with the release of CUDA by Nvidia, who introduced software and hardware specialized for GPGPU computing. As Nvidia, other GPU manufacturers adapted their devices to this new methodology and have released software development environments for the realization of parallel programs. Hardware manufacturers have released APIs (Application Programming Interface) compatible only with their devices and this limited the development of portable software. However, in 2008 the standard OpenCL (Open Computing Language) was released for the implementation of parallel programs on heterogeneous systems. Gradually, all major manufacturers of GPU and CPU have released their implementation of OpenCL, providing developers an instrument capable of producing portable software on a large number of devices. Today we have reached the conclusion that hybrid systems based on CPU and GPU represent the future of supercomputing.

Among the different methodologies used for modelling processes, such as numerical analysis, high order difference approximations and finite differences, Cellular Automata (CA) ([26]) has proven to be particularly suitable when the behaviour of the system to be modelled can be described in terms of local interactions. Originally introduced by von Neumann in the 1950s to study self-reproduction issues, CA are discrete computational models widely utilized for modeling and simulating complex systems. Regarding the modeling of natural complex phenomena, Crisci and co-workers proposed a method based on an extended notion of homogeneous CA, firstly applied to the simulation of basaltic lava flow, which makes the modeling of spatially extended systems more straightforward and overcomes some unstated limits of the classical CA, such as having few states and look-up table transition functions. Mainly for this reason, the method is known as Complex Cellular Automata (CCA) (or Macroscopic Cellular Automata [9] or Multicomponent Cellular Automata [1]).

This paper presents an implementation of a well-known, reliable and efficient CCA model adopted for lava flow risk assessment, namely the SCIARA model [22], in GPGPU environments. Tests performed on two types of GPU hardware, a AMD Sapphire 280x graphic card and a Tesla K40c computing processor, and by adopting difference implementation strategies, have shown the validity of this kind of approach.

In the following sections, after a brief description of the basic version of the SCIARA CCA model for lava flows, a quick overview of GPGPU paradigm together OpenCL is presented. Subsequently, the specific model implementation

and performance analysis referred to benchmark simulations and of a real event are shown, while conclusions and possible outlooks are reported at the end of the paper.

2 Cellular Automata and the SCIARA lava flow simulation model

As previously stated, CA are dynamical systems, discrete in space and time. They can be thought as a regular *n*-dimensional lattice of sites or, equivalently, as an *n*-dimensional space (called cellular space) partitioned in cells of uniform size (e.g. square or hexagonal for *n*=2), each one embedding an identical finite automaton. The cell state changes by means of the finite automaton transition function, which defines local rules of evolution for the system, and is applied to *each* cell of the CA space at discrete time steps. The states of neighbouring cells (which usually includes the central cell) constitute the cell input. The CA initial configuration is defined by the finite automata states at time *t*=0. The global behaviour of the system emerges, step by step, as a consequence of the simultaneous application of the transition function to each cell of the cellular space.

When dealing with the modelling of spatial extended dynamical systems, CCA can represent a valid choice especially if their dynamics can be described in terms of local interaction at macroscopic level. Examples of successful applications of CCA include the simulation of lava flows [6], debris flows [16], density currents [20], water flux in unsaturated soils [17], soil erosion by rainfall [17] as well as pyroclastic flows [5], and forest fires [25].

For the OpenCL parallelization of the CA, the release fv2 of the SCIARA numerical model for simulating lava flows was adopted. SCIARA is a family of bi-dimensional CCA lava flow models, successfully applied to the simulation of many real cases such as the 2001 Mt. Etna (Italy) Nicolosi lava flow [6] and the 1991 Valle del Bove (Italy) lava event [2], which occurred on the same volcano and was employed for risk mitigation. In formal terms, the SCIARA-fv2 model [22] is defined as:

$$SCIARA\text{-}fv2 = < R, L, X, Q, P, \tau, \gamma >$$

where:

- R is the set of square cells covering the bi-dimensional finite region where the phenomenon evolves;
- $L \in R$ specifies the lava source cells (i.e. craters);
- X = {(0, 0), (0, 1), (-1, 0), (1, 0), (0, -1), (-1, 1), (-1,-1), (1, -1), (1, 1)} identifies the pattern of cells (Moore neighbourhood) that influence the cell state change; in the following we will refer to cells by indexes 0 (for the central cell) through 8;
- $Q = Q_z \times Q_h \times Q_T \times Q_f^8$ is the finite set of states, considered as Cartesian product of "substates". Their meanings are: cell altitude a.s.l., cell lava thickness, cell lava temperature, and lava thickness outflows (from the central cell toward the eight adjacent cells), respectively;
- $P=\{w, t, T_{sol}, T_{vent}, r_{Tsol}, r_{Tvent}, hc_{Tsol}, hc_{Tvent}, \delta, \rho, \varepsilon, \sigma, c_v\}$ is the finite set of parameters (invariant in time and space) which affect the transition function (please refer to [22] for their specifications);
- $\tau : Q^9 \rightarrow Q$ is the cell deterministic transition function, divided in *elementary processe*s and applied to each cell at each time step, which describes the dynamics of lava flows, such as cooling, solidification and lava outflows from the central cell towards neighbouring ones. In particular, In the fv2 version of SCIARA, the so called *elementary processes* [9] describing the cell's transition function are: (i) σ1, which determines lava outflows based on an opportune version of the Minimisation Algorithm of Differences; (ii) σ2, which determines lava thickness computation; (iii) σ3, which determines lava temperature and (iv) σ4, which determines the eventual lava solidification.
- $\gamma : Q_h \times \mathrm{N} \rightarrow Q_h$ specifies the emitted lava thickness from the source cells at each step $k \in N$ (N is the set of natural numbers).

3 OpenCL and GPGPU programming

In recent years, mainly due to the stimulus given by the increasingly demanding performance of gaming and graphics applications in general, graphic cards have undergone a huge technological evolution, giving rise to highly parallel devices, characterized by a multithreaded and multicore architecture and with very fast and large memories. A GPU can be seen as a computing device that is capable of executing an elevated number of independent threads in parallel. In general, a GPU consists in a number (e.g., 16) of SIMD (Single Instruction, Multiple Data) multiprocessors (or compute units) with a limited number of floating-point processors that access a common shared-memory within the multiprocessor.

OpenCL [21] is a framework that allows the user to perform tasks *both* on GPU than on CPU. The OpenCL routines can be performed on the GPU or CPU which are produced by major parallel computing brands, such as AMD, Nvidia, and Intel. Specifically OpenCL is nonproprietary, because it is based on a public standard, and can be freely downloaded.

The goal of OpenCL is thus to unify the programming model software to run the code on heterogeneous devices. In fact, today OpenCL supports different platforms that include CPUs (e.g. Intel, AMD, ARM, etc), GPUs (e.g., AMD, Intel, Nvidia), besides FPGA and DSP (Digital Signal Processors). As known, in Parallel Computing developers can create and manipulate concurrent task. When developers need to program a solution in OpenCL, they must decompose the problem in different tasks. Parallel programming assigns computational task to multiple processing elements that are executed at the same time. In the OpenCL language, these tasks are called *kernels*. A kernel is a special function written in C99 that is intended to be performed by one or more OpenCL devices. The kernels are sent to the devices through the host program. The host program is written in C / C ++

and runs on the user's development system. The host application manages the connected devices using a container called *context*. To create a kernel, the host selects a function from a container called *program*. Subsequently, it associates the kernel with its data and sends it to a structure called *command queue*. The command queue is the mechanism by which the host tells devices what to do and subsequently, when a kernel is queued, the device will perform the corresponding function. An OpenCL application can configure different devices to perform different tasks, and each task can operate on different data. OpenCL provides thus a full task-parallelism. Figure 1 shows the kernel distribution among OpenCL-compliant devices.

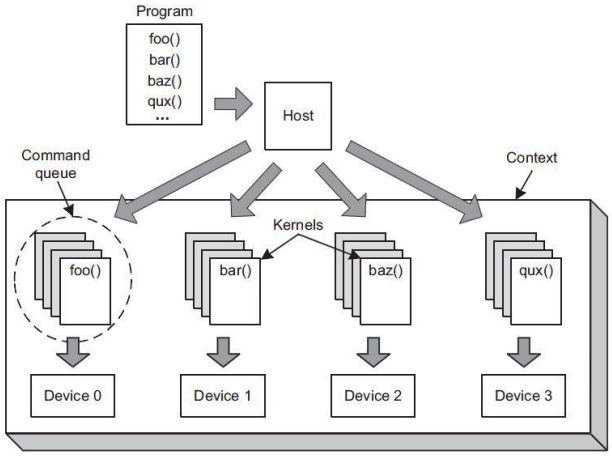


Figure 1: Assignment of the kernel to the devices contained in the *context* structure (figure taken from [21]).

Thus, in order to create an OpenCL application it is necessary to:

- Create a host program to manage the available devices and assign them the kernels to be performed;
- Create kernels, i.e. the routines to be performed on the selected devices from the host program.

3.1 Creation of the host program

The host program of an OpenCL application is written in C/C ++, but there are libraries created by third parties that allow to develop an application using the java and python languages [21]. The library defines six essential structures for the creation of the program host: *platform*, *device*, *context*, *program*, *kernel*, and *command queue*.

To access the computing devices on the system, OpenCL defines three structures: *platform*, *device* and *context*. Every manufacturer that supports OpenCL releases an SDK (software development kit) which contains an implementation of OpenCL compatible with its devices. The structure *platform* provides access to the OpenCL implementations installed on the system and to use all devices of the manufacturer. For example, installing a Nvidia SDK, all Nvidia devices on the system can be accessed via its platform. Devices are represented by the *device* structure and to be used they must be inserted into a container called *context*. In the host program, several context instances containing more devices can be defined. However, devices belonging to different contexts cannot communicate with each other and cannot be inserted in the same context devices belonging to different platform (for example, one can not create a context containing a AMD and a Nvidia device).

3.2 Kernel assignment and execution

The structure that allows communication between the host program and OpenCL devices is the *command queue*. Through the command queue, not only a kernel is assigned to a device, but can also perform data transfer operations, between two devices or between a device and the host program. Moreover, thanks to this structure one can carry out synchronizations between different kernels and profiling operations. To assign a kernel to a device one needs to decide how the data should be partitioned and assigned to compute units. Depending on the partitioning chosen by the user, kernel instantiations called *work-items* are carried out. Each work-item (i.e., thread), represents an execution of the same kernel but on different portions of data (i.e., in a SIMD fashion). For the assignment of kernels, OpenCL provides two functions:

`clEnqueueTask`. The task assigned by the host program to the device will run as a single work-item.

`clEnqueueNDRangeKernel`. The task assigned by the host program to the device will be split into multiple work items that will be executed in parallel.

The host program must therefore define the number of work items to be used and optionally may decide to divide the work items in groups called *work-groups*. The work items contained in a work-group have a shared memory block (local memory) which permits to access data much faster than the global memory shared by all the work items. Furthermore, the work items within a work-group can be synchronized. As expected, the latter of the two functions is fundamental, as it allows to perform tasks in parallel.

3.3 Data Transfer

Generally, the execution of a task includes the processing of data. From the moment the host program assigns a kernel to a device, it is necessary that the device has the data that is used to run the kernel. To send data to the device the function `clSetKernelArg` is used to allow the association of a data set to an argument of the kernel function. Basic data types that can be associated to the kernel are:

Pointers to primitive data types Associates a given primitive type.

Pointers to buffer object Associates a large set of data. A buffer object (represented by the structure `cl_mem`) can be

created using the function `clCreateBuffer`, but in order that data transfer takes place correctly, data must be stored on the host program contiguously.
After a buffer object is associated to a kernel, it is possible to reuse the same structures to transfer data both between two devices, between a device and the host program, etc.

3.4 Memory hierarchy

As reported before, a kernel function can be associated to the data required for processing. The host program is responsible for transferring the data to the device. Each device has different memory spaces (cf. Figure 2) in which to store the data received from the host program:

Global memory. Stores data accessible by all the work items for both reading and writing.

Constant memory. Similar to the global memory but data can be accessed in read-only.

Local memory. Stores data accessed by the work items contained in the same work-group.

Private memory. Stores data accessible by a single work-item.

All data from the host program is initially stored in the global/constant or private memory (the local memory can be allocated only by the host program but not initialized). The global/constant memory is larger than the others, but access to it is slower. Work items can indeed access the local memory much faster (100×) than that in the global/constant memory. Access to private memory is faster but its dimension is very small. With regards to constant memory, some devices have an apposite portion of memory, in other cases the constant memory space coincides with that of the global memory. To specify the memory space in which a given data must be stored the qualifiers `__global`, `__constant`, `__local`, `__private` are used. If omitted, data will be stored in private memory.

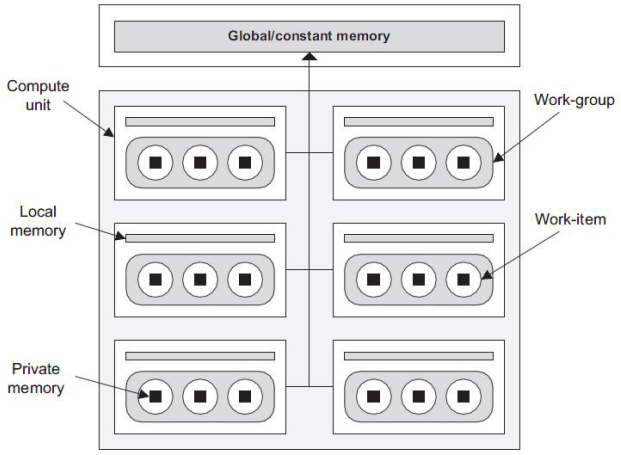


Figure 2: The OpenCL Memory model (figure taken from [21]).

3.5 Data Partitioning

The function clEnqueueNDRangeKernel described in subparagraph 3.3 allows to perform a task in parallel. To use this feature, one must:

- define on how many dimensions data is distributed (a two-dimensional matrix, etc.);
- define the number of work items for each dimension;
- define the number of work items in a work-group for each dimension.

To perform a task in parallel each work-item must be able to access the data portion that has been assigned to it. To each work-item is associated an ID that distinguishes it from all others, and generally these IDs are used to partition the data in a typical SPMD fashion. For example, suppose that the data consist of an array of `n` elements and also to have `n` work-items with their relative ID. Data can be partitioned by associating each work item to the array element with index corresponding to the ID. Moreover, for the work-items an ID that identifies them in a work-group is also associated. In this case, the purpose is to give the possibility to partition the data, even if here the partitioning occurs within a work-group. Other information that is accessible to the work items for the partitioning of data are:

• the total number of work items for each dimension;
• the total number of work-item contained in a work-group for each dimension;
• the total number of work-group
• the ID of the work-group to which the work-item belongs.

4 Implementation of the Sciara model

As previously stated, CA models, such as SCIARA, can be straightforwardly implemented on parallel computers due to their underlying parallel nature. In fact, since CA methods require only next neighbor interaction, they are very suitable and can be efficiently implemented even on GPUs. In literature, to our knowledge, few examples of Complex Cellular Automata modeling with GPUs are found, while some interesting CA-like implementations, such as Lattice Boltzmann kernels, are more frequent (e.g., [24], [15]).

The approach here adopted resembles many approaches in the field: typically, the CA parallel implementation involves two memory regions, which will be called `CAcurrent` and `CAupdated`, representing the current and next states for the cells respectively. For each CA step, the neighbouring values from `CAcurrent` are read by the local transition function, which performs its computation and writes the new state value into the appropriate element of `CAupdated`.

In accordance to the recent literature in the field (e.g., [3], [10]), in the GPGPU parallel implementation of the model, most of the automaton data (i.e. both the `CAcurrent` and `CAupdated` memory areas) was stored in the GPU global memory. In addition, the initialization of the CA (`CAupdated`) implies a copy from CPU to GPU. At the end of the computation, results from the device are copied back to the host through a GPU to CPU data transfer. In

addition, at each step, in order to update the status of the previous step with the current one, a copy between the two CA data buffer memory areas on the device takes place.

A crucial step is to identify the set of instructions (i.e. the elementary processes of the transition function) that can be performed independently on the cells of the CA space. The instructions will be invoked in parallel using a OpenCL kernel for each of elementary process. Note that at each step of the simulation, only a few cells of the entire cellular space are involved in the computation. Thus, a typical problem related to GPGPU parallelization (as reported later) which can affect the speedup of the model, can lead to an overuse of computationally inactive work-items.

In particular, we will describe two different strategies that were adopted for an efficient parallelization of the SCIARA-fv2 simulation model. The first of these, defined as Whole Space Strategy (WS), is based on a naive approach to the problem consisting in the use of only global memory which is shared by the totality of work-items that make up the mapping grid. The second version, called Active Cells Strategy (AC), has a significant performance improvement of the algorithm, achieved thanks to the adoption of a data structure that manages the CA computationally active cells. In this strategy the computation takes place within a grid of work-items that adapts dynamically to the active cells.

4.1 Naïve implementation

The first strategy for the parallelization of the SCIARA-fv2 model is based on a one work-item - one cell approach, where each cell in the cellular space is computed by OpenCL work-item. The Whole Space (WS) strategy version involves the use of global only memory, where each kernel runs on a grid of work-items divided into work-groups and mapped on the entire cellular space. Work items are thus executed in parallel and synchronized each time an elementary process ends. Importantly, the elementary processes must be defined in such a way that the work items are executed independently from each other and that each work-item accesses exclusively to the portion of data that it has been assigned, since OpenCL does not provide mechanisms for synchronization of work-item belonging to different work-group [21]. The host program assigns the kernel to devices by sending them the CA model data (sub-states, type of neighborhood, size of the space cell, etc.). The execution cycle is then managed by the host program, while the transition function is performed on the devices.

The following excerpt shows execution cycle:

```
steps = 0;
wiNum = ROWS * COLS;
while (steps < maxStep) { // CA steps
    // kernel execution for each elementary process
    clEnqueueNDRangeKernel(queue,updateVentsEmissi
on, dimNum, NULL, wiNum, NULL, 0, NULL, NULL);
    // update substates
    clEnqueueCopyBuffer(queue,           CAupdated,
CAcurrent, 0, 0, bufDim, 0, NULL, NULL);

    clEnqueueNDRangeKernel(queue,   empiricalFlows,
dimNum, NULL, wiNum, NULL, 0, NULL, NULL);
    clEnqueueCopyBuffer(queue,           CAupdated,
CAcurrent, 0, 0, bufDim, 0, NULL, NULL);

    clEnqueueNDRangeKernel(queue,     width_update,
dimNum, NULL, wiNum, NULL, 0, NULL, NULL);
    clEnqueueCopyBuffer(queue,           CAupdated,
CAcurrent, 0, 0, bufDim, 0, NULL, NULL);

    clEnqueueNDRangeKernel(queue,
updateTemperature, dimNum, NULL, wiNum, NULL, 0,
NULL, NULL);
    clEnqueueCopyBuffer(queue,           CAupdated,
CAcurrent, 0, 0, bufDim, 0, NULL, NULL);

    steps++;
}
```

As an example, the following excerpt reports the kernel definition for the `empiricalFlows` lava outflow computation elementary process (i.e., $\sigma 1$).

```
__kernel void empiricalFlows(__global double *
SUBSTATES, Parameters parameters) {

int i = get_global_id(0);
int j = get_global_id(1);
int SLT = 2; //lava thickness substate index
int F = 3; //outflows substates index

// check if cell contains lava
if (SUBSTATES[ROWS*COLS*SLT +(i*ROWS + j)] > 0) {
    double outflows[MOORE_NEIGHBORS];
    outflowsMin(SUBSTATES,    i,    j,    outflows,
parameters); //minimization algorithm application

    // update outflows substate
    for (int k = 1; k < MOORE_NEIGHBORS; k++)
        if (outflows[k] > 0)
          SUBSTATES[ROWS*COLS*(F+k-1) +(i*ROWS + j)]
          = outflows[k];
    }
}
```

While a similar straightforward strategy has proven to be effective in other parallelizations and applications (e.g. [14], [8], [7]), the speedups here achieved were not quite positive, probably due to the excessive use of computationally inactive threads and overuse of global memory. At the contrary, the following approach has given more positive results and can be considered as a starting point for more sophisticated applications.

4.2 Active cells optimization

The *active cells optimization strategy* (AS) allows to apply the transition function only on the *active* cells, omitting those that are in a quiescent state. The active cells are contained within a list and elementary processes can add to the list new cells, or remove them. An array designed to contain all active cells is initially allocated. Specifically, for each active cell, a work-item is instantiated and can add new cells to the list, or remove them. However, to maintain the list in a consistent state, accesses must be performed by the work-item in an exclusive manner. Unfortunately, OpenCL doesn't support exclusive accesses to global data, so an alternative approach was here adopted. In particular, each cell is associated with a Boolean value that indicates whether the cell is active or inactive. The work-item, in order to add or remove a cell, must change this value. This information is

used by a *stream compaction* algorithm to build the list of active cells.

Stream compaction algorithms ([4], [18]) are used to remove unwanted items in a set of scattered data. In this case, data is located in an array of Boolean values whose elements correspond to the cells of the cellular space. Items with a true value correspond to the active cells, while false ones correspond to the inactive cells. The implemented stream compaction algorithm, adapted from [13] takes as input this array and computes an array containing only the active cells. Suppose we have `p` work-items and a Boolean array with `n` elements, with `n> p`. The array is divided into `p` equal parts. The algorithm consists of three phases:

1. Each work-item counts the number of active cells in its part of the array.
2. Each work-item computes the array index of the output array from which it can start entering the active cells contained in the portion of its array.
3. Each work-item enters the active cells of its portion in the output array starting from the offset calculated in previous step.

Figure 3 shows the functioning of the algorithm for a matrix representing a CA space size of 4×4 with 6 active cells highlighted in red. The array used to track active cells is given as input to the stream compaction algorithm that outputs the list with only the active cells.

The second stage uses a two phase algorithm called *prefix sum*. Specifically, the algorithm takes as input the array calculated in the previous step containing the sum of the active cells calculated by each work-item. The array is seen as a balanced tree where its elements are the nodes of the tree. In the first phase, the tree is crossed from the leaves to the root calculating, for each level of the tree, the partial sums of the nodes of the previous level (by a parallel reduction pattern). In second phase, the tree is traversed from the root (containing the total number of active cells) to the leaves. At each iteration, each node sets the value of the right child to the sum of its value and the value of the left child. In addition, each node sets the value of its left child to its value. At the end of this phase, an array is created that specifies, at each location, the position from which each work-item can write its active cells in the global active cells output array.

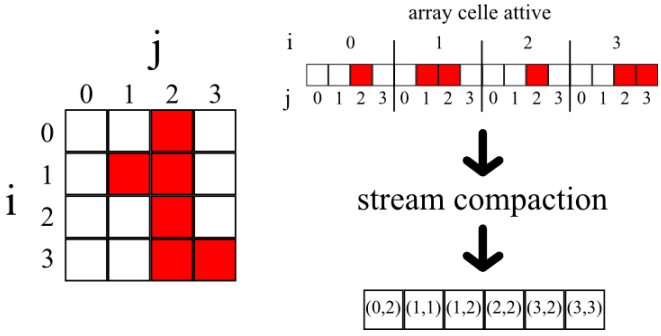


Figure 3: Stream compaction algorithm example referred to a 4 x 4 CA.

5 TESTS AND PERFORMANCE RESULTS

Two graphic devices were adopted for experiments: a NVIDIA high-end Tesla K40c and a AMD Sapphire 280x graphic card, both with a theoretical peak performance about 3,5 GFLOPS. In particular, the Tesla computing processor has 2880 stream processors (i.e., CUDA cores) and 15 compute units, 12 GB global memory and high-bandwidth communication between CPU and GPU, whereas the AMD graphic card has 2048 stream processors, 32 compute units and 3 GB global memory. The sequential SCIARA-fv2 reference version was implemented on a 2.8 GHz Intel Quad-core Xeon based desktop computer. The sequential CPU version is identical to the versions that were developed for the GPUs, that is, at every step, the CA space array is scrolled and the transition function applied to each cell of the CA where lava is present.

A first test regarded the simulation of well-known and documented real lava flow event, the Mt. Etna Nicolosi event [6] occurred in July, 2001. The simulation was carried out for 10000 steps, considering two craters for lava flow emission. In order to further stress the efficiency of the GPU version, further benchmarks experiments (stress test) were performed by considering 200 lava sources equally spaced over the area. Moreover, the two parallelizations (i.e., Whole Space and Active Cells strategies) reported in section 4 were considered. Table 1 reports the results of tests carried out for experiments, where the CA space is a 517 × 378 two-dimensional grid.

From all performed experiments, we can note that in all cases the versions using the active cells optimization are more efficient. As expected, this is due to the fact that the optimization does not apply the transition function to cells which are in a quiescent state, and thus many unnecessary calculations are not executed. Instead, in the standard implementation, there is an excessive use of computationally inactive threads and overuse of global memory.

For the first series of tests, which takes into account only two sources lava, we can see that the WS parallel version reaches a speedup of 22× (23× for the Tesla K40c) compared to its sequential version. In the case of the parallel version with active cells (AS) optimization, performance reaches a maximum speedup of only 2× (3× for the Tesla K40c), with respect to its corresponding sequential active cells version. This is due to the fact that with only two sources of lava the active cells are relatively few and therefore also the sequential implementation results extremely efficient. Moreover, performance of the AMD Sapphire 280x GPU is unexpectedly comparable to that of the Nvidia GPU K40c for the standard WS version, probably due the high boost clock of the AMD hardware with respect to the lower GPU clock of the Nvidia card (1000 MHz vs 745 MHz). However, performances related to the optimized AS version in terms of execution times are better for the Nvidia hardware, probably due to the higher number of

streaming processors (i.e., 2880 vs 2048) that can be fully exploited.

As for the stress test, we can observe a speed up of 41× (40× for the Tesla K40c) for the parallel version that is not optimized, and a speedup of 39× (41× for the Tesla K40c) for the parallel version with active cells optimization compared to the corresponding sequential versions. Here, we can see that the speedup of the versions with active cells optimization is comparable with the versions without optimizations, due to the relative equal (and elevated) number of cells involved in the stress test. Even in this case, both considered hardware give the same performance, as for the previous test. A likely explanation is due to the fact that on one hand the Nvidia hardware has a higher number of streaming processors while, on the other, the AMD card has more compute units, which basically compensate both hardware computing capabilities.

Table 1: Execution times of experiments (in seconds) carried out for evaluating the performance the GPU version of the SCIARA CCA lava-flow model on the considered hardware. Experiments refer to the Whole Space (WS).

CA dim / Device	Xeon (sequential)	K40c (WS)	Sapphire (WS)
517 ×378 (2 craters)	1122	49	52
517 ×378 (200 craters – Stress Test)	2790	69	68

Even if timings achieved for the single case simulation cannot be considered positive, the stress test experiments have revealed the full suitability of the parallel system for intensive computations like applications, such as for the construction of hazard maps (e.g., [7], [16]). Typically, the most general approach for computing a hazard map in a extended area consists of a Monte Carlo approach in which a high number (e.g. thousands) of different simulations are carried out and on geological-geomorphological field survey and statistical analysis.

Table 2: Execution times of experiments (in seconds) carried out for evaluating the performance the GPU version of the SCIARA CCA lava-flow model on the considered hardware. Experiments refer to the Active Cells (AC) strategies (see text).

CA dim / Device	Xeon (sequential)	K40c (AC)	Sapphire (AC)
517 ×378 (2 craters)	62	22	30
517 ×378 (200 craters – Stress Test)	2005	49	51

Eventually, to test if single-precision data can be considered sufficient for SCIARA simulations, tests were carried out on the 2001 lava flow event (10000 CA steps) and compared results produced by the GPU version with those produced by the CPU (sequential) version with double precision CPU implementation (i.e., `double` type variables). Comparison results were satisfactory, since the areal extensions of simulations resulted the same, except for few errors of approximation in a limited number of cells. In particular, comparing the GPU version with the CPU single-precision version approximation differences at the third significant digit were only for 4% of cells, while differences were less for remaining cells. Differences were even minor compared to the previous case by considering the single precision GPU version and a CPU version which adopts double-precision variables.

6 CONCLUSIONS

This paper reports the implementation of a Complex Cellular Automata model using GPU architectures. As shown, the OpenCL technology, in combination with the an efficient memory management, can produce a very efficient version of the SCIARA-fv2 lava flow simulator. Several tests were carried out to evaluate the implemented parallelizations. In particular, tests using the GPU Sapphire 280x show that the parallel version, without optimizations, achieved a 41× speedup compared to its sequential version. The parallel version with optimization active cells also reached a speedup of 41× compared to its corresponding sequential one. As expected, in all cases, versions using the active cells optimization have resulted to be more efficient than versions without the optimization.

Future work will also regard the exploitation of graphic hardware for the construction of hazard maps, such as in [16] which are fundamental for determining locations that are subject to future events and their related risk. The positive performances obtained for the more intensive computations (stress test) will imply the extension of the AC strategy in a multi-simulation context, by using OpenCL to accelerate simultaneous concurrent SCIARA-fv2 lava flows simulations.

The results obtained on the SCIARA model are therefore to be considered positive, but further testing should be performed to assess the functionality of the adopted strategies on other models and their ability to fruitfully exploit parallel systems resources.

Acknowledgments - This research was partially funded by the UE POR FSE CALABRIA PIA 2010 "Laboratorio in Campo" Project DDG n. 17198. Authors also gratefully acknowledge the support of NVIDIA Corporation for this research.

REFERENCES

[1] M.V. Avolio, S. Di Gregorio, W. Spataro, G.A. Trunfio. "A theorem about the algorithm of minimization of differences for multicomponent cellular automata". In: Sirakoulis GC, Bandini S, editors, ACRI. Springer, volume 7495 of Lecture Notes in Computer Science, 289-298, 2012.

[2] D. Barca, G.M. Crisci, Di Gregorio, S., Nicoletta, F. "Cellular Automata for simulating lava Flows: A method

and examples of the Etnean eruptions". Transport Theory and Statistical Physics, 23, 195-232, 1994.

[3] G. Bilotta, E. Rustico, A. Hérault, A. Vicari, G. Russo, C. Del Negro, G. Gallo. "Porting and optimizing MAGFLOW on CUDA", Annals of Geophysics 5 (54), 2011.

[4] M. Billeter, O. Olsson, U. Assarsson U. "Efficient stream compaction on wide SIMD many-core architectures". In: Proceedings of the Conference on High Performance Graphics 2009. ACM, 159–166, 2009.

[5] G.M. Crisci, S. Di Gregorio, R. Rongo, W. Spataro. "PYR: a Cellular Automata model for pyroclastic flows and application to the 1991 Mt. Pinatubo eruption". Future Generation Computer Systems 21 (7), 1019-1032, 2005.

[6] G.M. Crisci, S. Di Gregorio, R. Rongo, W. Spataro. "The simulation model SCIARA: the 1991 and 2001 at Mount Etna". Journal of Vulcanology and Geothermal Research, 132, 253-267, 2004.

[7] D. D'Ambrosio, G. Filippone, D. Marocco, R. Rongo, W. Spataro. "Efficient application of GPGPU for lava flow hazard mapping". The Journal of Supercomputing, vol. 65, no. 2, 630–644, 2013.

[8] M. De La AsunciòN, J.M. Mantas, M.J. Castro, E. D. Fernàndez-Nieto. "A MPI-CUDA implementation of an improved Roe method for two-layer shallow water systems". Journal of Parallel and Distributed Computing, 72, 9, 1065–1072, 2012.

[9] S. Di Gregorio, R. Serra. "An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata". Fut. Gener. Comp. Syst., 16, 259–271, 1999.

[10] S. Di Gregorio, G. Filippone, W. Spataro, G.A. Trunfio. "Accelerating wildfire susceptibility mapping through GPGPU". Journal of Parallel and Distributed Computing 73: 1183 – 1194, 2013.

[11] M. Domínguez Jose, J.C. Crespo Alejandro, Gómez-Gesteira Moncho. "Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method". Computer Physics Communications, 184, 3, 617-627, 2013.

[12] A. Grama, G. Karypis, V. Kumar, A Gupta. "An Introduction to Parallel Computing: Design and Analysis of Algorithms", Second Edition. USA: Addison Wesley, 2003.

[13] M. Harris, S. Sengupta, J.D Owens. "Parallel prefix sum (scan) with CUDA", GPU Gems 3 (39), 851-876, 2007.

[14] D. Jacobsen, J. C. Thibault, , I. Senocak. "An MPI-CUDA implementation for massively parallel incompressible flow computations on Multi-GPU clusters". In: American Institute of Aeronautics and Astronautics (AIAA) 48th Aerospace Science Meeting Proceedings, 2010.

[15] F. Kuznik, C. Obrecht, G. Rusaouen, J.J. Roux. "LBM based flow simulation using GPU computing processor". Computers and Mathematics with Applications, 59, 2380–2392, 2010.

[16] F. Lucà, D. DAmbrosio, G. Robustelli, R. Rongo, and W. Spataro. "Integrating geomorphology, statistic and numerical simulations for landslide invasion hazard scenarios mapping: an example in the Sorrento peninsula (italy)". Computers & Geosciences, vol. 67, 163–172, 2014.

[17] G. Mendicino, A. Senatore, G. Spezzano, S. Straface. "Three-dimensional unsaturated flow modeling using cellular automata". Water Resources Research, 42, 2006.

[18] H. Nguyen. Gpu Gems 3. First. Addison-Wesley Professional, 2007.

[19] NVIDIA CUDA C Programming Guide, 2011a. Available from: http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3a2J6b3gl [accessed May 2015]

[20] T. Salles, S. Lopez, M. Cacas, T. Mulder. "Cellular automata model of density currents". Geomorphology, 88: 1 – 20, 2007.

[21] M. Scarpino. OpenCL in action. Manning Publications Co., 2012.

[22] W. Spataro, M.V. Avolio, V. Lupiano, G.A. Trunfio, R. Rongo, D. D'Ambrosio. "The latest release of the lava flows simulation model sciara: First application to Mt Etna (Italy) and solution of the anisotropic flow direction problem on an ideal surface". Procedia Computer Science 1: 17-26, 2010.

[23] J.C. Thibault, I. Senocak. "Accelerating incompressible flow computations with a Pthreads-CUDA implementation on small-footprint multi-GPU platforms". The Journal of Supercomputing, 59, 2, 693 – 719, 2012.

[24] J. Tolke. "Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by NVIDIA". Comput. Vis. Sci., 13 1, 29–39, 2008.

[25] G.A. Trunfio, D. D'Ambrosio, R. Rongo, W. Spataro, S. Di Gregorio. "A new algorithm for simulating wildfire spread through cellular automata". ACM Transactions on Modeling and Computer Simulation (TOMACS), 22, 6, 2011.

[26] J. von Neumann (Edited and completed by A. Burks),. Theory of self-reproducing automata. USA: University of Illinois Press, 1966.

A Tool for Automatically Suggesting Source-Code Optimizations for Complex GPU Kernels

Saeed Taheri[1], Apan Qasem[2], and Martin Burtscher[2]
[1]Department of Computer Science, School of Computing, University of Utah, Salt Lake City, Utah
[2]Department of Computer Science, Texas State University, San Marcos, Texas

Abstract - *Future computing systems, from handhelds to supercomputers, will undoubtedly be more parallel and heterogeneous than today's systems to provide more performance and energy efficiency. Thus, GPUs are increasingly being used to accelerate general-purpose applications, including applications with data-dependent, irregular control flow and memory access patterns. However, the growing complexity, exposed memory hierarchy, incoherence, heterogeneity, and parallelism will make accelerator-based systems progressively more difficult to program. In the foreseeable future, the vast majority of programmers will no longer be able to extract additional performance or energy-savings from next-generation systems because the programming will be too difficult. Automatic performance analysis and optimization recommendation tools have the potential to avert this situation. They embody expert knowledge and make it available to software developers when needed. In this paper, we describe and evaluate such a tool. It quantifies performance characteristics of GPU code through profiling, employs machine learning models to estimate the suitability and benefit of several known source-code optimizations, ranks the optimizations, and suggests the most promising ones to the user if the expected speedup is sufficiently high.*

1. Introduction

There are two primary difficulties with using accelerators such as GPUs. First, they can only execute certain types of programs efficiently, in particular programs with enough parallelism, data reuse, and regularity in their control flow and memory access patterns. Second, it is harder to write effective software for accelerators than for CPUs because of architectural disparities such as wide parallelism, exposed memory hierarchies, lockstep execution, and memory access coalescing. Several new programming languages and extensions have been proposed to hide these aspects to various degrees and thus make it easier to program accelerators [1].

We study the alternative approach of making the programming and performance optimization easier for software developers who are not experts in GPU programming, specifically when it comes to complex irregular codes that are hard to parallelize. In particular, we describe a machine-learning-based recommendation tool for GPU kernels that automatically determines performance bottlenecks and suggests appropriate source-code optimizations, if any.

Several efficient GPU implementations of irregular algorithms have been published, showing that GPUs are capable of accelerating rather complex codes if they are implemented in a GPU-friendly fashion [2, 3, 4]. However, most software developers have no formal education in parallel programming, much less in accelerator programming, and could therefore greatly benefit from access to a performance/parallelism expert. Unfortunately, there are only relatively few such experts and each expert may only know a certain aspect or application domain. That raises the question of how to best deliver such expertise to programmers.

We believe the best solution to be automatic program analysis and recommendation tools. They embody the know-how of performance optimization experts and automatically determine where the bottlenecks lie and how to improve a given piece of code on a given system. Based on its analysis results, the tool recommends possible courses of action. Section 2 describes our tool in more detail.

Since the tool's recommendation accuracy hinges on how well it predicts the expected speedup of the optimizations in its database if they were applied to user-provided code, we evaluate it by comparing its predicted speedups with the actual speedups obtained when truly incorporating the suggested source-code optimizations. To make these comparisons possible, we wrote 64 versions each of two CUDA programs that include all possible combinations of six source-code optimizations and use different subset of these implementations to train and test our tool.

This paper makes the following contributions. 1) It describes how to build source-code recommendation tools that can automatically adapt to the underlying hardware and to changes in their optimization database. 2) We built such a tool for GPU programs and show that it delivers good recommendation accuracies on the platform and optimizations we tested, including on complex irregular CUDA code. 3) We study different scenarios to determine conditions that affect the tool's prediction accuracy.

The rest of this paper is organized as follows. Section 2 describes the design of our tool. Section 3 provides background information upon which the later sections are based. Section 4 discusses related tools and how they differ from our approach. Section 5 explains the experimental methodology. Section 6 presents the results and analyzes them. Section 7 concludes with a summary and future work.

2. Tool Design

Our tool employs a three-tiered design backed by an optimization database. The first tier performs code evaluation, the second tier analyzes the results, and the third tier handles the optimization selection. The tiers communicate through a simple interface. This makes it possible to design each tier independently and to easily replace any tier.

Tier 1 is concerned with evaluating code behavior and producing performance data. We refer to these data as *feature vectors*. They are produced using NVIDIA's Visual Profiler [5]. It can measure a large number of hardware performance counter events such as instruction counts, cache hits/misses at different levels, etc. We normalize these *features* by the cycle count to make them independent of the runtime. The normalized features are then combined into a feature vector.

It should be noted that our tool does not depend on any particular profile information. Rather, the accuracy of the recommendations simply improves with better profiling data. This makes the tool easy to port to platforms with different profilers or GPUs with other performance counters.

Before we discuss the second tier, it is important to explain the content of the optimization database. The database is an unordered set of independent entries, where each entry represents an optimization, including a description with an example that illustrates how to apply it as well as pairs of before and after code samples that do not and do include the optimization, respectively. Each code sample includes one or more inputs to run it with.

A key feature of this database is that each entry is independent, making it easy to delete unwanted entries, modify existing entries, and add new entries. Thus, anybody can contribute optimizations, in particular experts from different domains. This makes the database very flexible, simple to port, and customizable to include only optimizations for a specific domain or hardware component.

Tier 2 analyzes the feature vector obtained from profiling the user's application to determine the most appropriate optimizations. Before it can do so, it must train itself on the before and after code samples from the database. It does this upon installation or when the database is modified by running the code samples through Tier 1 to obtain before and after feature vectors. From these vectors, it learns to recognize when a given optimization is needed and how much benefit it can deliver on the target platform. Tier 2 employs the ML algorithms listed in Section 3.4 for this purpose.

Tier 3 collects the recommendations from the second tier and sorts them by expected benefit. It then outputs the top choices if their benefit is above a preset threshold. The user can select how many recommendations to maximally display, whether to include the explanations and/or examples in the output, etc. These user-interface aspects are relatively straightforward and not the focus of this paper.

3. Background

3.1 GPU Architecture

This subsection provides a brief overview of the architectural characteristics of the Kepler-based Tesla K20c compute GPU we use and explains some of the features that make GPUs difficult to program. GPU programs require hierarchical parallelization across threads as well as across thread blocks of up to 1024 threads. The K20c consists of 13 streaming multiprocessors (SMs) to which the thread blocks are mapped. Each SM contains 192 processing elements (PEs) for executing the threads. Whereas each PE can run an individual thread of instructions, sets of 32 PEs are tightly coupled and must either execute the same instruction (with different data) in the same cycle or wait. This is tantamount to a SIMD instruction that conditionally operates on 32-element vectors. The corresponding sets of 32 coupled threads are called warps. Warps in which not all threads can execute the same instruction are subdivided by the hardware into sets of threads such that all threads in a set execute the same instruction. The individual sets are serially executed, which is called branch divergence, until they re-converge. To maximize performance, branch divergence has to be minimized, but it is typically difficult to implement programs in a manner such that sets of 32 threads follow the same control flow.

The memory subsystem is also built for warp-based processing. If the threads in a warp simultaneously access words in main memory that lie in the same aligned 128-byte segment, the hardware merges the 32 reads or writes into one coalesced memory transaction, which is as fast as accessing a single word. Warps accessing multiple 128-byte segments result in correspondingly many individual memory transactions that are executed serially. Hence, uncoalesced accesses are slower, but it is in general hard to write programs in such a way that sets of 32 threads access words from the same 128-byte segment. Part of the main memory, called constant memory, can only be written by the CPU. GPU accesses to constant memory benefit from a special cache.

The PEs within an SM share a pool of threads called thread block, synchronization hardware, and a software-controlled data cache called shared memory. A warp can simultaneously access 32 words in shared memory as long as all words reside in different banks or all accesses within a bank request the same word. Barrier synchronization between the threads in an SM can take as little as a couple of cycles per warp. The SMs operate largely independently. They can only communicate through global memory (main memory in DRAM). The SMs support special instructions such as voting, where all threads in a warp compute a combined predicate (i.e., a reduction and broadcast operation), and rsqrtf, which quickly computes an approximation of one over square root. However, programmers may not be aware of such features, which can drastically boost the performance.

3.2 N-body Code and Barnes-Hut Algorithm

To obtain test cases for evaluating our tool, we created 128 different versions of two *n*-body simulation codes (64 each) [6]. The first code, called NB, is regular and has O(n^2) complexity. The second code, called BH, is irregular and has O(n log n) complexity. Both programs simulate the time evolution of a star cluster under gravitational forces for a given number of time steps. However, the underlying algorithm (see below) and the code base of the two implementations are completely different. n denotes the number of stars (aka bodies). Both of these codes have been written in such a way as there is essentially no execution taking place on the CPU.

The direct NB algorithm performs precise force calculations based on the O(n^2) pairs of bodies. Since identical computations have to be performed for all bodies, the implementation is very regular and maps well to GPUs. The force calculations are independent and can be performed in parallel. In each time step, the O(n^2) force calculation is followed by an O(n) integration where each body's position and velocity are updated based on the computed force. For the values of n we consider, the integration represents an insignificant fraction of the overall execution time.

```
bodySet = ...
foreach timestep do {                  // O(n log n) + ordered sequential
  bounding_box = new Bounding_Box();
  foreach Body b in bodySet {          // O(n) parallel reduction
    bounding_box.include(b);
  }
  octree = new Octree(bounding_box);
  foreach Body b in bodySet {          // O(n log n) top-down tree building
    octree.Insert(b);
  }
  cellList = octree.CellsByLevel();
  foreach Cell c in cellList {         // O(n) + ordered bottom-up traversal
    c.Summarize();
  }
  foreach Body b in bodySet {          // O(n log n) fully parallel
    b.ComputeForce(octree);
  }
  foreach Body b in bodySet {          // O(n) fully parallel
    b.Advance();
  }
}
```

Figure 1: Pseudo code of Barnes-Hut algorithm

The Barnes-Hut (BH) algorithm approximates the forces acting on each body [7]. It recursively partitions the volume around the n bodies into successively smaller cells and records the resulting spatial hierarchy in an octree (the 3D equivalent of a binary tree). Each cell summarizes information about the bodies it contains. For cells that are sufficiently far away from a given body, the BH algorithm only performs one force calculation with the cell instead of one force calculation with each body inside the cell, which lowers the time complexity to O(n log n). However, different parts of the octree have to be traversed to compute the force acting on different bodies, making the control flow and memory-access patterns quite irregular. The force calculation is by far the most time consuming operation in BH, which is why we only consider source-code optimizations that affect this kernel. We use the BH implementation from the LonestarGPU suite [8]. It encompasses the algorithmic steps shown in Figure 1, each of which is implemented using different CUDA kernels. Since this implementation is irregular, we believe it is a good candidate for testing our tool.

In summary, the NB code is relatively straightforward, has a high arithmetic intensity, regular control flow, and accesses memory in a strided fashion. In contrast, the BH code is quite complex (it repeatedly builds an unbalanced octree and performs various traversals on it), has a low arithmetic intensity, performs mostly pointer-chasing memory accesses, and has data-dependent control flow. Due to its lower time complexity, it is faster on a K20c GPU than the NB code when simulating more than about 15,000 stars.

3.3 Source-Code Optimizations

We modified our two test programs to make it possible to individually include or exclude all possible combinations of six source-code optimizations through conditional compilation, i.e., to produce 64 different versions of each programs. In particular, there are 32 versions of each program that do not and 32 that do include a particular source-code optimization. This enables us to create different subsets of these versions for training (providing before and after code samples), testing, and evaluating our tool.

For NB, we study the following six optimizations:

- CONST copies immutable kernel parameters (i.e., most of the parameters) into the GPU's constant memory rather than passing them every time a kernel is called, i.e., it lowers the calling overhead.
- FTZ is a compiler flag that allows the GPU's floating-point ALUs to flush denormal numbers to zero, which results in faster computations. While strictly speaking not a code optimization, the same effect can be achieved by using appropriate intrinsic functions in the source code.
- PEEL separates the innermost loop of the force calculation into two consecutive loops, one of which has a known iteration count and can therefore presumably be better optimized by the compiler. The second loop performs the remaining iterations.
- RSQRT calls the CUDA intrinsic "rsqrtf()" to quickly compute one over square root instead of using the slower but slightly more precise "1.0f / sqrtf()" expression.
- SHMEM employs blocking, i.e., it preloads chunks of data into the shared memory, operates on this data, and then moves on to the next chunk. This reduces the number of global memory accesses.
- UNROLL uses a pragma to request unrolling of the innermost loop(s). Unrolling often allows the compiler to schedule instructions better and to eliminate redundancies, thus improving performance.

For BH, we study the following six optimizations.

- FTZ is identical to its NB counterpart.
- RSQRT is also identical to its NB counterpart.

- SORT approximately sorts the bodies by spatial distance to minimize the tree prefix that needs to be traversed during the force calculation.
- VOLA copies some volatile variables into non-volatile variables and uses those in code regions where it is known (due to lockstep execution of threads in a warp) that no other thread can have updated the value. This optimization reduces memory accesses.
- VOTE employs thread voting instead of a shared-memory-based code to perform reductions.
- WARP switches from a thread- to a warp-based implementation that is more efficient because it does not suffer from branch divergence and uses less memory as it records certain information per warp instead of per thread.

3.4 Machine Learning Algorithms

We utilize various subsets of the feature vectors from our test programs to train the Machine Learning (ML) methods in our tool such that they can learn how much speedup an optimization might provide under different conditions. The goal is to be able to predict by how much each of the optimizations in the database will improve or hurt the performance of a given CUDA kernel. Based on these predictions, the tool selects which optimizations to suggest.

Machine learning approaches generally use data attributes as features to perform classification/prediction. Each data entry can be viewed as a point in N-dimensional space, where N is the number of attributes per data item. This allows, for example, to place each training data point into an N-dimensional space so that any test data point can be classified based on "nearby" training data points.

We examined three different ML approaches: linear and logistic regression, instance-based learners, and decision trees. Regression is concerned with modeling the relationship between variables that is iteratively refined using a measure of error in the predictions made by the model. Regression methods are important in statistics and have been cooped into statistical machine learning.

The instance-based learning model is a decision problem with instances or examples of training data that are deemed important to or required by the model. Such methods typically build a database of examples and compare new data to the database using a similarity measure to find the best match and make a prediction. The focus is on the representation of the stored instances and the similarity measures used between instances. In our experiments we use IBK, which is an instance-based classifier that uses the k-nearest neighbor (KNN) method for classification. During training, all labelled instances are recorded. When invoked on a new test instance, the model attempts to find the k recorded instances that are most similar to the given test instance. Similarity is measured by the Euclidean distance between the feature vectors of the test and training instances. The mode value of the label for the k nearest neighbors is used to predict the outcome. Although we experimented with several different values of k, the results presented in this paper all use $k = 10$, which proved to be most effective.

Decision tree methods construct a model of decisions made based on the values of the attributes in the data. Decisions fork at each level in the tree until a leaf node is reached, where a prediction decision is made based on the training cases that reached the same leaf node. Decision trees are trained on data for classification and regression problems [9]. We employ M5P, a special type of decision-tree where each leaf node is a linear regression model. This model utilizes the M5 technique proposed by Quinlan [10]. First, an induction algorithm is used to construct a standard decision tree. Then a multivariate regression model is constructed for each node in the tree. However, instead of using all features in the regression model, only the features that appear in the subtree that contains the node are used. Finally, the leaf nodes are replaced by the newly constructed regression models. Once this regression-based decision tree has been built, standard pruning and smoothing techniques are applied.

4. Related Work

Paradyn [11] is one of the first tools for automatic performance analysis. It uses dynamic instrumentation to efficiently obtain performance profiles of unmodified executables. KOJAK [12], Scalasca [13, 14], Vampir [15] and VampirTrace [16] are trace-based tools that support MPI, OpenMP, and hybrid codes. For instance, the highly scalable Scalasca tool employs TAU's rich instrumentation capabilities [17] and processes the trace data in parallel. It scores and summarizes the trace report and shows it on a GUI.

Periscope [18] evaluates the performance while an application is running and searches for previously specified performance problems or properties. It is MPI-based and focused on efficient communication between cores/processors. TAU [17] is a portable tool for performance instrumentation, measurement, analysis, and visualization of large-scale parallel applications. Using the library wrapping benefit of TAU, TAUCuda [19] can measure GPU performance. It requires no modification of the source or binary code. The recently released Score-P tool [20] represents a portable infrastructure for performance measurement tools. Each of the above tools utilizes a different measurement output format. For example, the output format Vampir is OTF and the output format of Scalsca is EPILOG/CUBE. Score-P tries to integrate all of these tools into a unified measurement infrastructure. HPCToolkit [21, 22] generates statistical profiles using interval timers and hardware-counter interrupts and evaluates both application binaries and source code.

NVIDIA created tools such as the CUDA Performance Tools Interface (CUPTI) [23], Visual Profiler [24], and Nsight [25] that focus on GPU performance bottlenecks.

Some tools, such as PAPI CUDA [26] and VTune Amplifier XE [27], use hardware counters to measure the performance. eeClust [28] determines relationships between the

behavior of parallel programs and the energy consumption.

Virtual Institute - High Productivity Supercomputing (VI-HPS) [29] is a collaboration of several partner institutions for improving the quality and accelerating the development process of complex simulation codes in science and engineering that are being designed to run on highly-parallel computer systems. Many well-known tools for parallel performance and measurement such as TAU, Scalasca and Vampir are designed and created by the partners of this big project. They also have a couple of ongoing and completed projects in the field of productivity and performance to improve their previous products. POINT, Score-P, SILC, HOPSA, PRIMA and LMAC are tools for integrating and improving the functionality of performance and measurement tools such as TAU and Vampir. For instance, LMAC adds the functionality of automatically examining performance dynamic for irregular behavior of parallel simulation codes to the established performance analysis tools Vampir, Scalasca, and Periscope.

Machine learning methods have also been used in MILEPOST GCC [30], a self-optimizing compiler that automatically learns the best optimization heuristics based on the behavior of the platform. There are also model-driven autotuning tools that are based on regression trees [31].

PerfExpert [32] is a tool that combines a simple user interface with an analysis engine to detect probable core-, socket-, and node-level performance bottlenecks in each important procedure and loop of a CPU application. For each bottleneck, PerfExpert provides a concise performance assessment. Unlike most of the tools described above, PerfExpert suggests steps that can be taken by the programmer to improve performance. In particular, its AutoSCOPE backend provides automatic recommendations for performance tuning, including compiler switches and optimization strategies with source-code examples [33]. It determines which suggestions to make by searching a manually annotated database of optimizations for the closest matches to PerfExpert's output metrics, which are derived from performance-counter measurements.

Our tool is most similar to that of PerfExpert/AutoSCOPE. We also use profiling based on hardware performance counters and compute derived metrics that are then used to identify suitable optimizations to recommend. However, instead of CPU procedures, we target complex GPU kernels, which can be challenging to make efficient. More importantly, instead of hand-annotating optimizations, which is tedious, error prone, and not very portable, our approach automates this step using ML algorithms that are trained using sample codes for each optimization. This not only makes it easy to port our tool to other systems but also enables the tool to automatically adapt the recommendations it makes to the performance characteristics of each system. Moreover, it provides the ability to alter the recommendation database without having to worry about how this change interacts with the remaining suggestions.

5. Experimental Methodology

We compiled the CUDA test programs using *nvcc* v6.0.1 with the *-O3 -arch=sm_35* flags. Our GPU is a Kepler-based 0.7 GHz Tesla K20c with 5 GB of main memory and 2496 CUDA cores distributed over 13 SMXs. Each SMX has 64 kB of fast memory that is split between the L1 data cache and the shared memory. The SMXs share a 1.5 MB L2 cache. For the machine learning methods, our tool leverages the algorithms implemented in Weka [34].

For the profiling, i.e., generating the feature vectors, we used *nvprof* from the Visual Profiler v6.5. We profiled each of the 128 versions of BH and NB described in Section 3.3 three times on the inputs shown in Table 1. Depending on the experiment, we use different subsets of the resulting feature vectors to train and test our tool. Table 2 lists the subsets used in each of the six experiments we performed. In experiments 1 through 4, we trained and tested based on the BH code. In experiments 5 and 6, the tool is trained on BH/NB and tested on NB/BH, respectively.

Table 1: Input sizes used for BH and NB

NB		BH	
Bodies	Time steps	Bodies	Time steps
50,000	2	125,000	2
100,000	2	250,000	2
100,000	5	250,000	5
200,000	5	500,000	5
-	-	500,000	10
-	-	1,000,000	10

Table 2: Experiments for evaluating the speedup predictions

Experiment	Training dataset	Training entries	Testing dataset	Testing entries	Testing includes training dataset	Train and test dataset from same prog. input
1	BH	64	BH	192	Yes	Yes
2	BH	64	BH	128	No	Yes
3	BH	128	BH	64	No	Yes
4	BH	192	BH	64	No	No
5	BH	192	NB	64	No	No
6	NB	192	BH	64	No	No

Since the tool sorts its recommendations by predicted speedup, our evaluation focuses on comparing the actual speedup of the tested optimizations with the predicted speedup. If the predicted speedup is reasonably close to the actual speedup, our tool is able to suggest the most useful optimization(s) to improve performance.

The strategy we chose for evaluating the results after training the tool is the following. For each specific optimization, we removed all feature vectors from runs that included this

optimization, which always leaves 32 feature vectors from runs that do not include the optimization. Testing these feature vectors on the trained tool generates six predicted speedups, one for each of the studied optimizations. The predicted speedups are then compared to the actual (measured) speedup when truly including this optimization in the code.

The ratio of the actual speedup (AC) over the expected speedup (EX) shows how close the prediction is to the true speedup. If the predictions are accurate, the tool can use them to rank the optimization, i.e., suggest the most promising optimizations (if any) to the user based on the expected speedup. To enhance readability, we show the AC/EX ratios in strip charts. A strip chart plots the data along a line with each data point represented by a star. The predictions do not have to be 100% accurate for our tool to work well. As long as the speedups are approximately correct, the tool will recommend the correct source-code optimizations, if any.

6. Results

This section presents the results of the prediction accuracy evaluations. We investigated three different machine learning methods to predict speedups: Logistic Regression, IBK, and M5P. Since the results of the logistic regression are substantially inferior to those of the other two methods, we only present results for IBK and M5P.

6.1 Train and Test on Same Code

When training and testing on the same program and input, the predictions are expected to be accurate. Instead of showing detailed strip charts for these simple experiments, we only compare the actual with the predicted speedup to see if they both show an increase or both show a decrease in performance. After all, if the predicted and the actual speedup are greater than one, it is correct for the recommendation tool to predict a performance gain. Similarly, if both the predicted and the actual speedup are less than one, using that optimization would hurt performance and not recommending the optimization is the correct behavior.

In experiment 1, we trained the tool based on the 64 feature vectors from a single run and input and tested all 192 feature vectors from the three runs of the same input, including the training data. Using the IBK method, on average over 97% of the predictions match the actual behavior, as shown in Table 3. The accuracy of the predicted behavior is significantly worse for M5P (86.4%). This reduced accuracy is largely due to M5P's inability to predict the behavior of FTZ, where it only achieves 57% accuracy. Upon further investigation we found that, in many of the test instances, FTZ applied by itself had very little impact on performance. Since M5P uses regression in the leaf nodes, even a small misprediction in the speedup can result in an incorrect final outcome (improvement vs. degradation). IBK does not suffer from this problem because it stores all of the training instances and is therefore able to predict the speedup of the training data exactly. IBK only enjoys this advantage if the training data include the test data. Next, we show how the accuracy of IBK is affected when we relax this assumption.

Table 3: Accuracy of negative/positive speedup predictions for different experiments and two ML methods

Experiment	Accuracy IBK (%)	Accuracy M5P (%)
1	97.3	86.4
2	96.0	86.4
3	96.3	33.3
4	92.0	81.6
5	83.6	33.3
6	55.7	60.1

6.1.1 Non-overlapping Training and Test Data

In experiment 2, we trained on the 64 feature vectors of a single run and tested on the 128 feature vectors from the other two runs. Although the training data are not included in the testing data, we still expect high accuracy because all feature vectors stem from the same program running the same input multiple times. The IBK results (96%) are almost identical to experiment 1 with just a slight decrease in accuracy due to excluding the training data from the testing dataset. The results for M5P are also very similar to those of experiment 1. M5P uses just a few features, so excluding the training data does not affect its prediction accuracy much.

6.1.2 Impact of Sample Size

In experiment 3, we trained the tool on 128 feature vectors and tested on the remaining 64 (experiment 2 uses the opposite approach). The expectation is that using more training data will improve the results. The prediction accuracies are comparable to the results from the previous experiments. Interestingly, the tested ML methods tend to underestimate the speedup. Nevertheless, the range of the ratios is 0.95 to 1.05 in all cases. These results show that adding more instances to the training data does not have a substantial impact on IBK. We note, however, that there is a significant drop in the accuracy of M5P. This is again explained by M5P's inability to accurately predict the behavior of FTZ. Although not shown, the accuracy of M5P is better in practice when using our tool with a threshold, i.e., when not recommending optimizations whose predicted speedup is below the threshold.

Obtaining about 96% prediction accuracy in the first three experiments is expected because training and testing on almost identical data (different runs of the same program and input) makes it easy for the tool to be accurate. In the following experiments, the training program input is different from the testing input.

6.1.3 Sensitivity to Program Input

In experiment 4, we trained the tool with all 192 feature vectors from the three runs on one program input and tested on 64 feature vectors each from the other program inputs. Figure 2 shows the results of the VOTE optimization with the

IBK method. The Y axis of the chart shows the ratio of the Actual Speedup (AC) over the Expected Speedup (EX). The closer the points are to 1.0 the more accurate the predictions are. The X axis represents different training and testing dataset combinations. Actually labeling the X axis resulted in illegible text, so we do not show the labels, which are not critical to the understanding. Note, however, that the input sizes increase from left to right and that the charts show sets of multiple strips for different runs of the same input size.

Most of the ratios in Figure 2 are around 1.0, i.e., the predicted speedups are close to the actual speedups. Unlike in the previous three experiments, where most of the IBK ratios were above 1.0, in this experiment the ratios are distributed evenly above and below the line. This is also true for the other optimizations shown in Figure 3. The few outliers in Figure 2 stem from test cases using the smallest inputs, which result in poor feature vectors that throw off IBK.

For WARP, SORT, and VOLA shown in Figure 3, the predictions on smaller inputs are also less accurate. The plotted ratios are denser close to the 1.0 line for all three optimizations because of the higher accuracy with larger inputs.

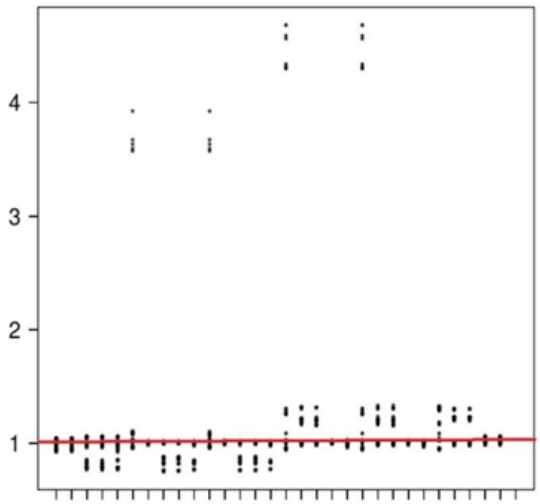


Figure 2: Ratios (AC/EX) of VOTE, Experiment 4, IBK

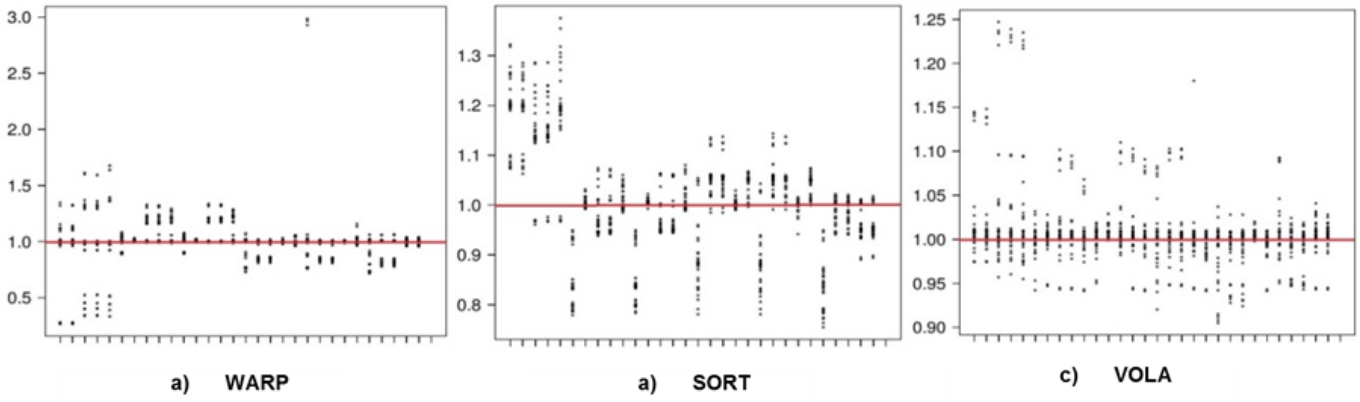


Figure 3: Ratios (AC/EX) of WARP, SORT, and VOLA, Experiment 4, IBK

Figures 4 and 5 show the IBK ratios for FTZ and RSQRT, respectively. The results are good for both FTZ and RSQRT. As shown in Table 3, the accuracy of positive/negative speedup is still 92% on average in experiment 4 for the IBK method. Clearly, training the tool on data from one input and testing on data from a different input does not hurt the tool's performance much. However, the accuracy of the prediction behavior of M5P is lower than IBK's (81.6%). This difference between absolute speedup prediction accuracy and behavior prediction accuracy, i.e., only predicting whether there will be a speedup, shows that the ratio of the actual speedup over the predicted speedup can be close to 1.0 yet the predicted speedup lies on the "other" side of the 1.0 line than the actual speedup. Fortunately, such cases are easily avoided in the recommendation tool by only suggesting optimizations that result in a speedup above the user-defined threshold.

6.2 Train and Test on Different Codes

Training the tool on a set of before and after feature vectors from code that is not related to the test code is the ultimate test of our approach (and the expected use case). In experiment 5, we trained on different versions of the BH code and used various versions of the NB code as test cases. In particular, this experiment shows results when we train the tool on data from an irregular GPU program and test it on a regular GPU program. Note that only the FTZ and RSQRT optimizations are common to both BH and NB. Hence, we can only compare the predicted and actual speedups of these two optimizations as we do not know the actual speedups of the remaining four BH optimizations when applied to NB.

Figures 6 and 7 show the results of experiment 5 using IBK. Almost half of the ratios are below the 1.0 line. The range of the ratios for FTZ is 0.2 to 1.7, which shows that

the prediction accuracy of the speedup is not as close as it was in the previous experiments. For RSQRT, the ratios are spread even wider. As before, the prediction results for test cases with larger input sizes tend to be better. For each model, we tested all 64 feature vectors of each set of four inputs on the NB code.

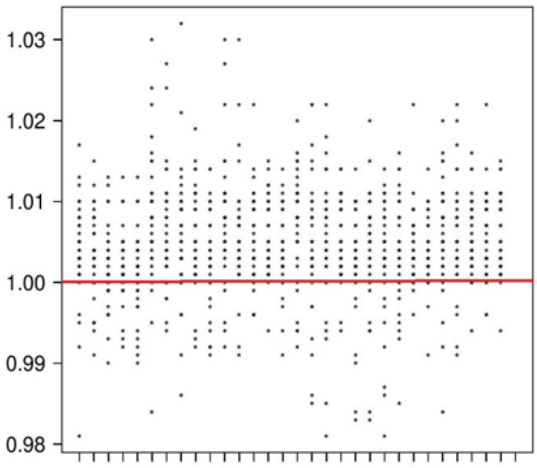


Figure 4: Ratios (AC/EX) of FTZ, Experiment 4, IBK

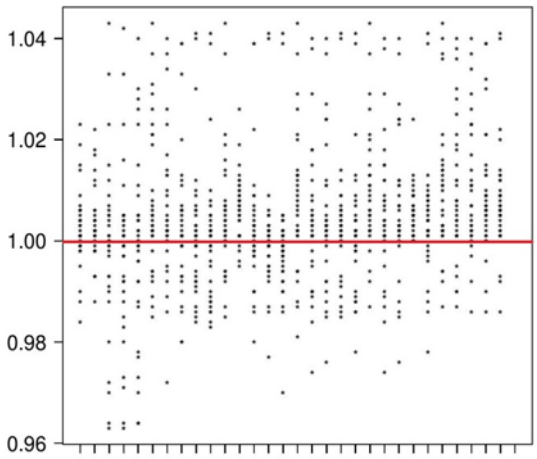


Figure 5: Ratios (AC/EX) of RSQRT, Experiment 4, IBK

Considering that we are training and testing on two different programs, the results are still good. The accuracy of the predictions for these two optimizations is almost 84%. The accuracy of the M5P method in this experiment for FTZ and RSQRT is only 33%. The reason for this low accuracy is that M5P uses very few features for making decisions. When the training and testing datasets stem from different programs, the possibility of accurate predictions based on just a few features is relatively low.

Experiment 6 is identical to experiment 5 except we switched the training and testing datasets, that is, we trained the tool on the regular NB code and tested it on the irregular BH code. Interestingly, all of the predicted speedups for FTZ using the IBK method are lower than the actual speedups on the BH code as shown in Figure 8. RSQRT yields more accurate predictions as Figure 9 shows. The range of the ratios is 0.78 to 1.45 and most of the ratios are close to 1.0. For smaller training and testing inputs, the tool tends to overestimate the speedups.

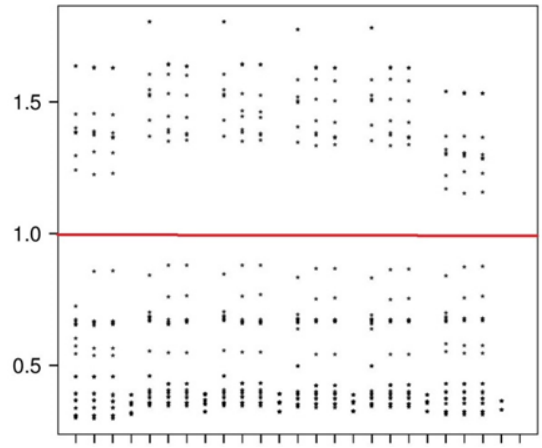


Figure 6: Ratios (AC/EX) of FTZ, Experiment 5, IBK

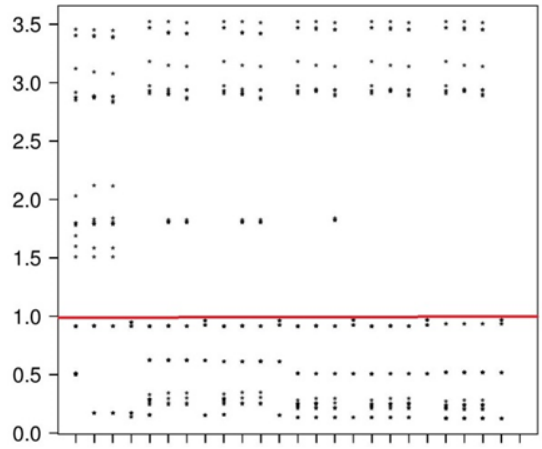


Figure 7: Ratios (AC/EX) of RSQRT, Experiment 5, IBK

Comparing the IBK results of experiment 6 with the results from experiment 5 in Table 3, we find that more accurate predictions are made when the tool is trained on irregular codes and tested on regular codes, which makes sense as irregular codes tend to be more complex.

In the first five experiments, the prediction accuracy of IBK is better than that of M5P. However, in experiment 6, the overall accuracy of M5P is better than that of IBK. Clearly, there is no ML model that is always the best for our tool. Apparently, M5P yields better performance because it narrows the features down to metrics that are significant for both irregular and regular codes. However, in most experiments, IBK yields more accurate speedup predictions than the other methods. Hence, IBK is the ML method of choice for our tool.

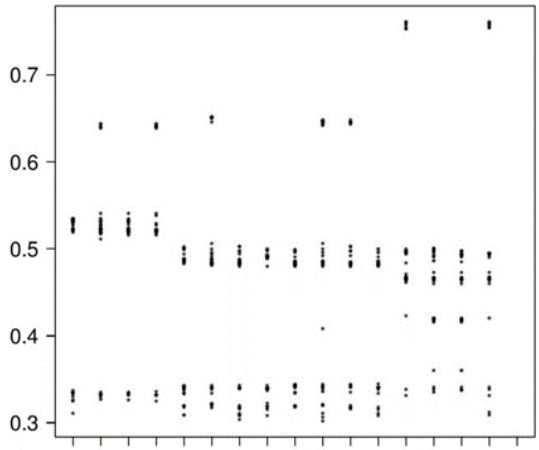


Figure 8: Ratios (AC/EX) of FTZ, Experiment 6, IBK

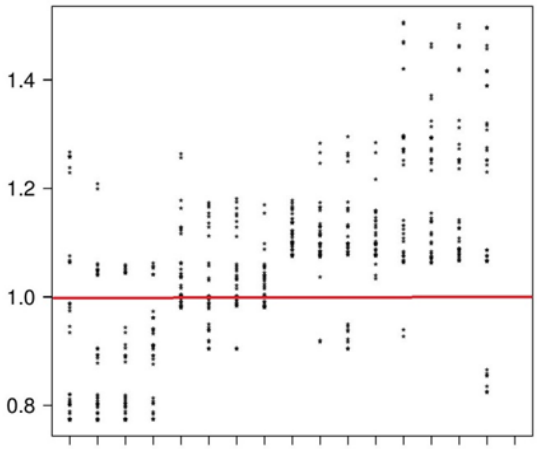


Figure 9: Ratios (AC/EX) of RSQRT, Experiment 6, IBK

7. Summary and Conclusion

This paper describes and evaluates a tool to suggest source-code optimizations to programmers in order to improve the efficiency of their GPU code, including complex irregular codes. The tool needs to be trained on profile data from different code samples that do and do not include certain source-code optimizations. During this training, the tool builds machine-learning models for each optimization in its database so that it can later estimate the speedup for each optimization when presented with profile data from other programs. To measure and quantify the prediction accuracy, we profiled differently optimized GPU codes with multiple inputs to gather a large set of performance data. The tool ranks the optimizations based on the predicted speedup and suggests the top optimizations to the user if the predicted speedup is above a preset threshold. To evaluate the accuracy of the predicted speedups, we compared them to the actual speedups obtained when truly adding the respective source-code optimizations.

We performed six experiments of training models and predicting speedups. In the first four experiments, we trained and tested the tool on the BH code and obtained up to 97% prediction accuracy. In the remaining two experiments, where we train on BH/NB and test on NB/BH, the tool delivers up to 82% accuracy, i.e., most of the suggested source-code optimizations truly result in a speedup when they are implemented.

Based on the results from Section 5, the predictions of our tool are more precise when training on data obtained with larger program inputs. This makes sense as larger inputs result in more profiling data and more stable-state utilization of the GPU. Expectedly, training the tool with more data yields better predictions.

When training on code that is different from the tested code, we found that training based on irregular codes and testing on regular codes seems to result in better predictions than training on regular code and testing on irregular codes. This is likely a combination of two factors. First, regular codes are less complex, making them easier to predict in general. Second, the higher complexity of irregular codes probably provides more diverse training data, which yield better ML models for making the predictions.

We studied three different machine learning methods. Our results show that there is no clear winner. However, IBK generally performs very well when predicting the likely speedup of source-code optimizations. Hence, we use IBK in out tool.

We used differently optimized Barnes-Hut implementations as a representative irregular GPU code. Of course, using additional (irregular) codes for training would be better. Also, we studied six source-code optimizations. Larger numbers of optimizations can and should be used to better test the accuracy of our approach. To verify portability, our study should be repeated on additional types of GPUs. For the machine learning phase, we investigated three different methods. Other types of ML methods could, of course, also be employed for predicting the speedups.

8. References

[1] http://www.hpcwire.com/2014/01/09/future-accelerator-programming/

[2] M. Mendez-Lojo, M. Burtscher, and K. Pingali. A GPU Implementation of Inclusion-based Points-to Analysis. 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 107-116. February 2012.

[3] Duane G. Merrill, Michael Garland, and Andrew S. Grimshaw. Scalable GPU Graph Traversal. 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. February 2012.

[4] M. Burtscher, R. Nasre, and K. Pingali. A Quantitative Study of Irregular Programs on GPUs. 2012 IEEE International Symposium on Workload Characterization, pp. 141-151. November 2012.

[5] https://developer.nvidia.com/nvidia-visual-profiler

[6] http://en.wikipedia.org/wiki/N-body_problem

[7] http://en.wikipedia.org/wii/Barnes%E2%80%93Hut_simulation

[8] http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu

[9] http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

[10] Ross J. Quinlan, Learning with Continuous Classes, Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, Singapore, 343-348, 1992.

[11] B. P. Miller and J. K. Hollingsworth and M. D. Callaghan, The Paradyn Performance Tools and PVM, Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing: Townsend, TN, USA, 25–27 May 1994, pp. 201-210, Society for Industrial and Applied Mathematics, 1994.

[12] Bernd Mohr and Felix Wolf, KOJAK - a tool set for automatic performance analysis of parallel programs, Springer-Verlag, 2003.

[13] Zoltán Szebenyi, Brian J. N. Wylie, Felix Wolf: SCALASCA Parallel Performance Analyses of SPEC MPI2007 Applications. In Proc. of the 1st SPEC International Performance Evaluation Workshop (SIPEW), Darmstadt, Germany, volume 5119 of Lecture Notes in Computer Science, pages 99-123, Springer, 2008.

[14] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, Bernd Mohr: The SCALASCA Performance Toolset Architecture. In Proc. of the International Workshop on Scalable Tools for High-End Computing (STHEC), Kos, Greece, pages 51–65, June 2008.

[15] W. E. Nagel and A. Arnold and M. Weber and H.-Ch. Hoppe and K. Solchenbach, VAMPIR: Visualization and Analysis of MPI Resources, 1996.

[16] Matthias S. Müller and Andreas Knüpfer and Matthias Jurenz and Matthias Lieber and Holger Brunst and Hartmut Mixand Wolfgang E. Nagel, Developing Scalable Applications with Vampir, VampirServer and VampirTrace, PARCO, Advances in Parallel Computing, Vol. 15, pp. 637-644, IOS Press, 2007.

[17] S. Shende and A. D. Malony, "The TAU Parallel Performance System," International Journal of High Performance Computing Applications, SAGE Publications, 20(2):287-331, Summer 2006

[18] Michael Gerndt, Karl Fürlinger, and Edmond Kereku, Periscope: Advanced Techniques for Performance Analysis, PARCO, John von Neumann Institute for Computing Series, Vol. 33, pp. 15-26, Central Institute for Applied Mathematics, Jülich, Germany, 2005.

[19] Allen D. Malony, Scott Biersdorff, Wyatt Spear, and Shangkar Mayanglambam. 2010. An experimental approach to performance measurement of heterogeneous parallel applications using CUDA. In Proceedings of the 24th ACM International Conference on Supercomputing (ICS '10). ACM, New York, NY, USA, 127-136. DOI=10.1145/1810085.1810105 http://doi.acm.org/10.1145/1810085.1810105

[20] http://www.vi-hps.org/projects/score-p/

[21] Laksono Adhianto and Sinchan Banerjee and Michael Fagan and Mark Krentel and Gabriel Marin and John Mellor-Crummey and Nathan Tallent, HPCToolkit: Performance tools for parallel scientific computing, SC'08 USB Key, ACM/IEEE, November 2008.

[22] http://www.hpctoolkit.org

[23] https://developer.nvidia.com/cuda-profiling-tools-interface

[24] https://developer.nvidia.com/nvidia-visual-profiler

[25] http://www.nvidia.com/object/nsight.html

[26] Browne, S., Deane, C., Ho, G., Mucci, P. "PAPI: A Portable Interface to Hardware Performance Counters," Proceedings of Department of Defense HPCMP Users Group Conference, June, 1999.

[27] http://software.intel.com/en-us/intel-vtune-amplifier-xe

[28] Michael Knobloch and Timo Minartz and Daniel Molka and Stephan Krempel and Thomas Ludwig 0002 andBernd Mohr, Electronic poster: eeclust: energy-efficient cluster computing, SC Companion, pp. 99-100, ACM, 2011.

[29] http://www.vi-hps.org/projects/

[30] Fursin, Grigori, Cupertino Miranda, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Ayal Zaks, Bilha Mendelson et al. "MILEPOST GCC: machine learning based research compiler." In GCC Summit. 2008.

[31] Bergstra, J.; Pinto, N.; Cox, D., "Machine learning for predictive auto-tuning with boosted regression trees, "Innovative Parallel Computing (InPar), 2012, vol., no., pp. 13-14 May 2011.

[32] M. Burtscher, B.D. Kim, J. Diamond, J. McCalpin, L. Koesterke, and J. Browne. "PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications." SC 2010 Int. Conference for High-Performance Computing, Networking, Storage and Analysis. November 2010.

[33] Olalekan Sopeju, Martin Burtscher, Ashay Rane, and James Browne. AutoSCOPE: Automatic suggestions for code optimizations using PerfExpert. 2011 International Conference on Parallel and Distributed Processing Techniques and Applications, pages 19-25, July 2011.

[34] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten, The WEKA Data Mining Software: An Update, SIGKDD Explorations, Volume 11, Issue 1, 2009.

Proposal and Implementation of Mixed Finite Automata Optimization by Balancing Active States and Transitions

Kosuke Nishimura[+], Kenichi Takagiwa[+], Hiroaki Nishi[*]
Graduate School of Science and Technology, Keio University, Japan
[+]{nishimura, takagiwa}@west.sd.keio.ac.jp, *west@sd.keio.ac.jp

Abstract – *A string matching program corresponding to a regular expression is implemented using two different finite automata: a nondeterministic finite automaton (NFA) and a deterministic finite automaton (DFA). However, these finite automata both have their own pros and cons. An NFA allows activating multiple states. It increases the access latency to the state memory because of frequent memory accesses, thereby degrading the processing throughput. In contrast, its memory footprint can be reduced. The memory footprint of a DFA becomes larger than an NFA. A DFA can improve the matching processing throughput and reduce the number of memory accesses because it always has one active state. In this paper, a mixed FA (MFA), a new automaton combining an NFA and a DFA, is proposed. An MFA combines an NFA and a DFA by changing their mixing ratio, and enables an adjustment of the memory footprint and the maximum number of active states.*

Keywords: Regular expressions, finite automata, string matching.

1 Introduction

A network intrusion detection system (NIDS) and an intrusion prevention system (IDS) are significant applications for cyberattack measures. An NIDS acquires packet data in a communication flow of a network, and provides intrusion detection or attack prevention by analyzing the flows in real time [1]. When an NIDS detects malformed data, it informs the arrival of the data to an administrator. A well-known NIDS software is Snort [2] [3], which detects attacks by matching a dedicated database against the data flowing in the network. The database consists of signatures and rules generated according to the characteristics of various attack methods.

An early NIDS only utilized single strings for describing the characteristics of viruses as their signatures. In recent years, an NIDS has been developed to use regular expression and extended regular expressions (Perl Compatible Regular Expression, PCRE [4]), instead of single strings. Regular expressions can describe a wide variety of patterns in a single string with special characters to detect diversified and sophisticated attacks, which are evolving into threats to newer computer systems. Therefore, it is natural that NIDS uses a regular expression for explaining complex strings of viruses and malformed attacks as their signatures or attacking processes for detecting them. Owing to the flexibility of regular expressions in describing these signatures and patterns, a regular expression is applied in various applications, such as content-based spam e-mail filters [5].

As a result of its many advantages, an NIDS with a matching function using a regular expression plays an important role in computer security, and its needs are increasing. However, the implementation of a regular expression as a computer software function has several problems, such as a large memory footprint in describing complex and massive expressions, and a reduction of the processing throughput caused by frequent memory accesses. This paper focuses on a string matching program that can detect complex strings using regular expressions.

A string matching program based on regular expressions are conventionally implemented using two different finite automata: a nondeterministic finite automaton (NFA) and a deterministic finite automaton (DFA) [6]. These finite automata both have their own pros and cons. Because an NFA permits multiple transitions of states in a character-by-character manner, various states can be activated. However, this characteristic degrades the processing throughput because it requires multiple memory accesses to step the matching process of one character to another. A DFA has a feature in which the number of state transitions is always limited one for each processing step of a character. This feature enables an acceleration of the processing throughput because one memory access is sufficient to step the matching process. Therefore, a DFA is faster than an NFA in its matching process throughput. However, a DFA increases the memory footprint in describing regular expressions because the number of states is exponentially increased according to the size and complexity of the given regular expressions. Hence, an NFA takes a longer time in match processing than a DFA [5].

As mentioned, there is a trade-off between an NFA and a DFA in terms of memory footprint and processing throughput. As a method for breaking this tradeoff, a compression technique of state memory in a DFA has been proposed [5] [7] [8]. A dual FA [5] isolates the parts of the automata used for processing special characters of repeats and compresses the total memory usage by preventing an exponential enlargement of the repeat process. This straightforward technique can reduce the memory usage. However, it does not consider the trade-off of memory usage and processing throughput. If a given automaton is composed of only repeats, it generates an automaton equal only to an NFA. Hence, the processing throughput of a dual FA is drastically degraded. As another approach, Google developed the RE2 regular expression algorithm [9], which solves the memory exhaustion problem by switching from a DFA to an NFA when the memory footprint exceeds a certain amount of memory usage. However, RE2 does not consider the optimization of memory usage. The

available memory footprint is varied based on the application or implemented system. Namely, a new method is required that utilizes all of the available memory space efficiently to maximize the processing throughput of regular expression matching.

In this paper, a mixed FA (MFA) is proposed, which considers this trade-off. It flexibly combines an NFA and a DFA according to the available memory footprint. An MFA changes the mixing ratio of an NFA and a DFA automatically to fit the best mixing ratio. Namely, as its novel feature, an MFA can balance the memory footprint and the maximum number of active states by maximizing the total performance under the limited memory space varied by the target applications. When there is a problem of required memory size in using a DFA, an MFA can optimize the required memory footprint. Although it degrades its processing throughput as a drawback, an MFA can provide a better performance than an NFA.

2 Research Background

2.1 Regular Expression

A regular expression describes a set of strings. For example, the regular expression "[bc]ook" matches "book" and "cook." A regular expression enables complex patterns to be searched efficiently. As an example, well-known applications such as grep text filter, vi text editor, and many kinds of script-based programming languages support regular expressions.

The matching string "[A-Q]" in a regular expression means it matches any character between A and Q. The wildcard character "." means it matches any one character. Simple character repetitions can be described using the expressions "a?" meaning that "a" is repeated zero or one time, "a*" meaning that "a" is repeated zero or any number of times, and "a+" meaning that "a" is repeated at least once. If repetition special characters of "?", "+," and "*" are used in a DFA, an increase in the number of states and transitions will occur. This problem should be solved, and a technique addressing the problem is described later in this paper.

2.2 Finite automaton

A finite automaton is a mathematical model with a discrete input and output. The destination state (in some cases the current state is same as the original state) is always unique in all states. One of the states is the initial state. Some of the states are the final states. The input characters of a string are processed from the initial state. A state transits to another state in a one-by-one according to the input characters. This state transition process repeats until the state reaches the final state, or the input string is exhausted.

2.2.1 Nondeterministic Finite Automaton

An NFA and a DFA are pattern matching automata that handle a set of regular expressions. An NFA permits multiple state transitions. Namely, multiple states can be activated in an NFA when an input character is received. Fig. 1 shows the state transition graph of an NFA generated for processing the regular expressions "c(a|b)*a." The circles and arrows in Fig. 1 represent states and transitions, respectively. In this figure, state q0 is the initial state, and state q2 is the final state. In Fig. 1, two arrows of "a" are output from q1. Therefore, if the character "a" is received as an input, the states of both q1 and q2 become active.

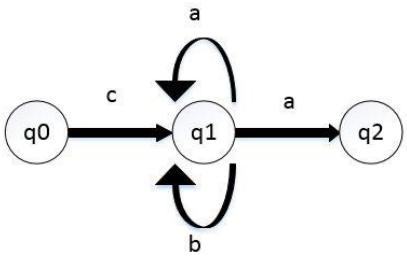


Fig. 1: The NFA for "c(a|b)*a"

The details of the activation process are as follows. When the NFA of "c(a|b)*a" of Fig. 1 processes the input text "cba," this NFA executes the process of the state transition given as

$$(q0) \xrightarrow{c} (q1) \xrightarrow{b} (q0, q1) \xrightarrow{a} (q0, q1, q2)$$

In this case, the maximum number of active states is three. Theoretically, an NFA requires the widest bandwidth in accessing memory. This characteristic degrades the processing throughput because it requires multiple memory accesses to step the matching process one character to another, especially when processing complex regular expressions. This is a disadvantage of an NFA. However, an NFA has an advantage in that the memory footprint is smaller because the number of states and transitions is less than in a DFA. This phenomenon is clearly shown when comparing the state transition graphs of an NFA in Fig. 1 and a DFA in Fig. 2.

2.2.2 Deterministic Finite Automaton

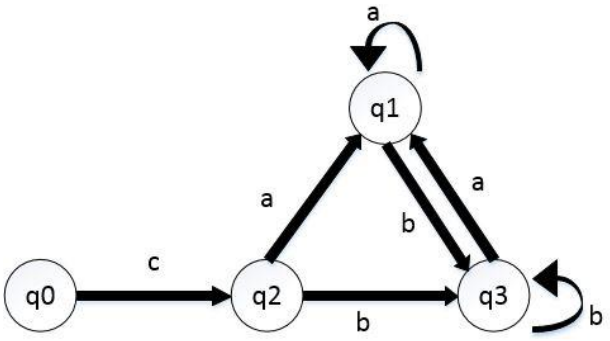


Fig. 2: The DFA for "c(a|b)*a"

The DFA for the regular expression "c(a|b)*a," which is the same as the example for an NFA, is shown in Fig. 2. As a difference from an NFA, every state in a DFA has only one

outgoing transition for each character. Therefore, the numbers of states and transitions of a DFA are larger than those of an NFA. In particular, a DFA consumes a larger memory footprint for processing repeat special characters because all possible states and transitions generated by the special characters should be stored in memory. This is a disadvantage of a DFA. However, a DFA always has only one active state because each state has one outgoing transition for each string. This feature is an advantage of a DFA in maximizing the matching process throughput and minimizing the processing time.

2.2.3 Nondeterministic Finite Automaton with ε transitions

An NFA allows a state transition by using null character ε as an extension of its function. An epsilon transition ε permits transitions without receiving an input string. Fig. 3 shows the ε-NFA transitions generated from a regular expression. These state transition graphs are atomic regular expressions. The filled-in circles in Fig. 3 represent the accepting state, where the state means that the input pattern matches the target regular expression. In the proposed MFA, this translation is accomplished first.

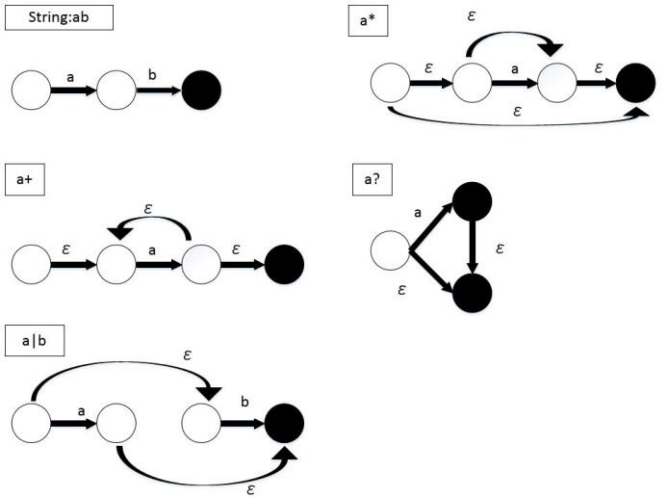


Fig. 3: Conversion from the regular expression to an NFA with ε transitions

3 Mixed Finite Automaton

3.1 Purpose

As mentioned above, string matching algorithms for a regular expression are generally implemented using either an NFA or a DFA. However, these finite automata both have their own pros and cons. The characteristics of an NFA and a DFA can be summarized in Table 1. It is necessary to consider the trade-off between the memory footprint and the maximum number of active states, namely between the size and processing throughput.

Table 1: Features of an NFA and a DFA

	Space complexity	Time complexity
NFA	Small	Large
DFA	Large	Small

The proposed MFA is adaptable to a variety of network applications. An MFA combines an NFA and a DFA flexibly according to the available memory footprint, and enables a change in the mixing ratio of both. It is possible to balance the memory footprint and the maximum number of active states by mixing an NFA and a DFA. An MFA observes this strategy, and by using the following dedicated algorithm, can maximize the total performance under a limited and varying memory space based on the target applications.

3.2 Algorithm

This section describes the method for combining an NFA and a DFA in an MFA. Initially, a regular expression is divided into two groups of atomic regular expressions. The first group is converted into a DFA, and the other group is converted into an NFA. This conversion order is effective from the viewpoint of memory access throughput. The reason this order is effective can be described through a simple example. As an example, a regular expression consists of two atomic regular expressions. If the basic syntax of these two expressions is the same, there are any difference between a case converted into DFA+NFA and one converted into NFA+DFA. However, the number of activated states differs between them. For the DFA+NFA case, there is one active state of the first DFA part. The NFA part increases the number of active states. The total number of active states depends only on the last NFA group. For the NFA+DFA case, the first DFA part increases the number of active states. The last NFA part maintains the number of active states. Namely, the total number of active states is about twice that of the first DFA part. This is an approximate estimation. In fact, there is one active state in the initial state, and this number gradually increases. Even with this fact, the total number of active states is larger than DFA+NFA. This is proved using a simple birth process under a multidimensional Markov diffusion process.

A regular expression "(a|b)*a(a|b)*a" is used as an example of MFA generation. To generate an MFA, this regular expression is separated into atomic regular expression parts. Each atomic regular expression is shown in Fig. 3. These atomic regular expressions can be converted into an NFA or a DFA. The given regular expression is divided into r1 = "(a|b)*," r2 = "a," r3 = "(a|b)*," r4 = "a," as atomic regular expression parts. In this case, these four atomic regular expressions are grouped into two DFA and NFA groups.

An MFA can adjust the mixing ratio of an NFA and a DFA flexibly by controlling the boundary between NFA and DFA groups, as shown in Fig. 4. The variety of burden between an NFA group and a DFA group assures the flexibility of the mixing ratio of an NFA and a DFA in an MFA. An MFA can

maximize the processing throughput by considering the memory consumption under the given memory footprint. An application can use the optimized matching automata of the given regular expression using an MFA.

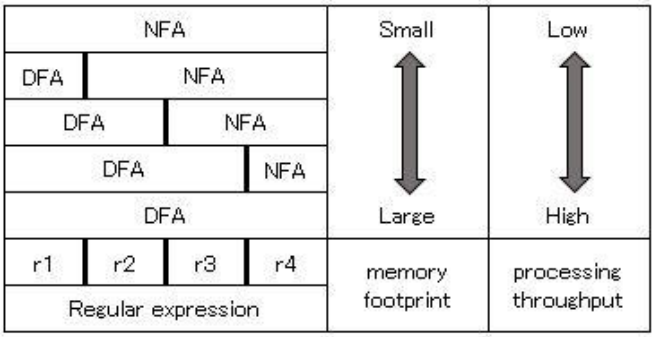


Fig. 4: Characteristics of MFAs with a different boundaries between NFA and DFA groups

3.3 Examples of Mixed FA

This section describes an example of a generated MFA. As an input regular expression, "(a|b)*a(a|b)*a" is used as well as section 3.2. This regular expression is divided into four atomic regular expression parts, i.e., r1 through r6. As an example of an MFA, "r1r2r3" is converted into an NFA, and "r4r5r6" is converted into a DFA.

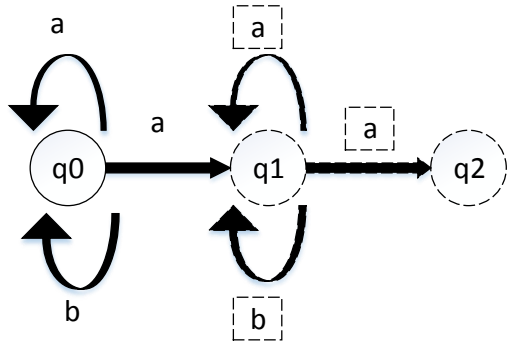


Fig. 5: NFA for "(a|b)*a(a|b)*a"

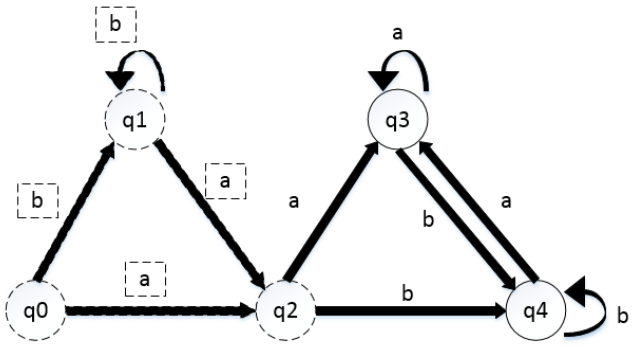


Fig. 6: DFA for "(a|b)*a(a|b)*a"

Figs. 5 and 6 show automata of an NFA and a DFA generated from the given regular expression. Through Figs. 5 and 6, the maximum number of active states of an NFA and a DFA can be counted. For an evaluation, the memory footprint is defined as the number of states and transitions. The NFA of Fig. 5 has three states and six transitions, whereas the DFA of Fig. 6 has five states and ten transitions. Hence, the memory footprint of a DFA is larger than that of an NFA in terms of the numbers of both the states and transitions.

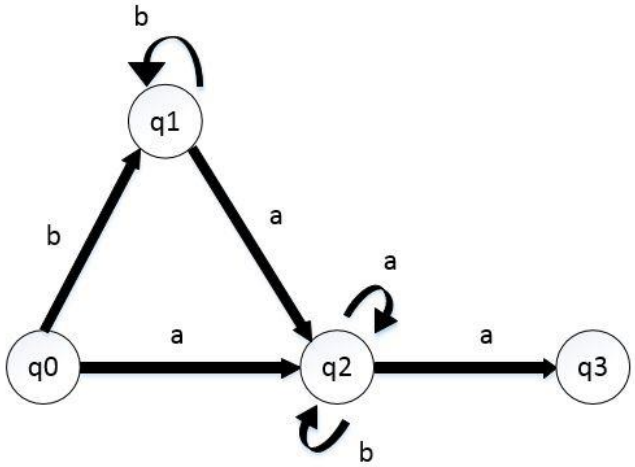


Fig. 7: MFA for "(a|b)*a(a|b)*a"

Fig. 7 shows an example of an MFA composed by combining the states and transitions of Figs. 5 and 6. In mixing the DFA and NFA, the composed MFA selects and mixes the dotted parts of the NFA of fig. 5 and DFA of fig. 6. The MFA of Fig. 7 supports the regular expression "(a|b)*a(a|b)*a."

In the composed MFA shown in Fig. 7, states q1 and q3 become active states when character "a" is input twice. Namely, two states become active. In contrast, three states become active in the NFA of Fig. 5, and the number of active states is always one in a DFA. Hence, the size of the memory footprint and the number of active states of an MFA are smaller than those for a DFA and an NFA, respectively. An MFA can flexibly adjust the mixing ratio of an NFA and a DFA by controlling the boundary between NFA and DFA groups. The number of states, number of transitions and maximum number of active states are compared in Table 2.

Table 2: Performance comparison of each automaton

	Number of states	Number of transitions	Maximum number of active states
NFA	3	6	3
MFA	4	7	2
DFA	5	10	1

4 Evaluation

7777Experimental environment

In this evaluation, an NIDS application is used for evaluating an MFA for a comparison with an NFA and a DFA. Snort is a well-known application of NIDS, as described in section 1. Snort rule sets consist of regular expressions for detecting the signatures of malformed messages in a network so as to prevent security attacks. Hence, regular expressions used in Snort are an appropriate benchmark rule set for this evaluation. We extracted the benchmark rule sets from Snortrules-snapshot-2970 of Snort ver. 2.9 [2]. The regular expression "a{n}" means that n iterations of "a" is partially supported. When more than ten iterations are found, the character is changed to a regular expression character "+." When fewer than ten iterations are found, it means these iterations were designed correctly.

Programming language C++ is used in implementing the regular expression processor, and g++ Ver4.4.7 is used as a compiler. The regular expression processor used in this experiment can generate not only an MFA but also an NFA and a DFA as special cases of an MFA. The number of states and transitions, the maximum number of active states, the configuration time, and the computation time of each automaton were evaluated using the regular expression processor. The configuration time of the automata denotes the delay time in generating an MFA for a target regular expression.

Table 3 shows an extracted pattern as a benchmark rule set from the Snort rule sets. These regular expression patterns are randomly selected from Snortules-snapshot-2970. The regular expression processor inputs these regular expressions one after another. The results are described in the next section.

Table 3: Regular expression pattern extracted from Snortrules-snapshot-2970 [2]

(\s*|\s*\r?\n\s+)
malware(\w|\s)*\d{10}
.*Root\x2User-cgi\x2f.*\x2ecgi[a-z0-9]+
\s\w+\s\d+\r?\n[^\n]*
(no|up|\d+\x2e\d+\x2e|d+\x2e\d+)
.PHP[a-z]+[a-f0-9]+[a-z]+=.*[a-z]+=.*[a-z]
\w+\x3b.*\x3b.*\x3b
.*aspn\x2fvgi-bin\x2f.*\x2ecgi[a-z0-9]+
User-Agent[^\n]*\x2eDIAN
Server\x3a[^\r\n]*Root{^\r\n]*Kit[^\r\n]*Scaner

4.1 Experimental results

Fig. 8 shows the number of states, the number of transitions, and the maximum number of active states of an NFA, a DFA, and an MFA.

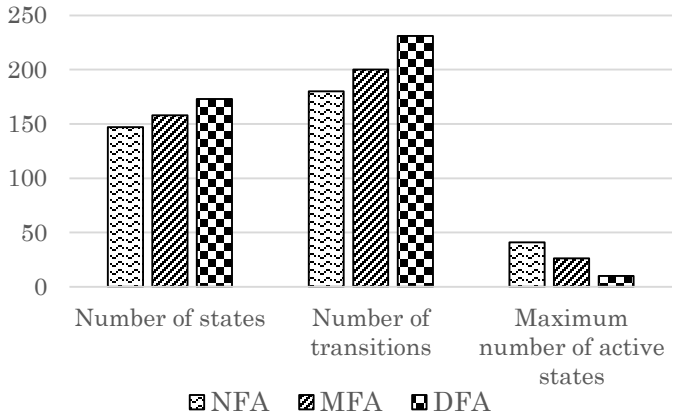


Fig. 8: The number of states and transitions, and the maximum number of active states of each automaton

As shown in Fig. 8, even in the case of Snort rules, the trends of the number of states and the number of transitions and maximum number of active states are the same with the cases of the simple examples shown in Figs. 5 and 6. Namely, the numbers of states and transitions of a DFA are larger than those of an NFA. In addition, the maximum number of active states of an NFA is larger than that of a DFA. In both cases, an MFA ranks between an NFA and a DFA. This result shows that the proposed MFA can provide the benefit of both an NFA and a DFA by considering the tradeoff between the size and the throughput.

Next, we will show that an MFA can adjust the memory footprint and the maximum number of active states. In this experiment, 100 KB of random text data was used. After dividing regular expression into ten, the regular expression processor of the MFA is inputted it. The configuration and computational time of the MFA are shown in Fig. 9. In the figure, the x axis indicates the mixing ratio of the DFA and NFA. A comparison of the number of states, the number of transitions, and the maximum number of active states in varying mixed rates of the NFA and DFA are shown in Fig. 10.

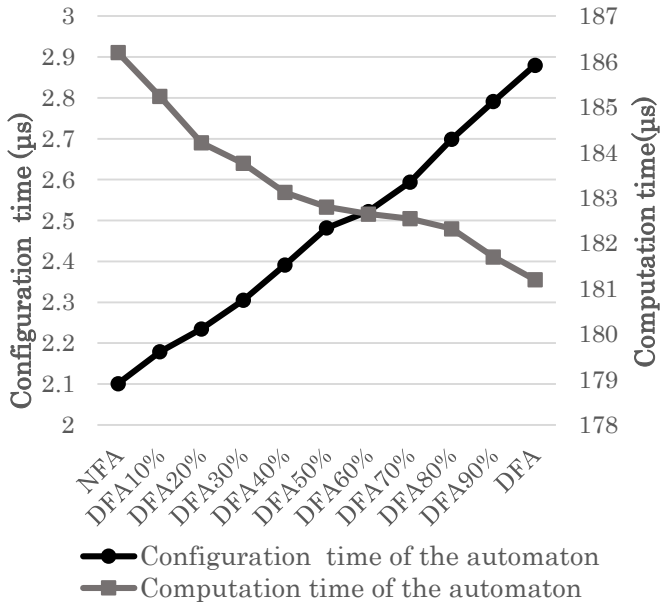


Fig. 9: Configuration and computation times of mixed FA when changing the ratio

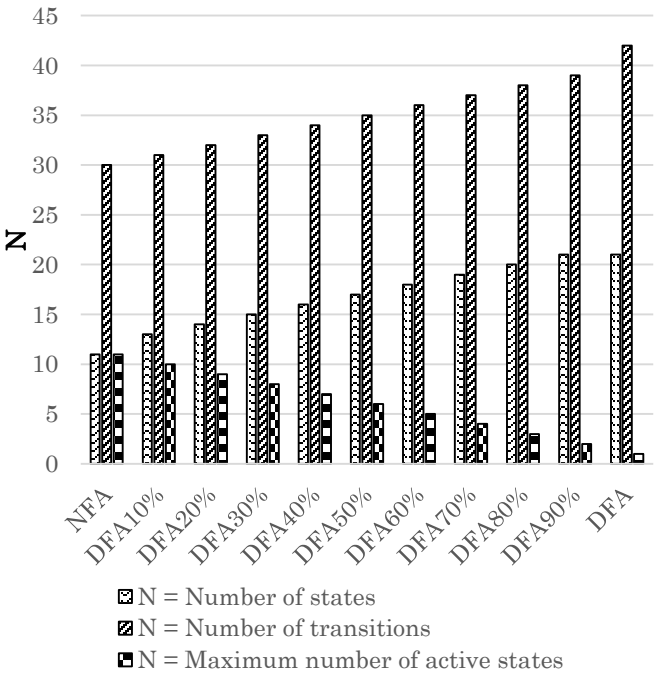


Fig. 10: The number of states and transitions, and the maximum number of active states of a mixed FA when changing the ratio

As shown in Fig. 9, the configuration time of the automaton was increased in association with the increase in the DFA ratio. In contrast, the computation time was reduced. As a result of the computational time, an MFA can vary the computation time. Fig. 10 demonstrates that the numbers of states and transitions were gradually increased corresponding to the increase in the DFA ratio. Namely, the memory footprint becomes larger in this case. In contrast, the maximum number of active states was reduced. Therefore, the computation time in Fig. 9 was improved.

Based on the above description, an MFA enables the memory footprint and the maximum number of active states to be adjusted for optimizing the total performance under the limited memory space available in a target application.

5 Conclusion

In this paper, an MFA, new automaton combining an NFA and a DFA, was proposed, implemented, and evaluated using the Snort rule set. An MFA combines the existing string matching programs of an NFA and a DFA by changing their mixing ratio. The results indicate that the proposed MFA can optimize the matching throughput under the required memory footprint size by combining an NFA and a DFA. An MFA enables a trade-off between the memory footprint and processing throughput, and varies both. Therefore, it can maximize the processing throughput while fulfilling the conditions of the memory size. An MFA has the potential to be applied to various applications owing to its flexibility.

ACKNOWLEDGEMENT

This work was partially supported by the funds of SECOM Science and Technology Foundation, and by MEXT/JSPS KAKENHI Grant (B) Number 24360230 and 25280033.

6 References

[1] S. Kumar, B. Chandrasekaran, and J. Turner, "Curing regular expressions matching algorithms from Insomnia, Amnesia, and Acalculia," In Proc. of ANCS'07, pp. 155-164.ACM.

[2] Snort. http://www.snort.org

[3] T.T. Hieu, T.N. Thin, T.H. Vu, S. Tomiyama, "Optimization of Regular Expression processing circuits for NIDS on FPGA," Second International Conference on Networking and Computing, 2011

[4] J. Shangjie, L. Mejian, "Research and Design of Preprocessor plugin based on PCRE under Snort Platform," Control, Automation and Systems Engineering (CASE), 2011 International Conference, IEEE, 30-31 July 2011

[5] C. Liu, J. Wu, "Fast Deep Packet Inspection with a Dual Finite Automata," Computers, IEEE Transaction on (Volume 62, Issue 2), Feb.2013

[6] J.E. Hopcroft, J.D. Ullman, "Introduction to Automata Theory," Addison Wesley, 1979

[7] M. Becchi, P. Crowley, "A Hybrid Finite Automata for Practical Deep Packet Inspection," CoNEXT 2007.

[8] J. Zhang, D. Zhang, K. Huang, "A Regular Expression Matching Algorithm Using Transition Merging," 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing,

[9] RE2. https://github.com/google/re2

A Novel Distributed Arithmetic Multiplierless Approach for Computing Complex Inner Products

Kevin N. Bowlyn[1], and Nazeih M. Botros[2]
1. Ph.D. Candidate, Dept. of Electrical & Computer Engineering, Southern Illinois Univ., Carbondale, IL, USA
2. Professor Emeritus, Dept. of Electrical & Computer Engineering, Southern Illinois Univ., Carbondale, IL, USA

Abstract - *In this paper we present a new integration approach for computing complex inner products using the Distributed Arithmetic (DA) technique and Complex Binary Number System (CBNS). By using the CBNS technique each complex number can be represented as one single unit instead of two. Our extended goal is to apply the approach in designing Fast Fourier Transform and realize the design on field-programmable gate arrays (FPGAs). A DA look-up-table (LUT) is used to store all linear combinations of coefficients. Our results show that for a radix-2 FFT computation, the number of "N" point arithmetic calculations would be decreased by 75% and approximately 67 % for the total number of real adders when compared to the traditional radix-2 FFT computation. This proposed design is multiplierless. The approach is implemented on a 3-tap filter; preliminary analysis shows a power consumption reduction of approximately 55% compared to the MAC approach.*

Keywords: Distributed arithmetic (DA), fast Fourier transform (FFT), look-up-table (LUT), complex binary number system (CBNS), complex numbers

1 Introduction

Most digital devices in today's technology require Digital Signal Processing (DSP) for calculating the vector dot products between two vectors. Complex numbers are widely used within many DSP applications and are considered essential for the different computer applications that are greatly dependent on the arithmetic use of complex numbers [1]. In spite of the fact that there is a great need for a better representation of complex numbers, as a single unit, especially with the FFT algorithm, today's modern technology still relies heavily on the divide and conquer approach, where real and imaginary number parts are treated as two separate entities [1].

1.1 Fast Fourier Transform

The traditional FFT algorithm which uses the vector dot products basically uses the divide and conquer approach by breaking up the Discrete Fourier Transform (DFT) computation into two half lengths of even and odd indexes [2] [3]. This approach of splitting up the DFT is quite useful as the output from each stage of the butterfly structure becomes the input of the next consecutive stage [2]. Although this computational algorithm is fast and efficient, in calculating the DFT, which greatly reduces the number of multipliers from an order of N^2 multiples to $N/2 \log_2 N$ multiples, this Radix-2 DIT algorithm still requires an extensive number of multipliers which are quite costly [4].

1.2 Distributed Arithmetic

The DA approach uses less hardware structure and computation without the use of any multiplier hardware structure [2]. The DA approach replaces the multiplier by using Look-up-tables (LUT). The LUT memory however, may grow exponentially in size from 2^K to 2^{K+1} words which is one of the major factors within this technique. For example, a 256-k DA based tap filter would require a memory size of 2^{257} which is significant. However, there are several different algorithms such as the use of ROM decomposition, the use of modifying the adder to an adder/subtractor, and the use of offset binary coding in which the size of the ROM can be greatly reduced [5]. Overall, the DA approach has been designed and tested to become the efficient tool in computing the vector dot product without any dedicated multipliers. It also consumes far less power and time in computing the inner dot product between two vectors [6] [7].

1.3 Complex Binary Number System

The CBNS algorithm uses far less arithmetic computations for computing complex numbers as each complex number is treated as a single entity instead of two. Unlike the traditional approach of calculating complex numbers, this method does not rely on the divide and conquer approach. This algorithm was first introduced by Penney in 1964 [8] [9] who developed a negative 4 base conversion for the complex number system. Later, in 1965, he introduced the (-1 + j) complex binary base algorithm in which a complex number is represented as a single entity in order to perform certain arithmetic computations such as addition, subtraction, and multiplication [1] [10]. Jamil [1] reintroduced in 2000 an efficient (-1+j)-based algorithm for doing addition, subtraction, multiplication and division using complex numbers [1] [11-16].

1.4 Aims/Objectives

The CBNS algorithm indicates the importance of having a single unit representation for complex numbers as this will be a great benefit given 21st century technology. Not only will it demonstrate a great advantage in computer applications but this algorithm will also result in superior performance in today's computer designs. Integrating this algorithm with the DA approach will result in a 100% reduction in the hardware structure as no multipliers are used in computing the FFT structure which will result in a decrease in area size and cost, as multipliers occupy a large volume of hardware and they are fairly costly in implementing.

1.5 Paper Structure

The remainder of this paper is organized as follows. Section 2 presents the proposed DA structure and ways on how to have the memory size reduced. A conversion of complex binary number system is presented in section 3. The results are presented in section 4. Work in progress is presented in section 5 and conclusion is presented in section 6.

2 The Distributed Arithmetic (DA)

In DA, multiplications are reordered and mixed such that the arithmetic becomes "distributed" through the structure rather than being "lumped". DA is very effective in calculation of inner products without the use of multipliers. It relies on simple operations: adders, shifters and look up tables (LUT). DA facilitates the mapping of these operations onto FPGAs. DA computes the inner products and stores all possible linear combinational sums in a LUT ROM. The inputs of the vector are two's complement binary numbers between fixed and variable input data with the most significant bit, which is the sign bit located to the left of the binary point [17] [18] [19]. In explaining the DA technique, consider the sum of products from equation (1)

$$y = \sum_{k=1}^{K} A_K x_K \tag{1}$$

where x_K is a 2's-complement binary number scaled such that $|x_K| < 1$ (fixed point number) input data and A_k are fixed coefficient vectors**.** Therefore, x_k can be expressed as

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \tag{2}$$

where b_{k0} is the sign bit, b_{kn} are bits 0 or 1, $b_{k,N-1}$ is the least significant bit (LSB), and N is the word length of the input variable. Substituting equation (2) into (1) and changing the order to express y in terms of the bits x_k yields the following equations:

$$\begin{aligned} y &= \sum_{k=1}^{K} A_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right] \\ y &= -\sum_{k=1}^{K} (b_{k0} \bullet A_k) + \sum_{k=1}^{K} \sum_{n=1}^{N-1} (A_k \bullet b_{kn}) 2^{-n} \end{aligned} \tag{3}$$

This equation is the "conventional form of expressing the inner product. Direct mechanization of this equation defines 'lumped' arithmetic computation" [12] as this calculation adds and multiplies the partial product of different shifts altogether before it is summed up to give a final result. By expanding equation (3) it yields the following equations:

$$\begin{aligned} y = &-\sum_{k=1}^{K} (b_{k0} \bullet A_k) + \sum_{k=1}^{K} \left[(A_k \bullet b_{k1}) 2^{-1} + (A_k \bullet b_{k2}) 2^{-2} + \cdots + (A_k \bullet b_{k(N-1)}) 2^{-(N-1)} \right] \\ y = &-\left[b_{10} \bullet A_1 + b_{20} \bullet A_2 + \cdots + b_{K0} \bullet A_K \right] \\ &+ \left[(b_{11} \bullet A_1) 2^{-1} + (b_{12} \bullet A_1) 2^{-2} + \cdots + (b_{1(N-1)} \bullet A_1) 2^{-(N-1)} \right] \\ &+ \left[(b_{21} \bullet A_2) 2^{-1} + (b_{22} \bullet A_2) 2^{-2} + \cdots + (b_{2(N-1)} \bullet A_2) 2^{-(N-1)} \right] \\ &\vdots \\ &+ \left[(b_{K1} \bullet A_K) 2^{-1} + (b_{K2} \bullet A_K) 2^{-2} + \cdots + (b_{K(N-1)} \bullet A_K) 2^{-(N-1)} \right] \end{aligned} \tag{4}$$

By regrouping each liked terms, the following sets of equations can be generated:

$$\begin{aligned} y = &-\left[b_{10} \bullet A_1 + b_{20} \bullet A_2 + \cdots + b_{K0} \bullet A_K \right] \\ &+ \left[(b_{11} \bullet A_1) + (b_{21} \bullet A_2) + \cdots + (b_{K1} \bullet A_K) \right] 2^{-1} \\ &+ \left[(b_{12} \bullet A_1) + (b_{22} \bullet A_2) + \cdots + (b_{K2} \bullet A_K) \right] 2^{-2} \\ &\vdots \\ &+ \left[(b_{1(N-1)} \bullet A_1) + (b_{2(N-1)} \bullet A_2) + \cdots + (b_{K(N-1)} \bullet A_K) \right] 2^{-(N-1)} \\ y = &-\sum_{k=1}^{K} (b_{k0}) \bullet A_k + \sum_{n=1}^{N-1} \left[b_{1n} \bullet A_k + b_{2n} \bullet A_2 + \cdots + b_{Kn} \bullet A_K \right] 2^{-n} \end{aligned} \tag{5}$$

Consequently, by interchanging the order of summation, a more distributed arithmetic computation can be obtained.

$$y = \sum_{k=1}^{K} A_k (-b_{k0}) + \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} \tag{6}$$

As a result, in DA, the multiplication block is eliminated such that the arithmetic computation becomes discrete throughout the structure and not in a combined form [20] [21] [22]. With this equation it can be seen that the partial products of equal shifts are being added before being summed to the next partial product shift.

2.1 ROM-based Construction

ROM-based DA speeds up the multiplication process by pre-computing all possible values and storing them in the LUT ROM. The bits of input data $\{x_{0k}, x_{1k}, \ldots x_{i,k}\}$ are used to form the ROM address directly in which an arithmetic accumulator feedback (scalar shift accumulator) is used to form successive scaling and shifting by the power of two. Multiplication by a power of two is no more than a bit shift and since this structure is being designed to multiply fixed point numbers, multiplication by a power of two will be no more than a bit shift to the right [17] [18] [20]. Take into consideration the square bracketed term from equation (6) shown below as equation (7).

$$\left[\sum_{k=1}^{K} A_k b_{kn}\right] \quad (7)$$

Since b_{kn} has two possible values, either 0 or 1, b_{kn} may only have 2^K possible values. Therefore, from equation (7), these pre-calculated values can be stored in a LUT of 2^K word addresses. These 2^K word addresses are used to directly access the memory location of the LUT ROM that contains the pre-computed result to that address [17] [20]. Since the constant coefficients A_k are known and b_{kn} values are 2's complement, either 0 or 1, each vector dot product is a result of the combination of each constant coefficient stored in the ROM.

Evaluating equation (6), in order to accommodate the negative term of the first summation, one more address line needs to be added to the LUT called T_s. The result is a ROM size of 2^{K+1} where T_s is a control timing signal and equal to one during the sign bit time, but otherwise zero. This bit is very significant as it is required to determine when the final result is completed [17]. Figure 1 shown below is a 3-tap 2^{K+1} DA-based implementation FIR filter

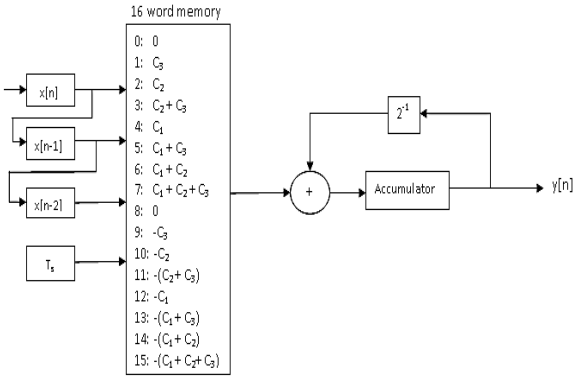


Figure 1: 3-tap 2^{K+1} DA-based implementation FIR filter

The size of the ROM however, can be reduced in three ways. This can be accomplished by the use of a ROM decomposition, by modifying the adder to an adder/subtractor, and by the use of offset binary coding (OBC).

2.2 ROM Decomposition

As previously described, the size of the ROM increases rapidly according to the number of K-tap filters, however, these ROMs can be divided into smaller units of DA-LUT in which their output sums from each unit can be added to give the final result. In other words, equation (6) can be broken into smaller "m" K-tap DA-based filters [19].

The total memory bank is now equal to m x 2^K where m is the number of DA-LUT units and K is the number of tap filters. For example, a 128-tap filter requires a LUT with 2^{128} memory bank. This memory bank, however, can be broken into 32 smaller DA-LUTs with 4-input taps for each unit allowing the memory component to decrease from 2^{128} to 32 x 2^4 which would just comprise of only 512 memory entries [19] [20]. Figure 2 shows the implementation of a ROM decomposition 4-tap DA-based filter with m = 2 and K =2. The output y[n], however, is only available after the N + [$\log_2$ (m)] clock cycle. The additional logarithmic term is for the adder tree as shown in the figure below [19].

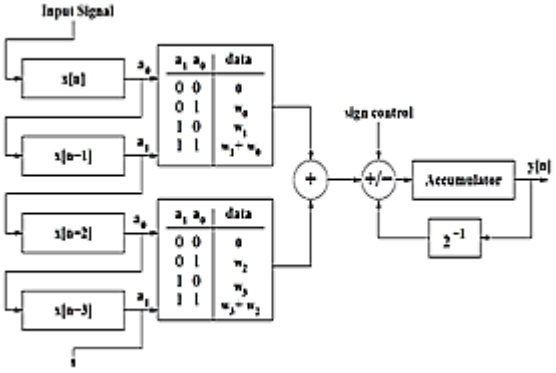


Figure 2: 4-tap DA-based filter with m = 2 and K =2

2.3 Modifying the Distributed Arithmetic to an Adder/Subtractor

In order to reduce the ROM size, certain aspects are needed to take into consideration. One such aspect is to modify the adder to an adder/subtractor allowing the memory size to be reduced to half of its size from 2^{K+1} to 2^K word ROM. This method is accomplished by using T_s, as the add/subtract-control line and not as a direct input into the LUT [17] [22]. Figure 3 shows the modified 4-tap 2^K DA-based implementation FIR filter.

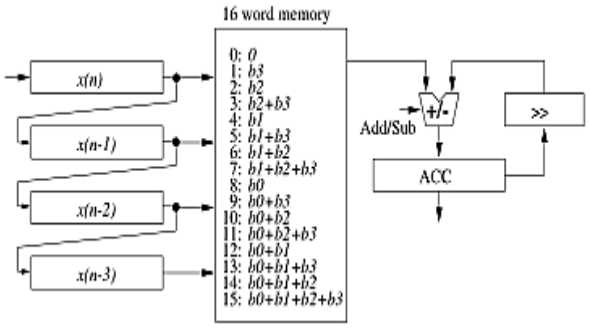


Figure 3: 4-tap 2^K DA-based implementation FIR filter.

2.4 Distributed Arithmetic using Offset Binary Coding (OBC)

It is possible to reduce the memory size from 2^K words to 2^{K-1} words by simply employing additional logics within its architectural structure. In order to do this, the input variable data are being read as an offset binary code of (-1, 1) and not in the usual conventional binary code form of (0, 1) [17] [22].

In analyzing the sum of product (SOP) from equation (1), which is shown below,

$$y = \sum_{k=1}^{K} A_K x_K$$

recall that x_k is the 2's complement input variable data, therefore, in order to cast the data into an offset binary code of (-1, 1), x_k can be rewritten as shown below.

$$x_k = \frac{1}{2}[x_k - (-x_k)] \tag{8}$$

The negative x_k term can also be expressed in the 2's complement form (1's complement plus 1) from equation (2) where the symbols with an overbar are the complements of its bit.

$$-x_k = -\bar{b}_{k0} + \sum_{n=1}^{N-1} \bar{b}_{kn} 2^{-n} + 2^{-(N-1)} \tag{9}$$

Rewriting equation (8) by combining equation (2) with equation (9) and regrouping yields the following equation.

$$x_k = \frac{1}{2}\left[-\left(b_{k0} - \bar{b}_{k0}\right) + \sum_{n=1}^{N-1} \left(b_{kn} - \bar{b}_{kn}\right) 2^{-n} - 2^{-(N-1)} \right] \tag{10}$$

Now the offset binary code c_{kn} can be defined as follows with c_{kn} being explicated as shown below.

$$c_{kn} = \begin{cases} b_{kn} - \bar{b}_{kn} & ,n \neq 0 \\ -(b_{kn} - \bar{b}_{kn}), n = 0 \end{cases} \quad where \quad c_{kn} \in \{-1, 1\} \tag{11}$$

Equation (10) can now be rewritten as seen in the equation below.

$$x_k = \frac{1}{2}\left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right] \tag{12}$$

Now by replacing x_k from equation (12) into the sum of products equation (1) yields equation (13).

$$y = \frac{1}{2} \sum_{k=1}^{K} A_k \left[\sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right] \tag{13}$$

Interchanging the order of summation produces equation (14), in which a more distributed arithmetic computation for the offset binary code implementation may be obtained.

$$y = \sum_{n=0}^{N-1} \frac{1}{2} \sum_{k=1}^{K} A_k c_{kn} 2^{-n} - \frac{1}{2} \sum_{k=1}^{K} A_k 2^{-(N-1)} \tag{14}$$

For simplicity, the first inner summation part of equation (14) may be written as a variable $Q(b_n)$ as shown in equation (15)

$$Q(b_n) = Q(c_{1n} c_{2n} \cdots c_{Kn}) = \frac{1}{2} \sum_{k=1}^{K} A_k c_{kn} \tag{15}$$

The latter summation part of equation (14) is an initial condition constant register and this may be expressed as Q(0).

$$Q(0) = -\frac{1}{2} \sum_{k=1}^{K} A_k \tag{16}$$

Now equation (14) can be simplified to equation (16)

$$y = \sum_{n=0}^{N-1} Q(b_n) 2^{-n} + 2^{-(N-1)} Q(0) \tag{17}$$

Table 1 is an example of a 2^{K-1} 8-word DA-based ROM filter where K = 4, b_{kn} are the input data memory addresses and c_{kn} is the data set that is been cast as (-1, 1) instead of (0, 1). It can also be depicted that the top half of the table, color-coded red, is just a mirror image (inverse symmetry) of the

bottom half of the table, color-coded blue [22] and since Ts is not directory fed into the ROM LUT, only the top half of the table will be used. Figure 4 shows 4-tap 2^{K-1} DA-based implementation filter

Table 1: 2^{K-1} 8-word DA-based ROM filter where K = 4

b_{1n}	b_{2n}	b_{3n}	b_{4n}	c_{1n}	c_{2n}	c_{3n}	c_{4n}	Content
0	0	0	0	-1	-1	-1	-1	-1/2(C1 + C2 + C3 + C4) = Q(0)
0	0	0	1	-1	-1	-1	1	-1/2(C1 + C2 + C3 - C4) = Q(1)
0	0	1	0	-1	-1	1	-1	-1/2(C1 + C2 - C3 + C4) = Q(2)
0	0	1	1	-1	-1	1	1	-1/2(C1 + C2 - C3 - C4) = Q(3)
0	1	0	0	-1	1	-1	-1	-1/2(C1 - C2 + C3 + C4) = Q(4)
0	1	0	1	-1	1	-1	1	-1/2(C1 - C2 + C3 - C4) = Q(5)
0	1	1	0	-1	1	1	-1	-1/2(C1 - C2 - C3 + C4) = Q(6)
0	1	1	1	-1	1	1	1	-1/2(C1 - C2 - C3 - C4) = Q(7)
1	0	0	0	1	-1	-1	-1	-1/2(-C1 + C2 + C3 + C4) = -Q(7)
1	0	0	1	1	-1	-1	1	-1/2(-C1 + C2 + C3 - C4) = -Q(6)
1	0	1	0	1	-1	1	-1	-1/2(-C1 + C2 - C3 + C4) = -Q(5)
1	0	1	1	1	-1	1	1	-1/2(-C1 + C2 - C3 - C4) = -Q(4)
1	1	0	0	1	1	-1	-1	-1/2(-C1 - C2 + C3 + C4) = -Q(3)
1	1	0	1	1	1	-1	1	-1/2(-C1 - C2 + C3 - C4) = -Q(2)
1	1	1	0	1	1	1	-1	-1/2(-C1 - C2 - C3 + C4) = -Q(1)
1	1	1	1	1	1	1	1	-1/2(-C1 - C2 - C3 - C4) = -Q(0)

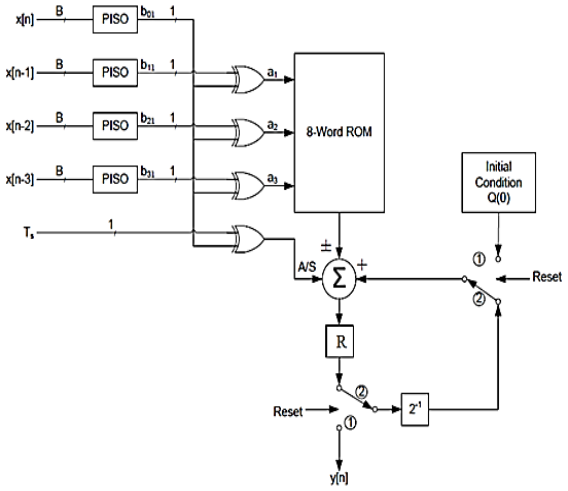


Figure 4: 4-tap 2^{K-1} DA-based implementation filters

3 Conversion of Complex Binary Number System

3.1 (-1 + j)-Based CBNS conversion algorithm

The binary complex number base (-1 + j) for fixed point numbers may be written in the form shown below,

$$a_0(-1+j)^0 + a_1(-1+j)^{-1} + \ldots\ldots + a_{n-2}(-1+j)^{-(n-2)} + a_{n-1}(-1+j)^{-(n-1)} \quad (19)$$

where "a" is a complex binary number (0 or 1) scaled such that | a | < 1. This equation is similar to the traditional base power series but instead of the base being in the power of 2, it is instead (-1+j) [1]. In order to convert from base-10 to base (-1 + j), we first have to convert the fixed point number (F) into its appropriate form such that its power can be expressed in terms of power of ½ as shown below [12] [13],

$$F = r_i = f_1 \bullet 2^{-1} + f_2 \bullet 2^{-2} + f_3 \bullet 2^{-3} + \ldots to\ computer\ limit \quad (20)$$

where r_i represents the remainder value and the coefficients of f_i are binary numbers, either 0 or 1. According to Jamil and Ali, the steps to convert the complex number algorithm to CBNS fraction are as follows [12]:

First step:

If $2r_0 - 1 < 0$ then $f_1 = 0$ and set $r_1 = 2r_0$
or if $2r_0 - 1 \geq 0$ then $f_1 = 1$ and set $r_1 = 2r_o - 1$

Second:
If $2r_i - 1 < 0$ then $f_{i+1} = 0$ and $r_{i+1} = 2r_i$
or if $2r_i - 1 \geq 0$ then $f_{i+1} = 1$ and $r_{i+1} = 2r_i - 1$

Just like the traditional way of computing fixed point numbers, this procedure will continue until the remainder $r_i = 0$, which signifies that the fraction has been terminated or when the computer limitation has been attained [12] [13]. In order to represent 2^{-i} into its respective equivalent base of (-1+j), we substitute 2^{-i} according to the table below, where i = 4s + t, s is any positive integer, and $0 \leq t \leq 3$ [11] [12]. Table 3.1 shows the first four values of i and Table 3.2 shows the overall binary representation for any 2^{-i} power.

Table 2: Representations for the First Four 2^{-i} Values in Base (-1+j)

i	2^{-i}	Base (-1+j)
1	2^{-1}	1.11
2	2^{-2}	1.1101
3	2^{-3}	0.000011
4	2^{-4}	0.00000001

Table 3: Representations for all 2^{-i} Powers in Base (-1 + j)

t	Base(-1+j)
0	0.0…(8s – 1) zeros followed by 1
1	0.0…(8s – 1) zeros followed by 111
2	0.0…(8s – 1) zeros followed by 11101
3	0.0…(8s + 4) zeros followed by 11

4 Results

4.1 Power Consumption and Time within the DA approach

In comparing the power consumption between the modified adder/subtractor DA technique with the traditional MAC approach, a 3-tap DA based filter was designed and implemented. The DA consumes 0.118 mW compared to 0.261 mW for the MAC approach which signifies that the

DA approach uses far less power consumption than the traditional MAC technique by approximately 55%. Also, in terms of time, the maximum time delay for the DA approach was far less than that of the MAC approach by approximately 52%. The maximum time delay for the DA approach was 8.775 ns while for the traditional approach it was 18.237ns. A 3-tap DA-OBC was also implemented and its power consumption amount was 0.340 mW but this is due to the extra logic that was needed to implement this design without the need of any multipliers. These techniques will be tested on a larger scale for computing the FFT structure and further be analyzed to see which technique will give the overall best power and time consumption.

4.2 Savings between the CBNS and FFT Arithmetic Computation

Given the review on CBNS, above, this subsection will focus on how efficient the CBNS algorithm will be in computing the radix-2 FFT calculations. With the CBNS represented as a single unit, this will reduce the arithmetic computations as seen in Table 4.

Table 4: N-point comparison between DFT, Radix-2 FFT, and Radix-2 CBNS FFT

Number of Points	Direct computation of DFT		Radix-2 FFT		Radix-2 CBNS FFT	
	Real Multipliers	Real Adders	Real Multipliers	Real Adders	Real Multipliers	Real Adders
4	32	24	16	24	4	8
8	128	112	48	72	12	24
16	512	480	128	192	32	64
64	8192	8064	768	1152	192	384
256	131072	130560	4096	6144	1024	2048
1024	2097152	2095104	20480	30720	5120	10240

From Table 4 shown above, it is shown that the radix-2 CBNS FFT uses less multiplier hardware structures by 75% and adder by approximately 67% in comparison to the radix-2 FFT. It also shows that the radix-2 CBNS FFT uses less arithmetic computations than the DFT.

5 Work in Progress

The flow chart shown in Figure 5 demonstrates how the CBNS N-point FFT will be implemented using the DA approach. Recall that a DA architecture is bit serial in nature and computes the SOP between two vectors: fixed and variable input data. In implementing this structure first, the fixed point complex numbers are converted into the (-1+j)-base representation. After its conversion, this new varying input data now become x[n] and the twiddle factor $W_N = e^{-j\frac{2\pi}{N}}$ now becomes the fixed constant coefficient value.

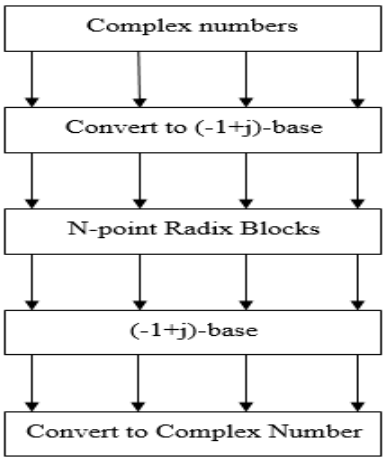


Figure 5: CBNS N-point FFT Implementation

These data now become the driving input data for the butterfly structure. For each m-stage of the butterfly structure, the twiddle factor will be loaded into the LUT for each stage where the DA structure will calculate the partial products of equal shifts, which are then added before being summed to the next partial product shift. This process will continue for each m-stage until y[n] contains the final result. The final results will be outputted into the (-1+j)-base representation which can then be reconverted using the algorithm described in section 3.1.

In this proposed design, the three DA Rom reductions discussed in sections II will be used in implementing the FFT structure using the DA approach. The outcome hardware structure will have zero multipliers which is a 100% reduction as shown in Table 5, as no multipliers will be used in computing the FFT structure. The multiplication process is done by shifting and adding only. The total number of real adders, however, will change slightly as the total adders that are needed to design the DA and FFT structure will be combined. The truth table for CBNS employed for the adder/subtraction technique and the (-1+j)-base structure, will be used instead of the conventional approach for adding and subtracting. Currently, this design for implementing the FFT algorithm, using the CBNS and the DA approach algorithm is in its initial stage of development.

Table 5: Proposed DA CBNS FFT Structure

Number of Points N	Radix-2 CBNS FFT Real Multipliers	Radix-2 CBNS FFT Real Multipliers
4	4	0
8	12	0
16	32	0
64	192	0
256	1024	0
1024	5120	0

6 Conclusion

For a DIT butterfly structure, in calculating the radix-2 FFT algorithm, the number of arithmetic computations for each butterfly structure will consist of four multiplications and six additions. In integrating the CBNS within the FFT algorithm, the arithmetic computations for each butterfly structure will be greatly reduced, due to the fact that each arithmetic operation is not based on the divide and conquer method as each complex number is represented as one single unit. Therefore, by treating each complex number as one entity, the arithmetic operations for each butterfly structure will consist of only one multiplication and two additions/subtractions. This will result in the number of "N" point calculations reduced by 75% for the multipliers and approximately 67% for the adder/subtractor. This proposed method will reduce the multiplier complexity by 100% and decrease the cost for implementing this structure compared to traditional ways of computing the algorithms. Given that there is a great demand for new and improved technology for DSP applications, in calculating the SOP, the methods discussed in this paper will be used in implementing the new radix-2 and radix-4 FFT structures employing DA. These three new methods will be compared with the traditional method to investigate which method will give the overall maximum performance in terms of its area size, speed, and timing

7 References

[1] T. Jamil, "The complex binary number system,"IEEE Potentials, vol. 20, no. 5, pp. 39-41, 2002.

[2] R. G. Lyons, "FFT Software Programs," in Understanding Digital Signal Processing,Prentice Hall, 2004.

[3] N. Govil and S. R. Chowdhury, "High performance and low cost implementation of Fast Fourier Transform algorithm based on Hardware Software co-design," 2014 IEEE Region 10 Symposium, pp. 403-407, 2014.

[4] R. Lyons, "Relationship of the FFT to the DFT," in Understanding Digital Signal Processing, 2nd ed., Prentice Hall, 2004.

[5] R. Guo, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," IEEE transaction on Circuits and System ll, vol. 58, no. 9,pp. 600-604, 2011.

[6] M. Jiang, B. Yang, R. Huang, T. Y. Zhang and Y. Y. Wang, "A multiplierless fast fourier transform architecture," Electronis Letters, vol. 43, no. 3, pp. 191-192, 2007.

[7] V. K. Sharma, K. K. Mahapatra and U. C. Pati, "An efficient distributed arithmetic based VLSI architecture for DCT," International Conference on Devices and Communications, pp. 1-5, 2011.

[8] W. Penney, "A numerical system with a negative base," Mathematical Student Journal, pp. 1-2, 1964.

[9] W. Penney, "A binary system for complex numbers," Journal of the ACM, vol. 12, no. 2, pp. 247-248, 1965.

[10] H. Zaini and R. G. Deshmukh, "A novel method for arithmetic operations using complex binary number system and the reconversion of the result to the decimal complex number system," Proceedings of the IEEE SouthestCon 2003, pp. 31-37, 2003.

[11] T. Jamil, "Impact of shift operations on (-1+j)-base complex binary numbers," Journal of Computers, vol. 3, no. 2, pp. 63-71, 2008.

[12] T. Jamil and U. Ali, "Effects of multiple-bit shift-right operations on complex binary numbers," Proceedings of IEEE SoutheastCon 2007, pp. 759-764, 2007.

[13] D. C. Blest and T. Jamil, "Efficient division in the binary representation of complex numbers," Proceedings of the IEEE SoutheastCon 2001, pp. 188-195, 2001.

[14] T. Jamil, "An introduction to complex binary number system," Fourth International Conference on Information and Computing, pp. 229-232, 2011.

[15] T. Jamil, N. Holmes and D. Blest, "Towards implemention of a binary number system for complex numbers," Proceedings of the IEEE SoutheastCon 2000, pp. 268-274, 2000.

[16] J. Goode, T. Jamil and D. Callahan, "A simple circuit for adding complex numbers," WSEAS Transactions on Information Science and Applications, vol. 1, no. 1, pp. 61-66, 2004.

[17] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutor review," IEEE ASSP Magazine, pp. 4-19, 1989.

[18] S. Ramprasad, N. R. Shanbhag and I. N. Hajj, "Low-power distributed arithmetic architectures using non-uniform memory partitioning," IEEE International Symposium on Circuits and Systems, vol. 3, pp. 470-473, 1999.

[19] W. Huang, "Implementation of adaptive digital FIR and reprogrammable mixed-signal filters using distributed arithmetic," PhD Thesis, Dept. Elect. & Comput. Eng., Georgia Institute of Tech., Atlanta, 2009.

[20] R. Guo and L. S. DeBrunner, "A novel adaptive filter implementation scheme using distributed arithmetic," *Signals, System and Computers*, pp. 160-164, 2011.

[21] R. M. Jiang, "An area-efficient FFT architecture for OFDM and digital video broadcasting," IEEE Transactions on Consumer Electronics, vol. 53, no. 4, pp. 1322-1326, 2007.

[22] S. Chandrasekaran and A. Amira, "Novel sparse OBC based distributed arithmeticarchitecture for matrix transforms," IEEE International Symposium on Circuits and Systems, pp. 3207-3210, 2007.

Evaluation of an FPGA-Based Shortest-Path-Search Accelerator

Yasuhiro Takei, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {takei, hariyama, kameyama}@ecei.tohoku.ac.jp

Abstract—*Shortest-path search over large scale graphs is widely used in various applications. However, shortest path algorithms such as Dijkstra's algorithm include complex processing. It is difficult for accelerators such as GPUs to accelerate these algorithms efficiently. This paper presents an FPGA-based accelerator with SIMD architecture for the shortest-paths algorithm. In the proposed architecture, processing of the Dijkstra's algorithm is done with a high degree of parallelism, and the memory usage is reduced by the replacement of the node data. According to the evaluation, the processing time of the proposed architecture is about a half of that of a CPU, and the amount of the node data stored in on-chip memory is about one-third of all nodes when the input graph is a lattice graph.*

Keywords: Shortest-path search, Dijkstra's algorithm, Single instruction multiple data (SIMD), FPGA

1. Introduction

Recently, there is a huge demand of processing large-scale graphs. Especially, finding the shortest-path in large scale graphs is used in many applications such as traffic simulation, social networking service and bioinformatics. To solve the shortest-path problem, various algorithms has been proposed. Dijkstra's algorithm [1] and Bellman-Ford algorithm [2] were proposed to solve the single-source shortest-path problem (SSSP). Warshall-Floyd Algorithm [3] was proposed to solve the all-pair shortest-path problem (APSP).

To accelerate processing speed of solving shortest-path problem, there have been many software-based studies in terms of improving a data structure and reducing a computational amount. In order to process the shortest-path problem for large scale graphs, PC clusters with many CPUs are often used [4] because of their large memory capacity. However, these computing systems need very large space and power consumption.

Some studies used GPUs for solving shortest-path problem. Harish [5] and Katz [6] have implemented shortest-path search on the GPU. GPUs are suitable for simple and parallelized processing. However, it is difficult to accelerate shortest-path searching efficiently when the shortest-path algorithm includes serial and complex data-flows.

Other studies used the FPGA-based accelerator for solving shortest-path problem. FPGAs can implement application-specific data-paths by reconfiguration after fabrication. Moreover, the power consumption of FPGAs are less than one-tenth of that of CPUs and GPUs. Tommiska [7], Fernandez [8], and Sridharan [9] have designed the FPGA-based architecture for SSSP with the Dijkstra's algorithm. Bondhugula [10] has designed the FPGA-based architecture for APSP with the Warshall-Floyd algorithm. However, their works did not consider processing large-scale graphs since the memory usage of the input graph is not considered.

To solve these problems, we design an FPGA-based accelerator for the Dijkstra's algorithm on large scale graphs. In order to accelerate processing and memory access, we design the SIMD (single instruction multiple data) architecture. We explain how to search the shortest path with a high degree of parallelism, and how to replace the node data on a limited memory space. In this paper, we implement the improved architecture from our previous work [11], and we evaluate the memory usage and processing time of the shortest path search.

2. Dijkstra's algorithm and its implementation on an FPGA

The Dijkstra's algorithm is one of the most popular algorithms to solve SSSP. Because it is easy to implement, this algorithm is used in various applications such as analysis of the internet, traffic simulation and so on. Let S be the node where we are starting. Let

$d(y)$ be the distance from S to node y. The flow of the Dijkstra's algorithm is represented by the following steps.

Step1: Assign to every node a tentative distance: set it to zero for S, and to infinity for all other nodes. Mark all nodes *"unvisited"*.

Step2: Select the *unvisited* node which has the smallest tentative distance and make it the *"current node"*.

Step3: For the *current node*, consider all of *unvisited* neighbor nodes and update their tentative distance. If the *current node* is A , and one of the *unvisited* neighbor node is B, set the tentative distance of B ($td(B)$) to $\min(td(B), d(A) + l_{AB})$,where l_{AB} is the length of the edge between A and B. When considering all of *unvisited* neighbor nodes of the *current node*, mark the *current node "visited"*.

Step4: Until all nodes are marked *visited*, go back to Step2.

The processing time of the Dijkstra's algorithm depends on searching the minimum distance in Step2 and updating tentative distances in Step3. In these processing, there are many comparison operations on multiple node data. Hence, a parallelized architecture such as the SIMD architecture is suitable for accelerating the processing of the Dijkstra's algorithm.

Since tentative distances and paths are read and updated frequently, on-chip memory on an FPGA is suitable for storing these data. However, the capacity of the on-chip memory is small. The memory management is required for reducing the on-chip memory usage and the total processing time. In the Dijkstra's algorithm, tentative distance of nodes that connects current or visited nodes is only used in the processing. As shown in Fig.1, the current node (C) and unvisited nodes connected to current or visited nodes (D,E) are only used in the processing until the next current node is determined. As shown in Fig.2, after the next current node (D) is determined, a previous current node (C) data is unnecessary in the processing. Hence the memory space for the previous current node data (C) can be reused for the new node data (F).

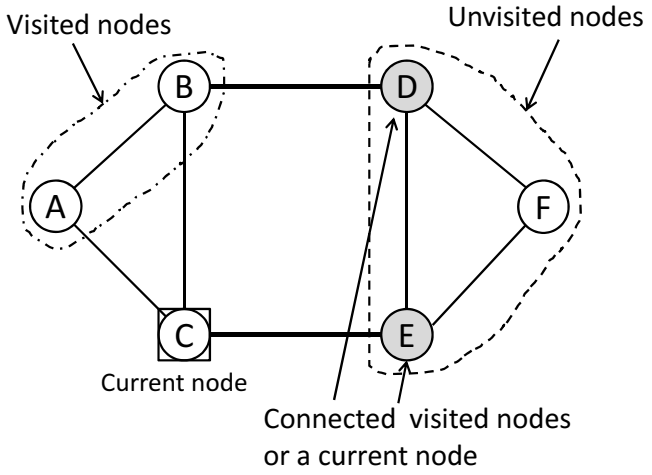


Fig. 1: The current node (C) and unvisited nodes connected to current or visited nodes (D,E)

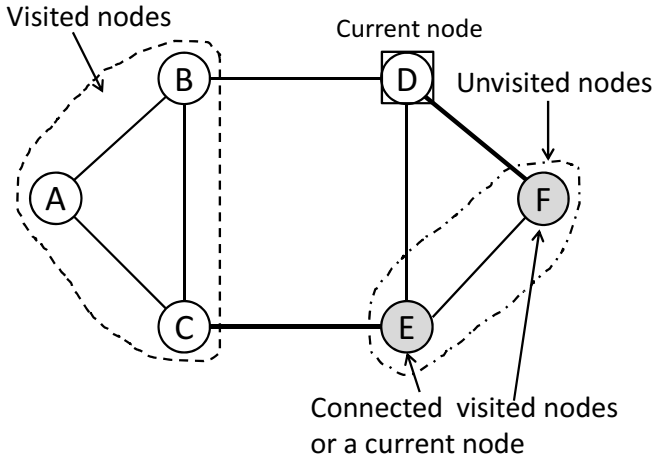


Fig. 2: The current node (D) and unvisited nodes connected to current or visited nodes (E,F)

3. Architecture

Figure 3 shows the overall architecture. This architecture consists of an external memory, a CPU core and a Dijkstra module. An external memory such as a DDR2 SDRAM stores the adjacency list of the input graph. The Dijkstra module consists of processing elements (PEs), selectors, a decoder, a current node register, and an address generation unit (AGU). The current node register stores the current node number and the distance from the start to the current node.

Figure 4 shows the architecture of the processing element in the Dijkstra module. This architecture consists of a node memory, modules for searching for

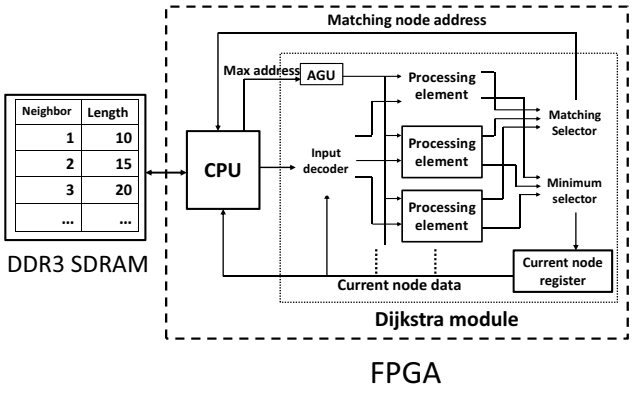


Fig. 3: Overall architecture

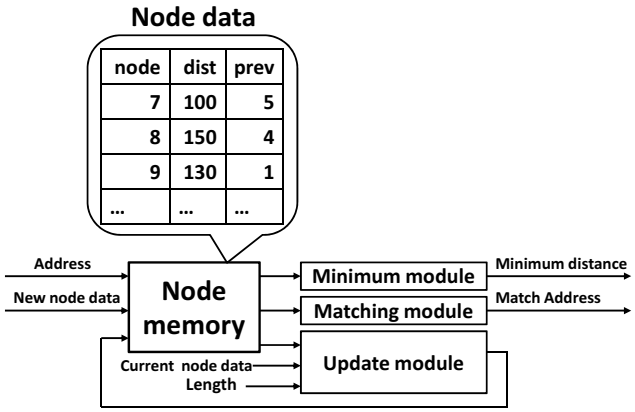


Fig. 4: Architecture of the PE

minimum distance, for matching the node number and for updating the tentative distance. These modules consist of comparators, registers and adders. The node memory stores the values of node number, the tentative distance and the previous node number of the shortest path.

For updating the tentative distances in node memories, the neighbor node number is searched by matching node modules in parallel as shown in Fig.5. In this case, the node number 8 is searched for. Then the tentative distance at node 8 is compared with the sum of the distance at the current node 6 and the length of the edge 6 to 8 by the update module. If the sum of the distance at the current node and the length the edge is smaller than the tentative distance, the tentative distance and the previous node number are updated as shown in Fig.6.

For the searching for the minimum distance in node memories, minimum distance modules and minimum selector are connected as shown in Fig.7, and searching in parallel. In this case, node 7 has the minimum distance, and node 7 is selected as the new current distance. When the minimum distance searching is completed, the new minimum distance and the node number are stored in the current node register.

After these data are stored in the current node register, the memory space for the current node can be overwritten to the first unvisited neighbor node data that have not existed in node memories as shown in Fig.8. If the number of these new neighbor nodes is more than two, the empty space in node memories is used for storing subsequent unvisited neighbor node data.

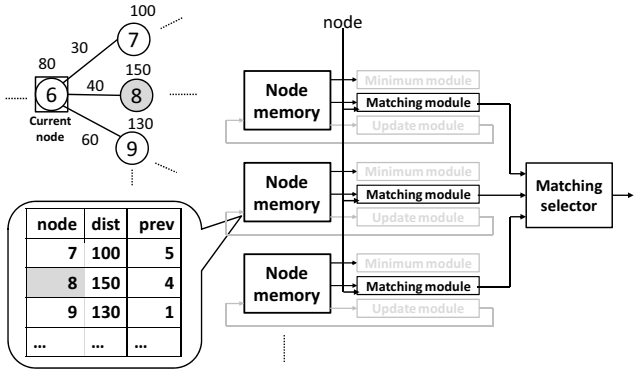


Fig. 5: Searching a node number data in node memories

4. Evaluation of the proposed architecture

In this evaluation, we use the Terasic DE4 FPGA board [12]. This board includes an Altera Stratix IV GX EP4SGX530, and a DDR2 SDRAM (4GB). Altera Quartus 13.1 is used for the FPGA implementation. For a proto-type design, we implement the proposed architecture with 8 PEs. 4,096 nodes can be processed in the implemented architecture. NiosII soft-core processor [13] is used for the CPU core as shown in Fig.3. It is designed using Altera Qsys 13.1 and programmed by C language using Nios II EDS 13.1. Table 1 shows the resource usage of the proposed accelerator. The usage of memory bits of NiosII is larger than that of the Dijkstra module since the programming code for storing the graph is large. Considering with the resource usage of the on-chip memory, about 300,000 nodes can be implemented on the FPGA board if the programming code on the NiosII is improved.

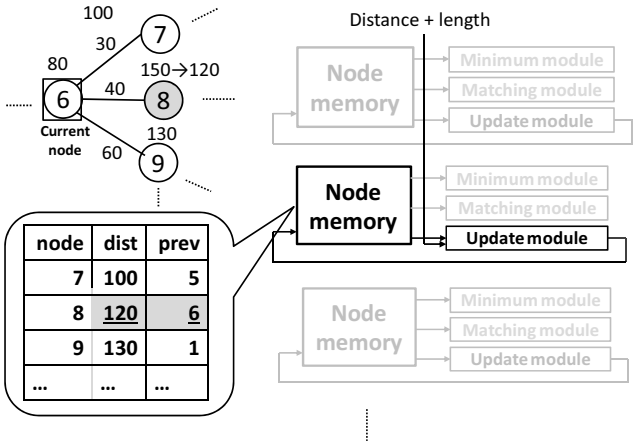


Fig. 6: Updating node data in the node memory

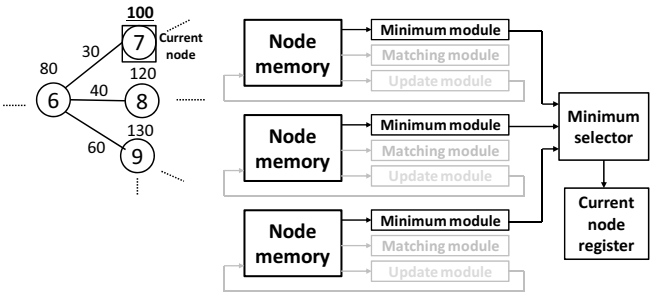


Fig. 7: Searching the minimum distance in node memories

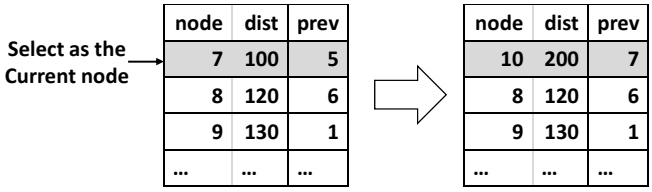


Fig. 8: Overwriting the current data to a new node data in the node memory

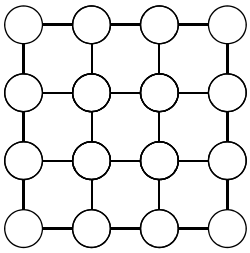


Fig. 9: Lattice graph

Let us compare the performance of the FPGA-based accelerator with that of Intel Core2 Quad processor. We implement the shortest-path problem on a lattice graph as shown in Fig.9. The Dijkstra's algorithm is implemented on the Core2 Quad processor by using C++ language. Microsoft Visual studio 2010 is used for compiling. Table 2 shows the processing time comparison. The processing time of the proposed architecture is about half of the CPU. The performance of the proposed architecture can increase when more PEs are implemented on the FPGA. Moreover, FPGAs with hard-core CPUs, such as Xilinx Zynq [14] and Altera Cyclone V SoC [15] can be used in order to reduce the control overhead. These FPGAs includes multicore CPUs such as the ARM Cortex-A9. The performance of these CPU cores is more than ten times as much as that of the NiosII core.

Let us consider the memory usage of the node memories. Table 3 shows the number of node data in node memories in the processing of the Dijkstra's algorithm. According to the ratio of nodes in the node memories to all nodes in the graph, the memory usage is about one-third of the all nodes data when the input graph is a lattice graph. As a result, about three times nodes of the node memories can be processed on the FPGA board when the input graph is as sparse as the lattice graph, such as a map data. Memory usage depends on a structure of the input graph. When the input graph is sparse, usage of the node memories becomes small. On the other hand, usage of the node memories becomes at maximum when the start node connects to all other nodes. In this case, (All nodes - 1) nodes are stored in the node memories.

5. Conclusions

We have proposed an FPGA-based accelerator for shortest-path search. We designed for processing the Dijkstra's algorithm in parallel and we explained the replacement of the node data to reduce the memory usage. According to the evaluation, the processing time of the proposed architecture is about half of the CPU, and the usage of the on-chip memory is about one-third of the all nodes when the input graph is a lattice graph.

The suitable architecture of the shortest-path-search accelerator depends on the structure of a input graph. The proposed architecture is suitable for sparse graphs. However, this architecture is not suitable for dense graphs since the usage of the node memories cannot reduce so mach. Hence, we should design another

Table 1: Resource usage

	LUT	Register	Memory bit	DSP
Dijkstra module	1509	607	196608	0
NiosII (CPU)	9298	12507	1832318	4

Table 2: Processing time(ms)

Input graph	FPGA (50MHz)	Core2 Quad (2.83GHz)
1024Nodes, 3968Edges	9.38	16.00
4096Nodes, 16130Edges	50.40	94.00

architecture for dense graphs, such as the GPU-like architecture for processing an adjacency matrix.

Moreover, to process a very large scale graph, it is important to reduce the bottleneck of the data-transfer of the graph data from the external storages such as SSDs to FPGA boards. We are going to implement the framework for graph compressing such as WebGraph [16] or Graphillion [17] on the FPGA board.

Acknowledgement

This work is supported by JSPS KAKENHI grant number 24300013 and Grant-in-Aid for JSPS Fellows grant number 15J04973.

References

[1] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, 1(1): pp.269–271, 1959.

[2] R. Bellman, "On a Routing Problem", Technical report, DTIC Document, 1956.

[3] R. W. Floyd, "Algorithm 97: Shortest Path", Commun. ACM, 5(6) pp.345–346, June 1962.

[4] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. "Pregel: a System for Large-Scale Graph Processing", In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 135–146, 2010.

[5] P. Harish, and P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA", High performance computing-HiPC 2007, Springer, pp.197-208, 2007.

[6] G. J. Katz and J. T. Kider Jr, "All-Pairs Shortest-Paths for Large Graphs on the GPU", In Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, 2008.

[7] M. Tommiska and J.Skytta. "Dijkstra's Shortest Paths Algorithm in Reconfigurable Hardware", In Proc. Field Programmable Logic and Applications, pp. 653–657, 2001.

[8] I. Fernandez, J. Castillo, C. Pedraza, C. Sanchez, and J. I. Martinez, "Parallel Implementation of the Shortest Path Algorithm on FPGA" In Proc. 4th Southern Conf. on Programmable Logic., pp. 245–248, 2008.

[9] K. S. T.K.Priya and P. Kumar, "Hardware Architecture for Finding Shortest Paths", In Proc. IEEE Region 10 Conf., pp. 1–5, 2009.

Table 3: The number of node data in node memories

All nodes in a graph	256	1024	4096
Nodes data in node memories	87	316	1329
Ratio of the node data	0.340	0.309	0.324

[10] U. Bondhugula, A. Devulapalli, J. Fernando, P. Wyckoff, and P. Sadayappan, "Parallel FPGA-Based All-Pairs Shortest-Paths in a Directed Graph", In Proceedings of Parallel and Distributed Processing Symposium, 2006.

[11] Y. Takei, M. Hariyama and M. Kameyama, "An SIMD Architecture for Shortest-Path Search and Its FPGA Implementation ", International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pp.53–56, 2014

[12] Terasic, "Altera DE4 Development and Education Board", http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=EnglishNo=501.

[13] Altera, "Nios II Processor", http://www.altera.com/devices/processor/nios2/ni2-index.html.

[14] Xilinx, "Zynq-7000 All Programmable SoC", http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm.

[15] Altera, "Cyclone V SoCs: Lowest System Cost and Power", http://www.altera.com/devices/processor/ soc-fpga/cyclone-v-soc/cyclone-v-soc.html.

[16] P. Boldi and S. Vigna. ,"The Webgraph Framework I: Compression Techniques". In Proc. of the 13th international conference on World Wide Web, pp. 595–602. ACM, 2004.

[17] T. Inoue, H. Iwashita, J. Kawahara, and S. Minato. "Graphillion: Software Library for Very Large Sets of Labeled Graphs", International Journal on Software Tools for Technology Transfer, 2014.

A Scalable Parallel Bisection Algorithm for Symmetric Tridiagonal Eigenvalue Problem

Barok Imana
Department of Engineering
Trinity College
Hartford, CT 06106 USA
Email: barok.imana@trincoll.edu

Peter Yoon
Department of Computer Science
Trinity College
Hartford, CT 06106 USA
Email: peter.yoon@trincoll.edu

Abstract— *Bisection method is a numerically stable algorithm used to find the eigenvalues of symmetric tridiagonal matrices. It is distinct from other methods in that it can be used to compute a subset of eigenvalues with high accuracy. However, the algorithm is significantly slow compared to other methods when a large number of eigenvalues are desired. Fortunately, the algorithm exhibits a high level of parallelism when it is implemented on various types of multiprocessors, including a single-GPU system. In this paper, we describe a highly scalable implementation using multi-GPU systems to accommodate large matrices. Our approach exploits the latest memory management features available on Nvidia Tesla K20c GPUs, including unified memory architecture, peer-to-peer data transfer, and dynamic parallelism. Our implementation was at least 60 times faster than a multi-core CPU system and exhibits a linear speedup with respect to the number of GPUs in the system.*

Keywords: Symmetric eigenvalue problem, bisection method, parallel algorithm, general-purpose GPU computing

1. Introduction

Consider an $n \times n$ symmetric tridiagonal matrix T,

$$T = \begin{pmatrix} a_1 & b_1 & & & & \\ b_1 & a_2 & b_2 & & & \\ & b_2 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & & b_{n-1} & a_n \end{pmatrix}$$

Suppose that the *eigenvalues* of T are ordered so that

$$\lambda_1 \leq \lambda \leq \cdots \leq \lambda_n$$

where each scalar $\lambda_i, i = 1, 2, \ldots, n$, satisfies

$$Tx_i = \lambda_i x_i$$

for non-zero vector $x_i, i = 1, 2, \ldots, n$. Here, the vector x_i is called ith *eigenvector* of T associated with the eigenvalue λ_i.

The solution to symmetric tridiagonal eigenvalue problem is a critical part of computing eigenvalues and eigenvectors of general symmetric matrices, where the matrix is first reduced to symmetric tridiagonal form. In addition, the computation of eigenvalues and eigenvectors of symmetric tridiagonal matrices arises in many important applications areas including structural dynamics, quantum chemistry, oceanography, economics theory and control process.

Several approaches to computing eigenvalues of symmetric tridiagonal matrices have been proposed, including the QR iteration [1], the divide and conquer method [5], [6], and the bisection method [8], [9], [11]. The QR iteration has been considered the most efficient method to compute all eigenvalues, only requiring $O(n)$ operations for a tridiagonal matrix, but, unfortunately, there are not efficient algorithms which are amenable to various parallel architectures.

The divide and conquer method is the fastest now available if all eigenvalues and eigenvectors of a symmetric tridiagonal matrix are desired. The method, implemented in LAPACK [12], begins with dividing the original matrix into two smaller symmetric tridiagonal matrices, computing the eigenvalues of the submatrices, and combining the computed eigenvalues using rank-one modifications [5]. Unlike the QR iteration, the method has been successfully implemented on several multiprocessors [6].

The bisection method may be used to find all eigenvalues or a subset of the eigenvalues, requiring only $O(nk)$ operations, where k is the number of eigenvalues desired. Since each eigenvalue can be computed independent of one another, the whole procedure can be made highly parallel. Several parallel implementations of the method have been proposed [9], [14]. In particular, as the use of general-purpose GPUs for scientific computing has been dramatically increasing in recent years, it has been noted that the GPUs have an enormous potential in computing eigenvalues using the bisection method.

The initial release of a GPU-accelerated bisection method has been included in Nvidia CUDA SDK [16]. This implementation utilizes simple yet highly efficient data structures for the division steps, but it suffers some issues such as overflow problems and its inability to deal with special structure

of the matrix. To that end, Volkov and Demmel [15] proposed an improved version which overcomes architectural limitations and delivers high performance and high accuracy of the eigenvalues. However, neither implementations are scalable to multiple GPUs in order to accommodate large matrices.

In this paper, we focus on a highly scalable implementation of the bisection method which will be amenable to multi-GPU systems. Our approach exploits the latest memory management features available on the Nvidia Tesla K20c GPU, including unified memory architecture, peer-to-peer data transfer, and dynamic parallelism. Our implementation of the bisection method, which is entirely done on multiple GPUs, runs at least 60x faster for large matrices, compared to multi-core CPU versions and exhibits a linear speedup with the number of GPUs used.

The remainder of the paper is organized as follows: Section 2 briefly describes the bisection algorithm, Section 3 presents a detailed parallel implementation, Section 4 presents the experimental results followed by discussions and future directions.

2. The Bisection Method

The main idea of the bisection method is based on Sylvester's Inertia Theorem [17], which states that the number of eigenvalues greater than λ is the same as the number of positive eigenvalues of $T - \lambda I$, that is, the matrix of the form,

$$\begin{pmatrix} a_1-\lambda & b_1 & & & & & \\ b_1 & a_2-\lambda & b_2 & & & & \\ & b_2 & a_3-\lambda & b_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & b_{n-2} & a_{n-1}-\lambda & b_{n-1} & \\ & & & & b_{n-1} & a_n-\lambda & \end{pmatrix}$$

Let Inertia(T) be the triplet (μ, ξ, π), where μ is the number of negative eigenvalues of T, ξ is the number of zero eigenvalues of T, and π is the number of positive eigenvalues of T. Then, the theorem states that

$$\text{Inertia}(T) = \text{Inertia}(U^T T U),$$

where U is nonsingular. Furthermore, if $T - \lambda I$ can be factorized to LDL^T, where L is nonsingular and D diagonal, then

$$\text{Inertia}(T - \lambda I) = \text{Inertia}(D).$$

which leads to Algorithm 1 which computes the number of negative elements of D, which is the same as the eigenvalues of T that are less than λ.

In Algorithm 1, the function Count(T, λ), given a real shift value λ, shifts matrix T by λ, and then performs LDL^T decomposition on the resulting matrix. By a careful examination of the function, one can show that the number of negative entries on the diagonal of D is the number

Algorithm 1 Count(T, λ)

```
1: count = 0
2: d = 1
3: for i = 1 to n do
4:     d = a_i − λ − b_i^2/d
5:     if d < 0 then
6:         count = count + 1
7:     end if
8: end for
```

of eigenvalues of T that are less than λ. Note that the function does not explicitly compute LDL^T decomposition. The bisection method uses Count(T, λ) function to calculate the total number of eigenvalues in interval (a, b), which is equivalent to Count(T, b) − Count(T, a).

The initial interval which contains all eigenvalues of T is given by Gerschgorin Circle Theorem [1] for symmetric tridiagonal matrix. The theorem states that all eigenvalues of T is bounded by the spectrum $\lambda(T)$, such that

$$\lambda(T) \in \bigcup [a_i - r_i, a_i + r_i]$$

where

$$r_i = b_i + b_{i-1}, i = 2, \ldots n-1$$
$$r_1 = b_1, r_n = b_{n-1}$$

Starting with a Gerschgorin interval, the bisection algorithm iteratively divides the interval into smaller subintervals. These subintervals are either discarded if they contain no eigenvalues of T, or continued being subdivided until they are sufficiently small, which is determined by a given tolerance. Finally, the eigenvalues of T are approximated by taking either bounds or midpoints of converged intervals.

3. Parallel Implementation

In this section, we describe a scalable parallel bisection algorithm and its implementation on a multi-GPU system. The use of GPUs in scientific computing applications has risen dramatically in recent years mainly because of their highly parallel architecture, energy efficiency and cost effectiveness. Unlike a CPU, a GPU consists of thousands of small computing cores designed to run tens of thousands of threads in parallel. Although each core may not be as powerful as a single CPU core, together, the thousands of cores produce higher throughput in applications that demand a high-level of data parallelism.

3.1 CUDA

One of the most important aspects in designing algorithms on GPUs is the memory structure. GPUs reside on an external hardware and have no direct access to the main memory of the CPU. Data residing on the main memory of the CPU has to be copied to the GPUs via system bus

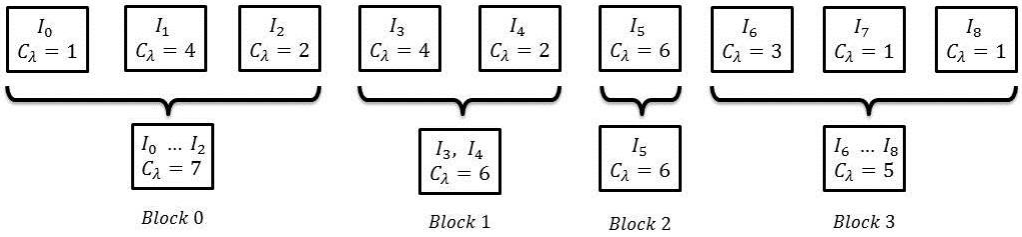


Fig. 1: Blocking for 2 GPUs with 8 threads per block

such as PCI Express. However, the PCI Express has limited bandwidth and poses a bottleneck in many applications that require frequent data transfer between CPUs and GPUs.

This is not the case for our implementation because most of our data is small enough to fit on the dedicated memory of individual GPUs. The only times we have to use the PCI Express channel is when loading input data on the GPUs and when transferring data between multiple GPUs. In the latter case, where data has to be copied to another GPU, the source GPU has to first copy the data to the main memory of the CPU and only then can the CPU can copy that data to another GPU. With newer GPUs such as Nvidia Tesla K20c, however, a direct data transfer between the GPUs, called *peer-to-peer*, is allowed to facilitate memory transfer more efficiently.

For our multi-GPU implementation of the bisection algorithm, we will be using CUDA, a parallel computing platform developed by Nvidia for general-purpose computing on GPUs [18]. In CUDA programming architecture, a CPU is referred to as a *host* as it launches a *kernel* on the GPU which is called the *device*. Once the kernel is completed, the device copies back the results back to the host. The kernel is run by means of *threads* organized into *blocks*. Each block retains a copy of the kernel, and the threads in the same block run the same code in parallel. Threads have their own local memory and have access to a shared memory space dedicated for the threads of the same block.

3.2 Single-GPU Implementations

Lessig [16] presented a single GPU implementation for the bisection algorithm. In this implementation, at every division stage, two kernels are launched: one for intervals containing exactly 1 eigenvalue, another for intervals containing more than 1. Their work was based on Tesla C870 with the peak throughput of 500 Gflops. Tesla K20c, on which this work is based, achieves 1.17 Tflops for double precision, and 3.52 Tflops for single precision.

Thus, by terminating the first kernel and reusing the kernel for all non-empty intervals, we may achieve additional speedup. In addition, the number of CUDA cores was also increased to 2,496 on the Tesla K20c. These improvements lead to significant increase in throughput to a point where the overhead of running to separate kernels is no longer justifiable. Therefore, all implementations in this paper will not group non-empty intervals based on the number of eigenvalues they contain.

Note that the Count(T, λ) function is called at each level of the bisection tree by each child interval. It has been shown in [15] that 90% of the computation time is spent on the Count(T, λ) functions for $n > 100$. Hence, a hybrid approach was proposed where a part of the function calls for the Count(T, λ) function run on CPUs and part of them on GPUs. However, in order to exploit the full capacity of multiple GPUs, we will consider a GPU-only approach.

Algorithm 2 GPU-Based Bisection

```
1:  procedure BISECT(S)
2:      Let S be the collection of the four arrays:
3:          s_left, s_right, s_left_count, s_right_count
4:      tid ← id of the current thread
5:      for each I in S do
6:          Divide I into I_1 and I_2
7:          if I_1 is not empty then
8:              Store I_1 in S[tid]
9:              Store I_2 in S[tid + N]
10:         else
11:             Store I_2 in S[tid]
12:         end if
13:     end for
14:     Perfrom scan-compaction on S
15:     return S
16: end procedure
```

3.3 Multi-GPU Implementation

A simple approach to a multi-GPU implementation is to take advantage of the inherent parallelism in the division stage. In this step, even though two kernels run one after another, they operate on different groups of intervals and, therefore, can be parallelized. Afterwards, we run these two kernels concurrently on two separate GPUs and gain additional performance. We start with the Gerschgorin interval which can be determined on a GPU, similar to the single-GPU implementation.

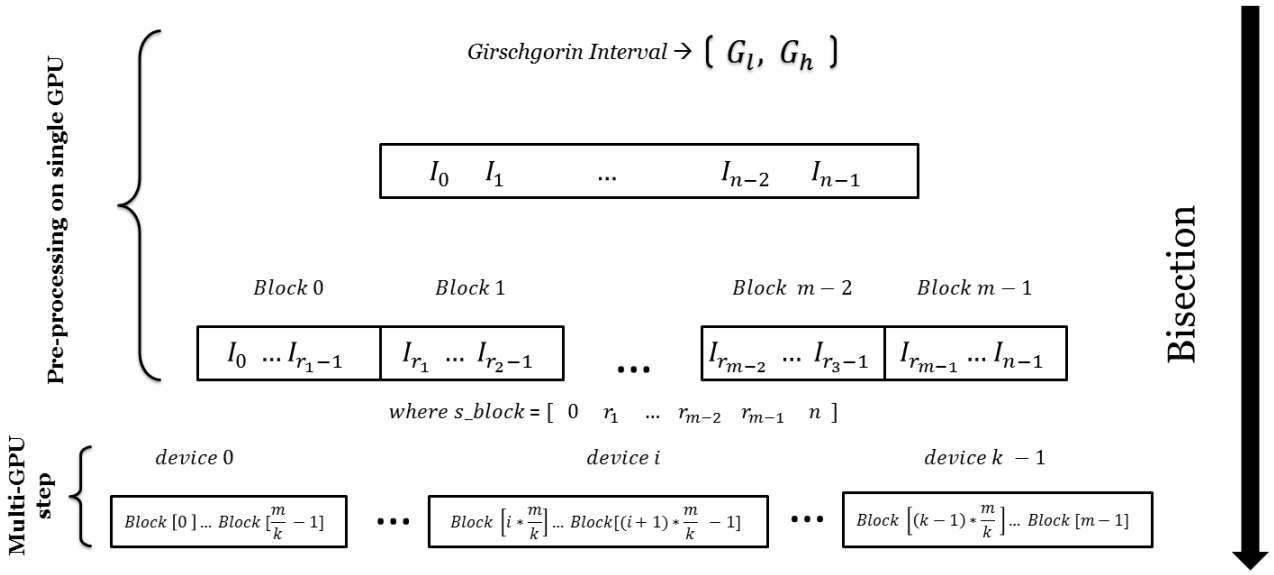


Fig. 2: Generalized Multi-GPU algorithm

The biggest overhead in this approach compared to the single-GPU implementation is data transfer. Unlike single-GPU versions, we must transfer data between the GPUs during the transition at the division stage. To minimize this communication overhead, we can take advantage of the way multi-GPU systems are designed. Data transfer can be done via a peer-to-peer manner, which is much faster than moving the data from device to host and then back to device. Though this approach may achieve a significant speedup, this implementation is limited to only two GPUs. In addition, as mentioned earlier, two kernels launches are unnecessary in most cases.

We now present a scalable approach that will ensure an even workload among multiple GPUs. In the case of bisection, workload has a direct dependency on the number of eigenvalues a device has to compute. Therefore, ensuring a balanced workload means ensuring that each device has to compute about the same number of eigenvalues.

As in the case of single-GPU systems, the first step in the bisection algorithm for multi-GPUs is calculating Gerschgorin interval of a given matrix. An intuitive approach for multi-GPUs is evenly dividing the computed Gerschgorin interval among the available GPUs. But this does not ensure a balanced workload because the spectrum of most matrices does not follow a uniform distribution. Some of those child intervals will likely contain more eigenvalues than the others. Hence, this approach will introduce load imbalance among the GPUs.

So, the approach we take is one that builds upon our algorithm for a single GPU system. Our implementation for a single GPU case begins by launching a bisection kernel to further process the Gerschgorin interval. This pre-processing step takes a constant time which involves bisecting the Gerschgorin interval until the number of child intervals exceeds the maximum number of threads available per block. Our multi-GPU implementation also begins with the same step as described below in more detail.

3.3.1 Pre-processing

In this step, we start with the Gerschgorin interval and keep bisecting it until the number of eigenvalues exceeds the maximum number of threads per block of the CUDA device. This step involves launching a kernel with a grid size of only one block. The results of the bisection are smaller intervals with the known number of eigenvalues. We then distribute these intervals using four arrays that are all stored in the shared memory of the single block that we launch on this kernel.

The left and right bounds of the intervals are, respectively, stored on the `s_left` and `s_right` arrays while the result of the Count(T, λ) function at those bounds are, respectively, stored in the `s_left_count` and `s_right_count` arrays. Note that all four arrays must be sorted. See Algorithms 2 and 3 for details.

3.3.2 Blocking Intervals

The second step which also takes place within the same kernel can be understood as grouping intervals from the pre-processing step into block. This involves grouping intervals so that the number of eigenvalues in each block will not exceed the maximum number of threads available in each block. As observed in [16], the blocking of the intervals is a particular instance of the knapsack problem that is known to be NP-hard.

Algorithm 3 Pre-processing

Let S be the collection of the four arrays:
  s_left, s_right, s_left_count, s_right_count
maxThreads $\leftarrow$ MAX_THREADS_BLOCK
$[G_l, G_h] \leftarrow$ computed Gerschgorin interval
$N \leftarrow$ num of intervals
Store $[G_l, G_h]$ in S
$N = 1$
while $N \leq$ maxThreads **do**
  S = BISECT(S)
  $N =$ number of intevals in S
end while

To this end, we use a greedy algorithm that involves scanning the four arrays. To illustrate this, we assume that there are a maximum of eight threads per block. Let C_λ represent the number of eigenvalues in an interval or a block. As can be seen in Figure 1, our algorithm does not give the optimal solution for interval blocking, but the blocks are still constructed so that the computing power of the GPU is utilized efficiently. The result of the blocking is stored in another shared array called `s_blocks`. For this example, `s_blocks` = [0, 3, 5, 6, 9]. Algorithm 4 gives a detailed description of this step.

Algorithm 4 Blocking intervals

Let S be the collection of the four arrays:
  s_left, s_right, s_left_count, s_right_count
$size \leftarrow$ size of current block
$M \leftarrow$ num of blocks
for each I in S **do**
  if $size + count(I) \leq$ maxThreads **then**
    $size = size + count(I)$
  else
    $M = M + 1$
    $s\_block[M] =$ index of I in S
    $size = count(I)$
  end if
end for

3.3.3 Distribution of Intervals

The results from the previous step are then written to global memory of the GPU and then copied to the host memory. To this end, CPU threads are created for each GPU. Each thread will determine the blocks to be assigned to the GPU based on the number of available GPUs.

It is important to note that the number of blocks assigned to each GPU is roughly the same. If the number of blocked intervals is not evenly divisible by the number of GPUs, the first GPUs will be assigned more blocks. Once the number of block assigned to the GPU is determined, all the results from the interval blocking step are copied to each GPU.

A kernel is then launched on each GPU with a grid size corresponding to the number of blocked intervals assigned to each GPU. For example, if there are only two GPUs, GPU 0 will be assigned block 0 and block 1, and GPU 1 will be assigned block 2 and block 3. Each of these kernels will run the bisection algorithm, described in Section 2, until the desired precision level is reached. See Figure 2 for details.

3.3.4 Collecting Computed Eigenvalues

At this stage, all the converged eigenvalues are copied back to the host CPU. All local lists of the eigenvalue from each GPU are combined to generate a global list containing all the eigenvalues. The whole reduction is done in parallel and takes time that is proportional to the logarithm of the number of blocks per grid.

Finally, the whole procedure on multi-GPU is given in the following algorithm:

Algorithm 5 Bisection on multi-GPUs

Let S be the collection of the four arrays:
  s_left, s_right, s_left_count, s_right_count
Let S_{multi} be each GPU's version of S
$K \leftarrow$ number of GPUs
$M \leftarrow$ number of blocks
for i 1 to $K - 1$ **do**
  start $= s\_blocks[i * \frac{M}{K})]$
  end = $= s\_blocks[(i + 1) * \frac{M}{K} - 1]$
  S_{multi} = S[start:end]
  repeat
    S_{multi} = BISECT(S_{multi})
  until all intervals converge
end for

4. Experimental Results and Discussion

Our CUDA implementation of the bisection algorithm was tested on a machine which comprises a dual Intel Xeon E5-2620 CPUs with 64 GB main memory, and four NVIDIA Tesla K20c GPU, each with 5 GB global memory.

Table 1 shows the actual running time and the speedup of our multi-GPU implementation with random matrices of various size. Clearly, our multi-GPU implementation outperformed the full configuration of the CPU, which comprises 16 cores. When n is large, our implementation runs up to 57x faster than that of CPU with 16 cores. We observe a more dramatic speedup when $n > 16,384$. This is because with large n the global GPU memory becomes saturated with the entries of T and temporary data required by the division steps.

It also shows how much faster the implementation is on multiple GPUs compared to one GPU. Our implementation achieves almost linear speedup with respect to the number of

Table 1: Performance Results on Random Matrices

	Time (sec)				Speedup		
n	16 CPUs	1 GPU	2 GPUs	4 GPUs	16 CPUs / 4 GPUs	1 GPU / 2 GPUs	1 GPU / 4 GPUs
1024	0.1	0.045	0.049	0.052	2.08	0.92	0.86
2048	0.6	0.088	0.094	0.098	5.75	0.93	0.90
4096	2.1	0.181	0.194	0.198	10.83	0.93	0.91
8192	8.7	0.575	0.398	0.416	20.81	1.44	1.38
16384	35.1	1.65	1.193	0.782	44.77	1.38	2.11
32768	133.6	6.23	3.382	2.338	57.16	1.84	2.66

Table 2: Performance Results on Uniform Matrices

	Time (sec)				Speedup		
n	16 CPUs	1 GPU	2 GPUs	4 GPUs	16 CPUs / 4 GPUs	1 GPU / 2 GPUs	1 GPU / 4 GPUs
1024	0.12	0.037	0.037	0.041	2.97	1.00	0.92
2048	0.38	0.074	0.077	0.080	4.73	0.95	0.92
4096	1.33	0.146	0.152	0.156	8.52	0.96	0.94
8192	5.15	0.523	0.302	0.306	16.84	1.73	1.71
16384	20.32	1.936	1.569	0.607	33.48	3.40	3.19
32768	77.54	6.380	2.114	1.143	67.84	3.02	5.58

GPUs, demonstrating the scalability of our implementation on multi-GPU systems.

In addition, when the eigenvalues are uniformly distributed, we achieve a superlinear speedup as shown Table 2. This can be explained by the relationship between matrix size and the number of multiprocessors available on the K20c GPUs. We see that for matrix size up to 4,096, the relationship between matrix size and run time is linear. However, for matrices that are larger than that, the relationship becomes almost quadratic. A possible explanation for this is that when the matrix size is 4,096, the GPU's computing capability is fully exhausted. Hence, for matrix sizes larger than this threshold, not all eigenvalues can be computed in parallel. A similar phenomenon exists in the case of two GPUs. For the matrices smaller than 16,384, the relationship is linear, but for larger matrices, the speedup becomes almost quadratic.

Our implementation of the bisection algorithm in this paper, however, has not taken full advantage of hardware and software features that GPUs have to offer. We are considering a hybrid approach that uses both CPU and GPU to speed up the bisection algorithm even further. For example, the Gerschgorin interval can be preprocessed on the CPU. Furthermore, CPU can employ multi-section algorithm instead of bisection, which means dividing a big interval into multiple smaller intervals instead of only two ones. The optimal number of subintervals in a multi-section algorithm remains to be found.

Future work should also be able to accommodate out-of-core cases. These situations may occur with very large matrices because symmetric tridiagonal matrices are represented by only two vectors for diagonal and off-diagonal elements. Parallel out-of-core version of the bisection method should be an interesting set of problems for creative solutions.

5. Acknowledgements

The authors would like to thank CUDA Teaching Center Program, Nvidia Research and Student Research Program, Trinity College for supporting this research. The authors would also like to thank Nam Thai of Trinity College for his contribution to a preliminary version of our implementation.

References

[1] G.H. Golub and C.F. Van Loan. *Matrix Computations, 4th ed.* Johns Hopkins University Press, 2012.

[2] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach* (Burlington, MA : Morgan Kaufmann Publishers, 2010).

[3] NVIDIA, cuBLAS Library User Guide. http://docs.nvidia.com/cuda/pdf/ CUBLAS_Library.pdf.

[4] NVIDIA, Profiler User's Guide. http://docs.nvidia.com/cuda/pdf/ CUDA_Profiler_Users_Guide.pdf

[5] J.R. Bunch, C.P. Nielsen, and D.C. Sorensen, "Rank-one modification of the symmetric eigenproblem," *Numer. Math*, vol. 31, pp. 31–48, 1978.

[6] J.J.M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem," *SIAM J. Sci. Stat. Comp*, vol. 2, pp. 139–154, 1981.

[7] M. Gu, S.C. Eisenstat, A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem, *SIAM Journal on Matrix Analysis and Applications*, 15(4), 1994, 1266-1276

[8] G. Baker, "Accelerated bisection techniques for tri and quintadiagonal matrices," *Int. J. for Num. Meth in Eng.,* pp. 203–218, 1992.

[9] R.H. Barlow nd D.J. Evans, "A parallel organization of the bisection algorithm," *Comp. J.* , vol. 22, pp. 267–269, 1977.
[10] C. Vomel, S. Tomov and J. Dongarra, Divide and conquer on hybrid GPU-accelerated multicore systems, *SIAM Journal on Scientific Computing*, 34(2), 2012, C70-C82
[11] W. Barth, R.S. Martin and J.H. Wilkinson, "Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method," *Numer. Math.*, vol. 9, pp. 386–393, 1967.
[12] E. Anderson, et al. *LAPACK User's Guide, 3rd ed.* SIAM, 1999.
[13] D. S. Watkins, *Fundamentals of Matrix Computations* (New York: John Wiley and Sons Incs., 1991).
[14] H.J. Bernstein and M. Goldstein, "Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal matrix," *Computing*, vol. 37, pp. 85–91, 1986.
[15] V. Volkov and J. Demmel, "Using GPUs to accelerate the bisection algorithm for finding eigenvalues of symmetric tridiagonal matrices," *Technical Report*, No. UCB/EECS-2007-179, 2007.
[16] C. Lessig, "Eigenvalue computation with CUDA," The Nvidia website. [Online]. Available: http://developer.nvidia.com/cuda-zone
[17] J. Demmel. *Applied Numerical Algebra* SIAM, 1997.
[18] NVIDIA, "CUDA Toolkit v6.0" The Nvidia website. [Online]. Available: http://docs.nvidia.com/cuda
[19] NVIDIA, CUDA C Programming Guide.

Use of the SCIDDICA-SS3 model for predictive mapping of debris flow hazard: an example of application in the Peloritani Mountains area

V. Lupiano[1], D.J. Peres[2], M.V. Avolio[3], A. Cancelliere[2], E. Foti[2], W. Spataro[3], L.M. Stancanelli[2], S. Di Gregorio[3]

[1]Dept. of Biology, Ecology and Earth Sciences, University of Calabria, 87036 Rende, Italy
[2]Dept. of Civil Engineering and Architecture, University of Catania, viale Doria 6, 95125 Catania, Italy
[3]Dept. of Mathematics and Computer Science, University of Calabria, 87036 Rende, Italy

***Abstract**: Extreme rainfall events and subsequent triggering of shallow landslides hit many regions worldwide, and climatic change increases the frequency of these phenomena, so that there is an increased need of appropriate risk mitigation tools. SCIDDICA-SS3 is a semi-empirical Cellular Automata (CA) model for simulating the run-out of debris/mud flows. In this study, the model was initially calibrated by comparison with real events occurred in the Giampilieri area, of the Peloritani Mountains (Sicily) which was devastated on October 1st, 2009, by more than 500 landslides, mostly debris flows, caused by extreme rainfall, causing more than 30 causalities and severe damage to roads, railway and buildings. Satisfying simulations of the six main debris flows in the Giampilieri area with appropriate values of SCIDDICA-SS3 parameters, enables to transfer results in order to assess hazard scenarios in areas with similar geo-physical features. In particular, the Messina Sud tollgate of A18 highway (Sicily, Italy) is an example case of CA simulation for hazard analysis, with detection of subareas of highest risk and their possible protection works.*

Keywords: Debris flows, Cellular Automata, Modelling and Simulation, Natural Hazard.

1 Introduction

The frequency of meteorological extreme catastrophic events has increased due to climatic change. Modelling complex dangerous phenomena by computer simulation can provide new tools, which may be an aid for natural hazard risk mitigation. On October 1st, 2009, the Peloritani Mountains (NE Sicily) was hit by an extreme meteorological event: 7 hours of high intensity rainfalls have triggered more than 500 landslides, mostly debris flows; catastrophic proportions were reached with 37 deaths and thousands of evacuated people in Messina area.

Debris flows are very complex systems, which involve many interacting processes; in summary, they start from soil detachments, that originate surface gravitational flows with mutable rheological properties, furthermore their passing causes soil erosion with inclusion of various matter and induction of secondary detachments. The complex dynamics of such phenomenon may be described in terms of local interactions [1], [2].

Data about evolution of events of debris flows are usually partial and their reproduction in laboratory by mock-up models cannot satisfy completely demands about variation of physical quantities. This limit of knowledge is important for conceiving predictive tools. Models based on PDE (Partial Differential Equations), that are related to fluid-dynamics equations and that approximate opportunely the various typologies of debris flow, were developed together with codes for computer simulation, e.g., TITAN2D [3], DAN3D [4], etc.

An alternative method to PDE methodologies is represented by Cellular Automata (CA), a computational paradigm for modelling complex dynamical systems, evolving mainly on the base of local interactions [5]. CA approach is based on the definition of "simple", but fundamental local rules, that have to observe conservation physical laws in the context of space and time discretization. Such rules have to capture the significant interacting processes. The phenomenon evolution "emerges" by local interactions: simple rules can generate very complex realistic behaviors. Macroscopic Cellular Automata (MCA), are an extension of classical CA, were developed in order to model many natural events macroscopic that seem difficult to be modelled in other CA frames, e.g. the lattice Boltzmann method, [6], [7], just because they take place on a large space scale.

SCIDDICA-SS3 [8], one of the last models of the SCIDDICA family, was applied for simulating 2009 debris flows in Giampilieri after an accurate examination of geological data of the zone, studies about pre- and post-event situations, data and features of landslides. Good simulations of past events permit reliable applications of the model, in areas with similar geomorphological and soil conditions, for hypothetical landslide detachments, according to accurate theoretical and field studies. This was performed for the small sensible area around the Messina tollgate of A18 highway in Sicily. An outline of computational approach together with basic characteristics of the model SCIDDICA-SS3 is introduced in the next section; the third section reports essential information about the model validation for zones with same geophysical features of Giampilieri Superiore area: preliminarily the geological setting, then the crucial simulation of debris flow, that flooded the village and permitted to tune parameters for characteristics of this area. The successive section examines the geophysical characteristics of area around the Messina

Sud tollgate of A18 and with hypothesized detachment points, from which landslides could be generated, and the results of simulations of possible debris flows reported. Eventually, a comparison with FLO-2D [9] simulations is performed, while at the end some comments and conclusions are considered.

2. SCIDDICA-SS3 Approach

A CA can intuitively be seen as a regular tessellation, i.e. a space, partitioned in equal cells, each one embedding an identical input/output computing unit. Each cell is characterized by its state. Input for each cell is local and is given by the states of the neighboring cells, where the neighborhood conditions are given by a pattern invariant in time and space. At time 0, cells are in arbitrary states (initial conditions) and the CA evolves changing simultaneously the state at discrete times (steps), according to local rules (transition function), that are invariant in time and space.

CA modelling and simulating real complex systems implies that a time-space correspondence must be explicitly established between the model and the real world in order to compare phenomenon development with simulation progress: the cell corresponds to a precise space portion, the transition step corresponds to a time interval. Moreover, the cell state accounts for components, the "sub-states", that are related to various characteristics of the space portion and the transition function, that accounts for a complexity of interrelating processes of similar or different nature is composed by "elementary processes".

SCIDDICA-SS3 was developed by adopting the methodology of Multicomponent (or Macroscopic) Cellular Automata (MCA), that includes the previous instances and considerations. MCA approach addresses complex systems, evolving mainly on the base of local interactions, especially macroscopic phenomena that need many components both for the states and the transition function in order to describe local processes constituting the overall phenomenon.

A MCA step consists in the application of an ordered sequence of the elementary processes, every elementary process implies the MCA state updating [10], [11]. Besides, some parameters (e.g. cell dimension, temporal correspondence of a MCA step, etc.) are global to all the cellular space. Some model parameters, by taking into consideration the physical/empirical features of the complex system, have to be opportunely "tuned" for reproducing correct dynamical behaviors of the phenomenon. Sub-states permit to operate in three effective dimensions by two-dimensions MCA, if all the quantities concerning the third dimension may be expressed as sub-states.

At the beginning of the simulation, cell states are initialized. By simultaneously applying the transition function, τ, to all cells and at discrete steps, states are changed and the evolution of the phenomenon can be simulated.

2.1 SCIDDICA-SS3 Formal Outline

SCIDDICA-SS3 is a hexagonal MCA model, where the third dimension is included into the set of states, formally defined as:

$$\text{SCIDDICA-SS3} = \langle R, X, S, P, \tau \rangle \quad (1)$$

- $\boldsymbol{R}$ is the set of regular hexagons covering the region, where the phenomenon evolves.
- $\boldsymbol{X}$ identifies the geometrical pattern of cells, which influence any state change of the central cell: the central cell (index 0) itself and the six adjacent cells (indexes 1,..,6).
- $\boldsymbol{S}$ is the finite set of states of the finite automaton, embedded in the cell; it is equal to the Cartesian product of the sets of the considered sub-states:

$$S = S_A \times S_D \times S_{TH} \times S_X \times S_Y \times S_{KH} \times S_{Mx} \times S_{My} \times S_E^6 \times S_{XE}^6 \times S_{YE}^6 \times S_{KHE}^6 \times S_I^6 \times S_{XI}^6 \times S_{YI}^6 \times S_{KHI}^6 \times S_D \quad (2)$$

where:
  - S_A is the cell altitude, S_D is the maximum depth of detrital cover, that could be transformed by erosion in landslide debris;
  - S_{TH} is the average thickness of landslide debris inside the cell, S_X and S_Y are the co-ordinates of the debris barycentre with reference to the cell centre, S_{KH} is the debris kinetic head, S_{Mx} and S_{My}; are the two components of the debris momentum.
  - S_E is the part of debris flow, the so called "external flow", (normalised to a thickness) that penetrates the adjacent cell from central cell, S_{XE} and S_{YE} are the co-ordinates of the external flow barycentre with reference to the adjacent cell centre, S_{KHE} is the debris kinetic head, (six components for all the sub-states);
  - S_I is the part of debris flow toward the adjacent cell, the so called "internal flow", (normalised to a thickness) that remains inside the central cell, S_{XI} and S_{YI} are the co-ordinates of the internal flow barycentre with reference to the central cell centre, S_{KHI} is the debris kinetic head, (six components for all the sub-states);
  - S_S accounts for some soil conditions as the initial detachment area or transepts of detachment propagation.
- $\boldsymbol{P}$ is the set of the global physical and empirical parameters, which account for the general frame of the model and the physical characteristics of the phenomenon:

$$P=\{p_a, p_t, p_{adh}, p_{fc}, p_{td}, p_{ed}, p_{mt}, p_{pe}\} \quad (3)$$

where:
  - p_a is the cell apothem;
  - p_t is the temporal correspondence of a CA step;
  - p_{fc} is the friction coefficient for debris outflows;
  - p_{td}, p_{ed} are the parameters for energy dissipation by turbulence and by erosion;
  - p_{pe} is the progressive erosion parameter;

- p_{adh} is the adhesion values, i.e. the debris thickness, that may not be removed;
- p_{tmt}, is the activation threshold of the mobilisation for the transepts.

- τ: $S^7 \rightarrow S$ is the cell deterministic state transition. Four elementary processes are considered:
  - altitude, kinetic head, momentum and debris thickness variation by detrital cover mobilization;
  - kinetic head and momentum variation by turbulence dissipation;
  - debris outflows (thickness, barycentre co-ordinates, kinetic head) determination and their shift deduced by the motion equations;
  - composition of debris inside the cell (remaining debris more inflows) and determination of new thickness, barycentre co-ordinates, kinetic head, momentum.

3. Giampilieri Superiore debris flows simulations

Giampilieri Superiore village is located on the eastern slopes of the Peloritani Mountains on left side of Giampilieri River (Fig. 1). In this area, we find in outcropping the Kabilo-Calabride Units, formed by continental crust fragments derived from the European margin, and in particular by crystalline-schistose rocks with different metamorphic degree. A colluvium with thickness varying from some tens of centimeters to a few meters is detected on this altered substrate. The slopes have high gradients (30-60 degree) and are crossed by numerous, generally short path, streams with torrential regime.

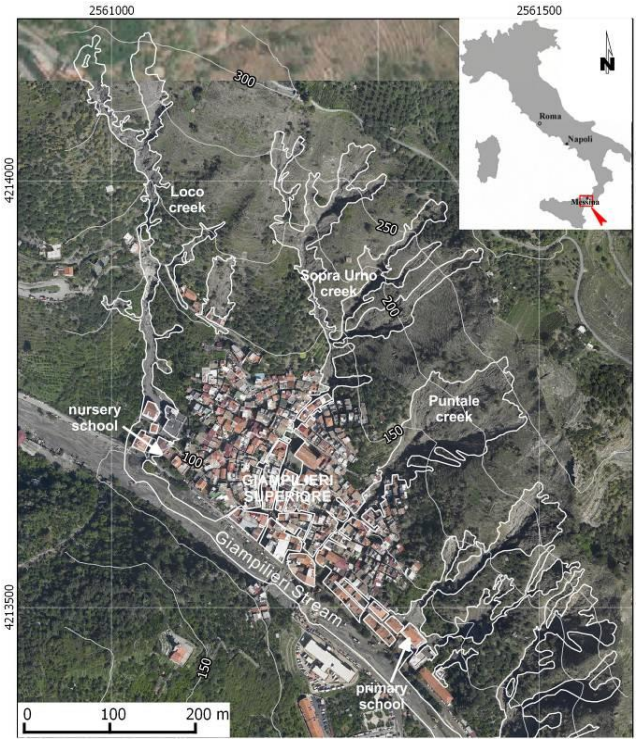


Fig. 1: October 2009 debris flows occurred in Giampilieri Superiore, obtained by interpretation of aerial photo.

The inhabited is placed on an alluvial fan at the base of Southern-East slope of Puntale S. Anna, and is crossed by various creeks, tributaries of the Giampilieri stream (Fig. 1). All of them are characterized by small catchments with extension ranging from 0.03km^2 to 0.1km^2 [12]. The riverbeds of these creeks, in the urban core, have been turned into streets with few or completely absent hydraulic works for regimentation and water disposal.

On the 1st October 2009, in the afternoon, fast moving debris flow phenomena which have produced most of damage on Giampilieri Superiore village (Fig. 1). Mud and debris, channeled in the streets (once river beds), inundated the urbanized area. The severity of the rainfall event was not the only cause of the disaster. A simple analysis of the phenomenon has to take into account also for a general neglected care of the lands beside the urbanized area once very well planted and managed from a hydraulic viewpoint and a diffused inadequacy of the urban drainage network [13]. In fact, for the alluvial case of Giampilieri, as also stressed by Ardizzone et al. [14], abandoned terraced slopes lacking proper drainage, and unmaintained dry walls were also related to slope failures.

Sopra Urno debris flow caused the largest number of casualties and damages, because the flows crossed the village (Sopra Urno creek became Chiesa Street in the urban section).

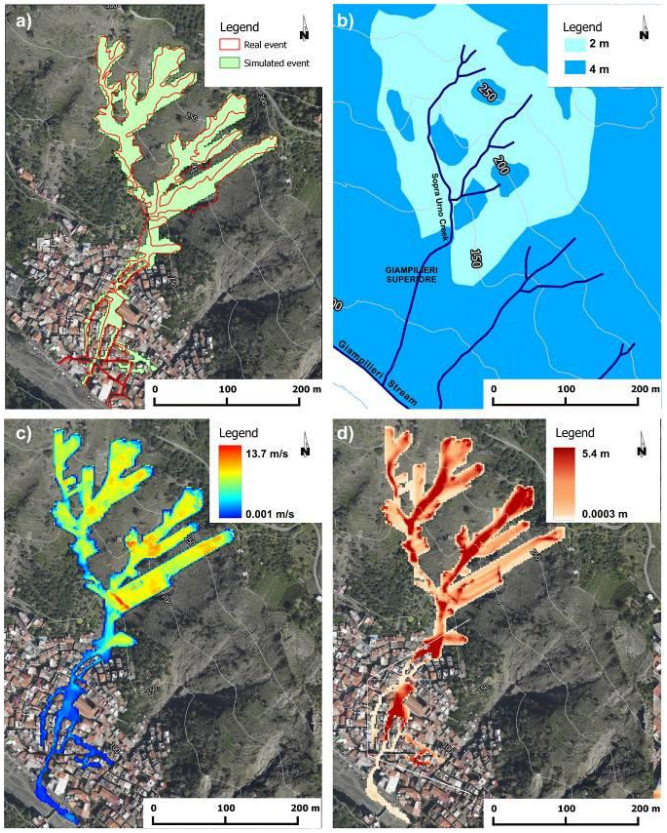


Fig. 2: a) comparison between Sopra Urno creek debris flows and simulated event; b) Regolith thickness used for simulation; c) maximum velocities reached by flows; d) maximum detrital thickness.

In such case, the mutual interaction between different, nearly simultaneous, debris flows produced dramatic effects in terms of loss of human lives and damages of buildings close to the hill (Fig. 1) and along the principal streams that cross the town.

This debris flow was simulated by SCIDDICA-SS3 model, outlined in the previous sections. Hexagonal cells, with apothem equal to 1m, were adopted. We used for simulation a pre-event DEM (Digital Elevation Model) with 2m cell size and the debris cover thickness map available for the area (Fig. 2b). The simulation describes (Fig. 2a) quite well the debris run-out, in particular, in high zone of slope. In the urbanized area, differences are noted with path of the real event, especially in lateral streets, but the result can be considered acceptable. The likelihood between the area involved by the real landslide and the area involved in the simulation can be measured by the fitness function:

$$f = \sqrt{((R \cap S)/(R \cup S))} \quad (4)$$

where R is the set of cells interested by the real landslide and S is the set of cells interested by the simulated landslide. This function returns values from **0** (completely wrong simulation) to **1** (perfect match); values greater than **0.7** are considered good results. Comparison of simulated and real-event (Fig. 2a) return a value of fitness $f = 0.74$.

The maximum velocities reached by simulated flows (Fig. 2c) are high, as expected, in the steeper areas, and decrease gradually at the outlet in downstream. Figure 2d shows the maximum detrital thicknesses during the simulation.

In order to validate the model, the same set of parameters was used to simulate other five debris flow run-out in the nearby catchments [15]. In all considered cases, the application of equation (4) return values between 0.70 and 0.78, and the path of the flows is adequately reproduced.

Scenarios of A18 tollgate

The SCIDDICA-SS3 model, calibrated and validated on Giampilieri debris flows, was applied in order to produce susceptibility scenarios in the areas around A18 highway, Messina Sud tollgate, which presented similar soil characteristics and morphological conditions of slopes. This is a necessary condition in order to apply in other areas the SCIDDICA-SS3 validated parameters.

The carriageway is protected by a wall high about 1.50 m, in correspondence of the tollgate is located a drainage channel that interrupts the continuity of the wall (Fig. 3). This introduces a source of risk for the highway. In fact, the tollgate area has been interested several times by debris inundation, due also to lack of appropriate maintaince of the channel. In fact, boulders and/or shrubs ripped along the path, from the flows, could obstruct the channel favoring the overflow of debris.

A pre-event DEM with 2m cell size and for debris cover a uniform thickness of 0.5 m was used for simulation. This value seems to produce results that are consistent with the frequency of observed events.

Fig. 3: Two views of Messina Sud tollgate area of A/18 highway.

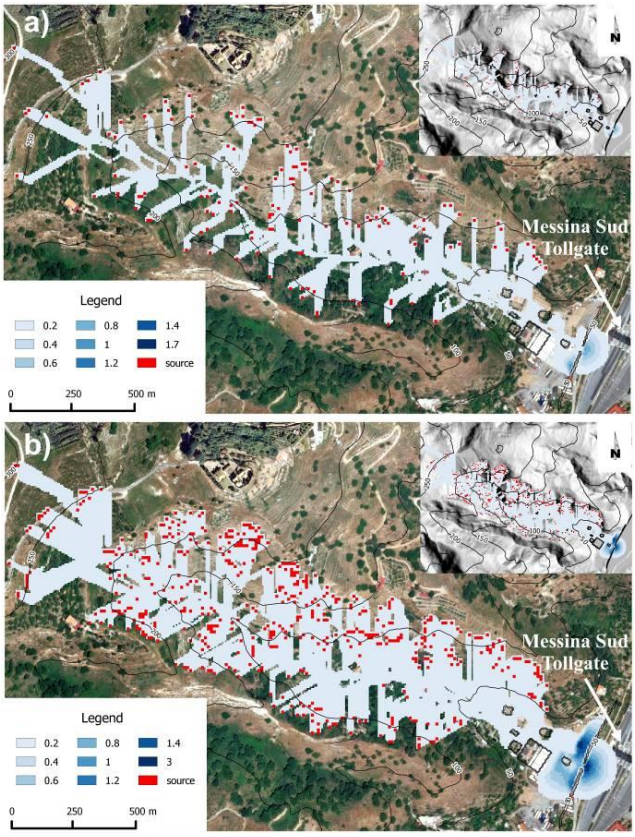


Fig. 4: Susceptibility scenarios for 4 (a) and 5 (b) rain return period.

Sources were identified for debris flow triggering return period of 4 and 5 years. These are obtained through the areas of probable detachments, i.e., analysis of precipitation for the determination of the rainfall Intensity-Duration-Frequency curves (IDF) [12], [16], [17]. In particular, the probabilistic rainfall analysis provides the rainfall input scenarios of different return periods, which are then used as input [16] to a model based of the USGS TRIGRS model to determine the corresponding trigged areas. TRIGRS [18], [19] developed for analyzing shallow landslide triggering is based on an analytical solution of linearized forms of the Richards' infiltration equation and an infinite-slope stability calculation to estimate the timing and locations of slope failures.

In Fig. 4a and 4b the results of simulations are shown, respectively for return period of 4 years and 5 years. Simulations (Fig. 4a and Fig. 4b) show that the flow is channeled and the debris invades the highway carriageway at the lowest point of the morphology, i.e., the drainage channel.

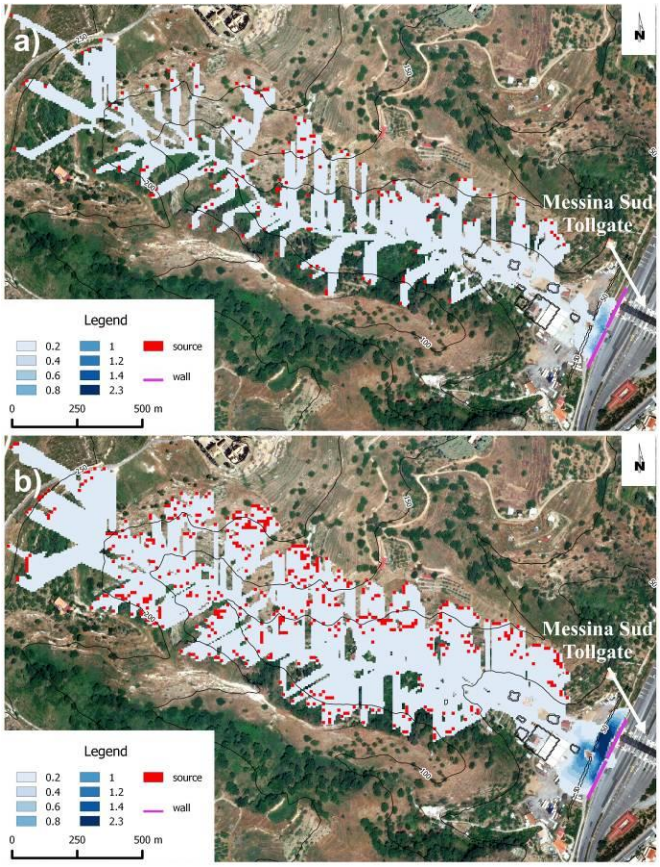


Fig. 5: susceptibility scenarios for 4 (a) and 5 (b) rain return period with insertion of a wall.

In order to verify whether the presence of a higher wall could prevent the invasion of the highway it was included in DEM a topographical alteration along the previous wall, that is raised by 2 meters. The added wall is imposed as "indestructible" in our simulations. Fig. 5a and 5b show the results of the simulations for the same return periods; such scenarios account for topographical alteration. The simulations show that such a wall is able to stem the flow completely for 4 years return period avoiding highway invasions and/or damage, while only small quantities of debris overcome the wall at the lowest point in the case of 5 years return period.

Preliminary simulations were performed by Flo-2D [16], [17], using the same detachment points and without topographical alteration as in some SCIDDICA-SS3 simulations. Fig. 6 synthetizes the results with scenarios respectively of return periods of 4, 5, 10, 25 and 50 years. Comparison between simulations of SCIDDICA-SS3 and FLO-2D evidence a weak point, where debris flows could invade the highway. More severe scenarios were derived by FLO-2D.

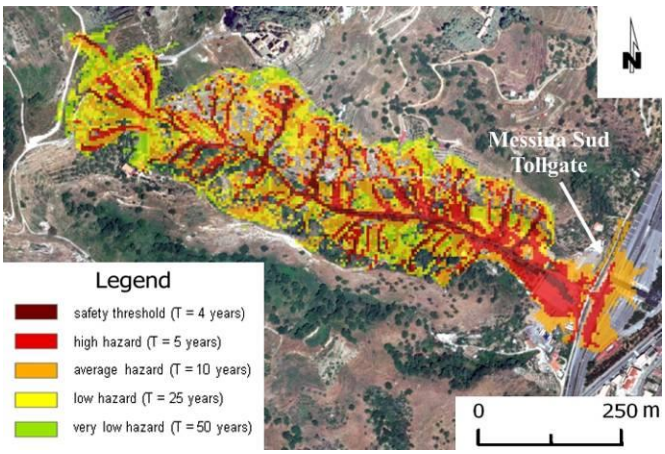


Fig. 6: FLO-2D scenarios.

Conclusions

An example of application of the SCIDDICA-SS3 model for predictive hazard mapping has been presented. The method we proposed starts with the calibration of the model, based on observed events. In our case, this has been carried out by simulating at best the debris flows occurred on October 1st, 2009 at Giampilieri Superiore.

An accurate study was performed in order to compare data of different sources and to obtain the most accurate reproduction of the observed event. The model behaviour was satisfactory in terms of reproducing global dynamic of the events, such as velocity, debris flow depth, thickness of deposit, and, in particular, the path of debris flows, that shown a good correspondence with the real events. Once the model has been calibrated, it has then been applied for predictive mapping in areas that may be considered similar from a geo-physical standpoint. In particular, an application was carried out to a creek which created risk to the Messina Sud tollgate of highway A/18 with very similar geological characteristics. On the basis of probabilistic rainfall analyses about hypothetical and probable detachment areas, source points were assessed for rain return period of 4 and 5 years.

Results of this complex study about the dangerous area of A18 Messina Sud tollgate has proved that the model is an aid for detection of potential debris-flow risk scenarios. The top of the wall, that is overlooking the drainage channel, is discontinuous in height with respect to the floor of the highway. The lowest point, where debris flows invade the highway in the simulation scenarios, corresponds to drainage channel. This is the weak point because a largest flow could be not drained and could pour into the tollgate zone. The wall would be opportunely extended and raised (or a new wall on the edge of highway) in order to protect the highway, without undermining the efficiency of draining channel. These are our concerns that are based on the interpretation of the scenarios developed by SCIDDICA-SS3, together with more general considerations that derive from site surveys.

Acknowledgments - This research was funded by PON Project No. 01_01503 "Integrated Systems for Hydrogeological Risk Monitoring, Early Warning and Mitigation Along the Main Lifelines", CUP B31H11000370005, in the framework of the National Operational Programme for "Research and Competitiveness" 2007-2013.

References

[1] D. D'Ambrosio, S. Di Gregorio, G. Iovine, V. Lupiano, L. Merenda, R. Rongo, W.Spataro. Simulating the Curti-"Sarno debris flow through cellular automata: the model SCIDDICA (release S2)", Physics And Chemistry Of The Earth, 2002, vol. 27 (36); p. 1577-1585, ISSN: 1474-7065.

[2] G. Iovine, S., Di Gregorio, V. Lupiano. "Debris Flows susceptibility assessment through Cellular Automata modeling: an example from the 15-16 December 1999 disaster at Cervinara and San Martino Valle Caudina (Campania, southern Italy)", Natural Hazards And Earth System Sciences, 2003, vol. 3; p. 457-468, ISSN: 1561-8633.

[3] S. McDougall, O. Hungr. "A model for the analysis of rapid landslide motion across three-dimensional terrain", Can. Geotech. J., Vol.41, 2004, pp. 1084–1097.

[4] E.B. Pitman, C.C. Nichita, A.K. Patra, A.C. Bauer, M.F. Sheridan, M. Bursik. "Computing Granular Avalanches and Landslides ", Physics of Fluids, Vol.15, No.12, 2003, pp. 3646-3638.

[5] B. Chopard, M. Droz. «Cellular Automata Modeling of Physical Systems», Cambridge University Press, 2005.

[6] G.R. McNamara, G. Zanetti. "Use of the Boltzmann equation to simulate lattice-gas automata"; Physical Review Letters 61, 2332e2335, 1988.

[7] S. Succi, R. Benzi, F. Higuera. "The lattice Boltzmann equation: a new tool for computational fluid dynamics"; Physica 47 (D), 219-230, 1991.

[8] M.V. Avolio, V. Lupiano, P. Mazzanti, S. Di Gregorio. "SCIDDICA-SS3: A New Version of Cellular Automata Model for Simulating Fast Moving Landslides", J. Supercomput., Vol. 65, 2013, pp. 682-696.

[9] J. S. O'Brien, P. Y. Julien, and W. T. Fullerton. "Two-dimensional water flood and mudflow simulation", J. Hydraul. Eng., 119, 244-261, 1993.

[10] S. Di Gregorio, R. Serra. "An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata"; Future Generation Computer Systems, Vol. 16, 259–271, 1999.

[11] D. D'Ambrosio, S. Di Gregorio, S. Gabriele, and R. Gaudio, 2001. "A Cellular Automata Model for Soil Erosion by Water". Physics and Chemistry of the Earth, 2001, Part B, 26(1), 33-40.

[12] L.M. Stancanelli, V. Bovolin, E. Foti. "Application of a dilatant - viscous plastic debris flow model in a real complex situation, River, Coastal and Estuarine Morphodynamics", RCEM2011, Tsinghua University Press, Beijing, 2001, pp 45-64.

[13] B. Manfrè, C. La Rocca, V. Nicolosi, E. Foti, L. M. Stancanelli. "Socially participated decision making process for hydrogeological risk mitigation: Giampilieri, 1st October 2009", Comprehensive Flood Risk Management Research for Policy and Practice, Edited by Timo Schweckendiek CRC Press 2012 Print ISBN: 978-0-415-62144-1eBook ISBN: 978-0-203-37451-1 DOI: 10.1201/b13715-173.

[14] F. Ardizzone, G. Basile, M. Cardinali, N. Casagli, S. Del Conte, C. Del Ventisette, F. Fiorucci, F. Garfagnoli, G. Gigli, F. Guzzetti, G. Iovine, A.C. Mondini, S. Moretti, M. Panebianco, F. Raspini, P. Reichenbach, M. Rossi, L. Tanteri, O. Terranova. "Landslide inventory map for the Briga and the Giampilieri catchments, NE Sicily, Italy", Journal of Maps, 8 (2), 2012. pp. 176-180.

[15] V. Lupiano, M.V. Avolio, S. Di Gregorio, D.J. Peres, L.M. Stancanelli. "Simulation of 2009 debris flows in the Peloritani Mountains area by SCIDDICA-SS3", Proceeding of 7th WSEAS International Conference on Engineering Mechanics, Structures, Engineering Geology, Salerno (Italy), 2014pp. 53-61, ISBN: 978-960-474-376-6.

[16] L. M. Stancanelli, D. J. Peres, L. Cavallaro, A. Cancelliere, E. Foti. "Debris flow hazard assessment by integrated modeling of landslide triggering and propagation: application to the Messina Province, Italy", AGU Fall meeting Abstracts, 15-18 December 2014, San Francisco, 2014.

[17] L.M. Stancanelli, G. Rosatti, L. Begnudelli, A. Armanini, E. Foti. "Single or Two-Phase Modelling of Debris-Flow? A Systematic Comparison of the Two Approaches Applied to a Real Debris Flow in Giampilieri Village (Italy)", Landslide Science and Practice, Vol.3, pp. 277-283, Springer-Verlag Berlin, 2013.

[18] R. L. Baum, W. Z. Savage, J. W and Godt. "TRIGRS – A FORTRAN program for transient rainfall infiltration and grid-based regional slope stability analysis, version 2.0", US Geological Survey Open-File Report 2008-1159, 75 pp. 2008.

[19] R. L. Baum, J.W. Godt, and W. Z. Savage. "Estimating the timing and location of shallow rainfall-induced landslides using a model for transient, unsaturated infiltration", Journal of Geophysical Research, Earth Surface. 2010, v. 115, F03013, doi:10.1029/2009JF00132.

SESSION

LATE BREAKING PAPERS: PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS

Chair(s)

TBA

VoxSurf: a Voxelized macromolecular Surface calculation program

Sebastian Daberdaku, Carlo Ferrari
Department of Information Engineering, University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy

Abstract— *We introduce VoxSurf, a Voxelized macromolecular Surface calculation program, which can produce discrete representations of molecules at very high resolutions. By employing compact data-structures and implementing a spatial slicing protocol, the proposed tool can calculate the three main molecular surfaces at high resolutions even when the available memory is limited.*
The generation of the Solvent Excluded surface is achieved by adopting an approximate Euclidean Distance Transform algorithm based on a data-structure called Hierarchical Queue. We show that the distance map values need to be calculated only for a small subset of the overall voxels representing the macromolecule by exploiting the geometrical relationship between the Solvent Excluded and the Solvent Accessible surfaces.
A parallel implementation of the slicing procedure is also proposed and discussed.

Keywords: macromolecular surface, protein surface, high-resolution surface, voxelization, Euclidean Distance Transform

1. Introduction

Different representations of the molecular surface can capture diverse aspects of the three-dimensional geometry of proteins and macromolecules in general. The currently most used methodologies are: the van der Waals surface (vdW) [1], the Solvent-Accessible surface (SAS) [2] and the Solvent-Excluded surface (SES) or Connolly surface [3]. Since surface complementarity drives protein interactions, accurate determination of protein surfaces is essential for understanding their biological roles in physiological processes. As a consequence, protein surface calculations from given three-dimensional structures have been used extensively in modern molecular biology studies, and different methods to compute the three macromolecular surfaces have been proposed [3–9].

Among the explicit representations, the voxelized ones (also known as dot-surface or grid-based representations) are the most simple, and yet widely appreciated for their accuracy and applicability in various contexts. The Katchalski-Katzir algorithm [10] is one of first employments of this kind of surface representations in the modeling of the protein docking process. Voxelized protein surfaces are currently being employed in descriptor-based protein docking and protein shape comparison. Kihara et al. propose protein docking, shape comparison and interface identification methods based on 3D Zernike descriptors (3DZD) [11–18], which are calculated over circular surface patches of voxelized macromolecular surfaces. In [19], dot-surfaces are used in the development of an invariant descriptor for the characterization of protein surfaces, suitable for various analysis tasks, such as protein functional classification or search and retrieval of protein surfaces over a large database. Invariant surface fingerprints have been introduced in [20] in order to compare local protein surface similarities rapidly and efficiently. The creations of these fingerprints employs a dot-surface representation of the molecular surface. The voxelized representation of a molecular surface can describe the molecule's flexibility [15, 21] and physicochemical property values, such as electrostatic potentials or hydrophobicity [22]. Grid representations of protein surfaces have also been used in cavity detection, binding-pockets identification and evaluation techniques [23–26].

All the techniques and algorithms employing the voxelized representation of the molecular surface calculate the latter on their own, usually from parametric surface representations or from other explicit representations such as triangle mesh surfaces. The triangulated protein surface is placed on a 3D grid, and the voxels (grid points) intersected by the mesh faces are marked as occupied (typically with 1, 0 otherwise). To the extent of our knowledge, there are no tools available which can produce a voxelized molecular surface representation starting from the data contained in the molecule's Protein Data Bank (PDB) [27] file. Thus, the idea of developing a specific tool for the calculation of voxelized macromolecular surfaces at arbitrary resolutions, starting from the macromolecular structure data derived from X-ray diffraction and NMR studies (PDB data).

In this paper we describe VoxSurf, a Voxelized macromolecular Surface calculation program, which can produce discrete representations of molecules at very high resolutions starting from the three-dimensional structural information given by Protein Data Bank entries. By employing compact data-structures for the 3D grid representation, and implementing a spatial slicing strategy, this tool can calculate the three main molecular surfaces at very high resolutions with very little memory usage. The tool is available at: http://www.dei.unipd.it/∼daberdak/VoxSurf.

2. Methods

The first step of the proposed methodology consists in reading the three-dimensional representation of a macromolecule from its corresponding Protein Data Bank entry. The atomic coordinates of each atom composing the

macromolecule are extracted and stored in an apposite data-structure. The algorithm calculates the axis-aligned bounding-box enclosing the whole molecule by determining the minimal and maximal coordinates of each of the atoms in the molecule.

Given a desired grid resolution parameter, the dimensions of the voxel grid which will contain the molecule, are calculated. All atomic coordinates previously imported are translated, scaled and quantized to the new coordinate system defined by the voxel grid: each atom center is mapped in its corresponding voxel in the voxel grid.

By implementing a space-filling algorithm, all voxels surrounding a given atom center, are marked as occupied by that atom if their distance from its center is less or equal to the corresponding atomic radius. Based on the desired surface (SAS, vdW or SES), different atomic radii values must be used. After all the atoms composing the macromolecule have been mapped into the grid, we obtain a voxelized representation of what is known as CPK model [28] (also known as calotte model or space-filling model).

To obtain the van der Waals or the Solvent Accessible surfaces, we simply extract the surface voxels from the voxelized representation of the CPK volumetric model of the macromolecule. The Solvent Excluded surface is trickier to calculate because it includes the re-entrant surface portions. We have implemented three different algorithms to calculate such surface. The first and most simple one is based on the rolling-sphere representing the solvent molecule, which "probes" the molecular surface. The other two implementations are based on the Euclidean Distance Transform (EDT) algorithm for surface smoothing. They differ in the data-structures employed, i.e. one uses speed-optimized data-structures, instead of the memory-optimized structures implemented in the other.

The resulting voxelized surface is exported in an output file. We have chosen the Point Cloud Data file format (*.pcd) [29] of the Point Cloud Library (PCL) [30], because of its simplicity, compactness and compatibility with different scientific visualization programs.

2.1 The voxel grid

In this work we employ cubic voxels only, and the voxel grid is a regular axis-aligned cubic grid in the three-dimensional space. The resolution of the grid is specified in terms of a floating point parameter p_{res}, so that the number of voxels per cubic Ångström is given by p_{res}^3.

We have defined and implemented a compact representation for the voxel grid, by tightly packing multiple Boolean variables in a single CPU word (32 or 64 depending on the CPU architecture).

2.2 Macromolecular Surfaces

In the CPK volumetric model a molecule is represented as a set of spheres that can overlap, each sphere representing a single atom in the molecule. The sphere representing an atom will have a radius equal to the van der Waals radius of that particular atom. The union of these spheres gives the CPK model for the molecule. For proteins and other macromolecules it is clear that the most of their van der Waals surface is buried in the inside of the molecules and is not accessible to the solvent or possible ligands. Thus the need to define the Solvent Accessible and the Solvent Excluded surfaces.

The Solvent Accessible surface (SAS) is defined as the geometric locus of the center of a probe sphere (representing the solvent molecule) as it rolls over the Van der Waals surface of the molecule.

The Solvent Excluded surface (SES) (or molecular surface

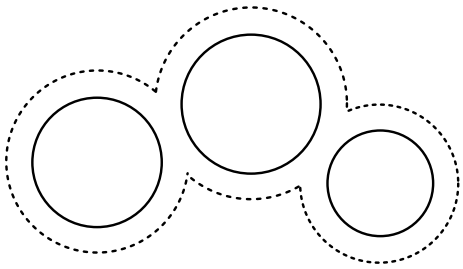


Fig. 1: The thick black line represents the van der Waals surface while the dashed one represents the SAS.

or Connolly surface) is defined as the locus of the inward-facing probe sphere as it rolls over the Van der Waals surface of the molecule. This surface can be considered as a continuous sheet consisting of two parts: the contact surface and the re-entrant surface. The contact surface is part of the van der Waals surface that is accessible to a probe sphere. The re-entrant surface is the inward-facing surface of the probe when it touches two or more atoms.

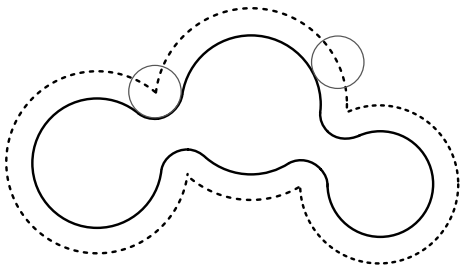


Fig. 2: The dashed line represents the SAS, the thick one represents the SES, while the two small circles drawn in gray represent the solvent probe-spheres.

There is a clear relation between the SAS and the SES, as the Solvent Accessible surface is displaced outward from the Solvent Excluded one by a distance equal to the probe radius.

Given the vdW or SAS voxelized volumetric representation of a macromolecule inside a voxel grid, it is easy to

derive its surface representation. The Solvent Excluded surface can be derived from the SAS exploiting the geometric relation between them, as the latter is displaced outward from the Solvent Excluded one by a distance equal to the probe radius.

Macromolecules can have solvent-excluded cavities and voids, which might generate spurious surfaces inside the real molecular surface. To overcome this issue we have implemented a simple three-dimensional flood-fill algorithm, which "colors" the voxels on the outside of the most external protein surface. The "coloring" process starts from one of the eight vertices of the voxel grid.

Algorithm 1 Flood-fill

```
queue<voxel> Q ← ∅ ▷ create an auxiliary empty queue
v_0 ← starting voxel
Q.push(v_0)
while not Q.empty() do
    voxel v = Q.front()
    Q.pop()
    for all neighbors n of v do
        if n is not occupied by any atom then
            color n
            Q.push(n)
        end if
    end for
end while
```

2.3 The Euclidean Distance Transform Method

In this section we will describe a method for the Solvent Excluded Surface calculation based on the Euclidean Distance Transform (EDT). The employment of the Euclidean Distance Transform for macromolecular surface calculation was first introduced in [31].

A distance transform (also known as distance map or distance field), is a derived representation of a digital image (usually a binary image). Distance maps are images where the value of each voxel (pixel if the image is 2D) of the foreground is the distance to the nearest voxel (pixel) of the background. A distance transform is qualified with the chosen metric. For example, one may speak of Euclidean distance transform, if the underlying metric is Euclidean distance.

Let I be a generic binary image, let S be the set of all background voxels of I (the set of all voxels set to 1) and let S^c be the set of all foreground voxels of I (the set of all voxels set to 0). Let $d(x, y)$ be the Euclidean distance between voxels x and y, where each voxel is identified by three integer coordinates. The Euclidean Distance Transform of I, $EDT(I)$, is a digital (grayscale) image, with the same dimensions as I, where each voxel $v = (x_i, y_i, z_i)$ in $EDT(I)$ contains the distance of voxel $v^* = (x_i, y_i, z_i) \in I$ from its nearest voxel in S, i.e. the distance from the nearest background (surface) voxel of the corresponding voxel in I. Clearly, all voxels of $EDT(I)$ whose corresponding voxels in I belong to S have a distance value of 0.

Let image I be the voxel grid, and the SAS the set of all background voxels S, and let us consider the EDT of the voxel grid $EDT(SAS)$. Because the Solvent Accessible surface is displaced outward from the Solvent Excluded one by a distance equal to the probe radius, it is clear that the SES can be obtained from the $EDT(SAS)$ by extracting all the voxels inside the Solvent Accessible volume with a distance map value equal to the probe radius.

The generation of distance maps using the Euclidean distance metric is a complex problem and several distance transform algorithms have been proposed [32, 33], offering various trade-offs between computation time and quality of the approximation of the Euclidean metric. Reviews on the different algorithms and techniques for the EDT calculation can be found in [34–36].

2.3.1 Region-growing EDT

The first observation about the EDT-based method is that we do not need the distance values for the whole voxel grid, but only for a subset of the voxels composing the Solvent Accessible volume. In particular, we only need the values of the ones inside the Solvent Accessible volume, which have a distance value smaller or equal to the probe-sphere radius. This problem is better solved by an algorithm which is progressive, in the sense that it can be stopped at any time and provide a sensible result, either by computing only within a certain distance, or improving the precision of the map by increasing the number of iterations. This implementation of the Euclidean Distance Transform calculation algorithm is based on the Region Growing method proposed by Cuisenaire in [37].

The method we implemented uses masks like the Chamfer DT ($3 \times 3 \times 3$ and $5 \times 5 \times 5$ masks), but, instead of raster scans, voxels are scanned by increasing distance value. This is implemented with a data-structure called Hierarchical Queues (HQ). Hierarchical Queues are made of a collection of FIFO queues. In-going elements enter any of the queues, outgoing elements are taken from the non-empty queue with the smallest label. The queue labeled i in the HQ contains the voxels for which i is the square of the distance to their nearest boundary voxel, which is clearly an integer with an Euclidean distance metric. For each voxel in the HQ, its location and its nearest boundary voxel are stored. A map data-structure is also created in order to store the squared distances for each voxel. Algorithm 2 formally summarizes this procedure.

2.4 The slicing procedure

To enable the computation of high resolution surfaces in spite of memory limitations we have developed a slicing

Algorithm 2 Region-Growing EDT using two HQs

$HQ_1(0) \leftarrow$ all voxels in the SAS
$HQ_1(i) \leftarrow \emptyset,\ \forall i > 0$
$HQ_2(i) \leftarrow \emptyset,\ \forall i$
$map[v] \leftarrow 0,\ \forall v \in SAS$
$map[v] \leftarrow MAXINT,\ \forall v \notin SAS$
$NBP(v) \leftarrow v,\ \forall v \in SAS$
while $HQ_1 \neq \emptyset$ **do**
  Extract voxel w from HQ_1
  for all $3\times3\times3$ neighbors n of
    $w : w \in SA$ volume **do**
    $d \leftarrow dist^2(NBP(w), n)$
    if $d < map[n]$ **then**
      $map[n] \leftarrow d$
      add n to $HQ_1(d)$
      $NBP(n) \leftarrow NBP(w)$
    end if
  end for
  // none of the neighbors of w was put in HQ_1
  if w is end && $d \geq 24$ **then**
    add w to $HQ_2(map[w])$
  end if
end while
while $HQ_2 \neq \emptyset$ **do**
  Extract voxel v from HQ_2
  for all $5\times5\times5$ neighbors n of
    $v : v \in SA$ volume **do**
    $d \leftarrow dist^2(NBP(v), n)$
    if $d < map[n]$ **then**
      $map[n] \leftarrow d$
      add n to $HQ_2(d)$
      $NBP(n) \leftarrow NBP(v)$
    end if
  end for
end while

protocol for the macromolecule. The molecule is sliced in a user-defined number of parts, and the surface is calculated separately for each part, in a sequential fashion. The slicing is done with planes perpendicular to the x-axis of the Cartesian coordinate system specified by the PDB.

Atom coordinates parsed from the PDB file are translated, scaled and quantized to the coordinate system defined by each slice. For each slice, we subtract the slice-length to the x coordinate of the translation vector $k-1$ times, where k is the current slice index ($k = 1, 2, ..., n$). The space filling procedure is performed for each slice separately, also taking into account any portions of atoms intersecting the slice whose centers might be located outside the current slice. Once the volumetric model for the current slice is created, the surface calculation proceeds as previously described with a few important differences.

2.4.1 Slice margin

For each slice we must consider some extra margin on the x coordinate in order for the surface computation to yield correct results.

Figure 3 depicts the calculation of the distance map for an intra-slice border region with no margin. The light blue squares represent the free voxels, the black squares represent the voxels belonging to the SA surface (or boundary) and the green squares correspond to the voxels inside the SA volume. The values shown in the figure are the squared Euclidean distances of the voxels from their Nearest Boundary Point. Clearly, all surface voxels have a zero distance map value. The dashed line represents the slicing plane, and the yellow square represent a voxel with an erroneous squared distance value, as the algorithm fails to correctly detect its Nearest Boundary Point.

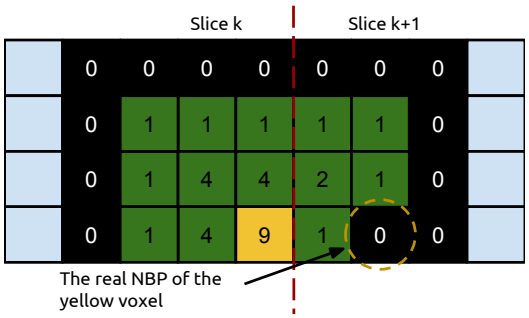


Fig. 3: The yellow square represents a voxel with an erroneous Distance Map value.

To guarantee a correct calculation of the Distance Map values for the voxels inside the slice volume, the margin size must be greater than the scaled and quantized probe-sphere radius.

2.4.2 Correct identification of cavities

The slicing protocol introduces another issue regarding the correct identification of solvent excluded cavities. When no slicing is applied, the proposed algorithm can correctly identify solvent excluded cavities and voids, which might lead to spurious surfaces. The slicing procedure complicates this step in the algorithm, as one must distinguish pockets from solvent excluded cavities when slicing planes passes through them.

The tool extracts all surface voxels belonging to potential pockets from each slice and store them in an apposite data-structure. Starting from the first two slices, the border voxels of the candidate pockets are matched against their neighbors on the other slice. If the border voxels have free neighbors on the adjacent slice, it means that the candidate pocket we are examining is in fact a real pocket (because it is solvent accessible), and thus its surface voxels must be conserved. Otherwise, border voxels belonging to another candidate

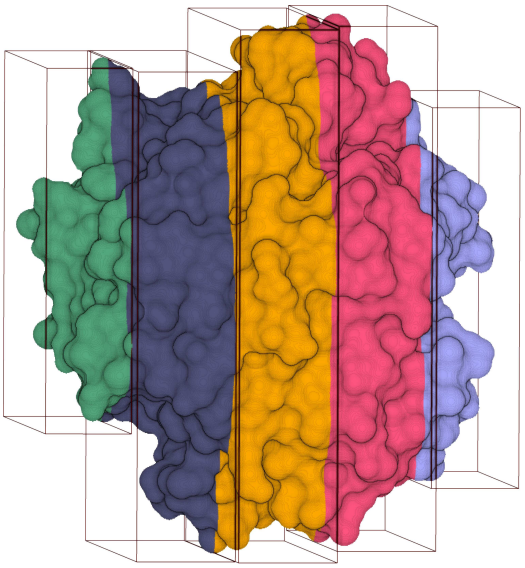


Fig. 4: SES of 1GZX.pdb, calculated with 5 slices, 1.4 Å probe-radius, $10^3 = 1000$ voxels per Å^3 resolution.

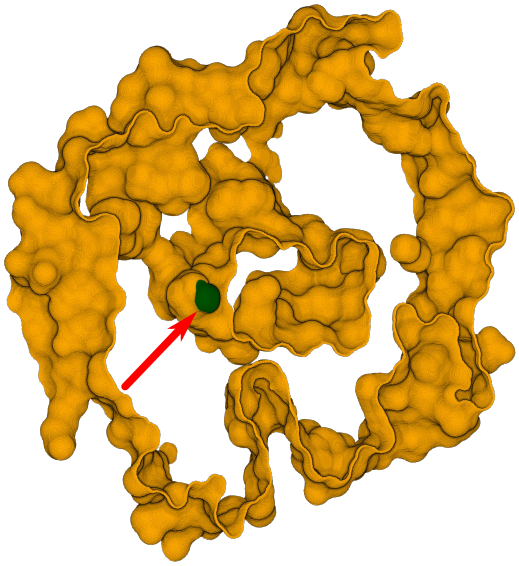


Fig. 5: Slice 3 - SES of 1GZX.pdb - 5 slices calculation. The pocket voxels of the fourth slice are correctly detected. 1.4 Å probe-radius, $10^3 = 1000$ voxels per Å^3

pocket on the next slice can be found. Because a pocket could run through two or more slices in length, we cannot discard these surface voxels yet. The procedure continues with the matching of potential pocket borders on adjacent slices in increasing index order.

After all slices have been scanned two-by-two in increasing index order, candidate pockets are either recognized as solvent accessible, or remain undetermined, as the same pocket could run through multiple slices. Slices must also be scanned in decreasing index order to resolve possible undetermined situations. When both the forward and backward scans are completed, the candidate pockets that have not been recognized as solvent accessible can be discarded, as they surely belong to solvent excluded cavities.

3. Parallelization

The macromolecular surface calculation protocol with slicing introduced in the previous sections suggests an immediate parallelization scheme. The surface calculation for each slice can be executed nearly-independently from the others, as process synchronization and communication is required only for the pocket-detection and extraction procedure, in order to correctly identify pockets spanning between two or more slices.

To obtain the best results in terms of speedup, the slicing procedure should guarantee a uniform distribution of the workload between processes. The volumetric model creation and the Euclidean Distance Map calculation procedures both depend on the number of atoms composing the molecule and the resolution, while the surface extraction and internal cavity filling procedures mainly depend on the voxel grid size (which in turn depends on the size of the molecule and the resolution). We have experimentally determined that the best speedup values are obtained with a uniform distribution of the number of atoms per slice (i.e. variable-length slices), instead of employing a constant slice length value.

We have run different tests on an IBM®Power®P770 Server with 6 IBM®Power7 CPU's and 640Gb of RAM, running SUSE Linux Enterprise 11, and experimentally determined the speedup values for different input molecules at various resolutions, while calculating the three molecular surfaces.

The experimental speedup values that follow were calculated on the average execution time of different tests. The same configuration (PDB entry, desired molecular surface, resolution, probe radius, number of CPUs) has been executed 100 times, and the speedup was derived from the average calculation time of these runs. We progressively increased the number of processors (from 1 to 64 CPUs) and evaluated the mean calculation time for each configuration.

For a given PDB entry, tests have shown that the calculation of the three molecular surfaces, at the same resolution, wields different speedup values for each surface (figures 6 and 7). Figure 6 shows the speedup values obtained while calculating the van der Waals and Solvent Excluded surfaces of the 1GZX PDB entry (Crystal Structure of T State Haemoglobin with Oxygen Bound At All Four Haems) [38] at a resolution of 1000 voxels per Å^3.

The vdW surface calculation has a higher speedup. The CPK model (volumetric model) creation takes most of the computation time in the vdW surface calculation, which mainly depends on the number of atoms per slice. On the other hand, in the SES calculation, the space-filling

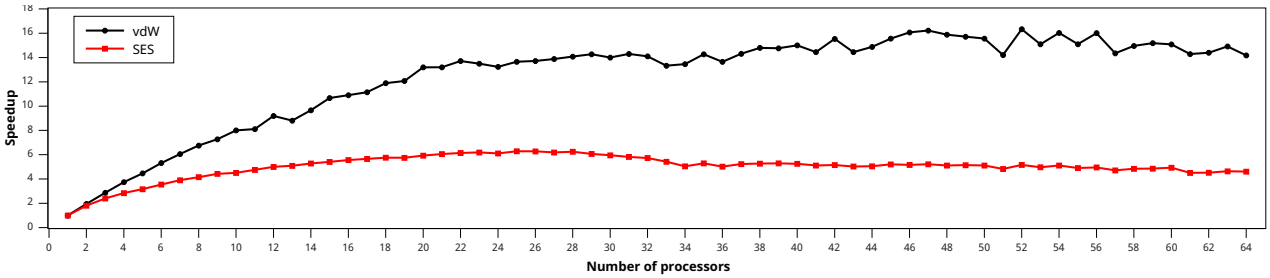


Fig. 6: van der Waals and Solvent Excluded surface calculation speedup for 1GZX.pdb at 1000 voxels per Å^3.

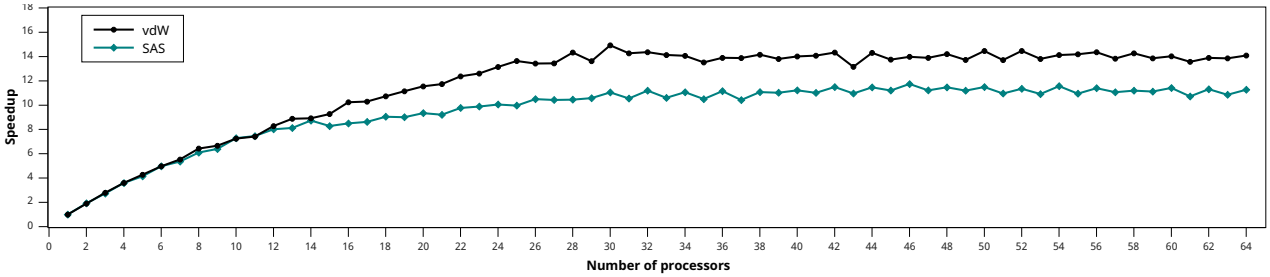


Fig. 7: van der Waals and Solvent Accessible surface calculation speedup for 2AEB.pdb [39] at 1000 voxels per Å^3.

algorithm used in the solvent excluded cavity detection is the most time consuming task, which mainly depends on the size of the slice. The vdW surface calculation also has no cavity detection step, and thus, no communications/synchronization between processes is needed.

Another important characteristic that greatly affects the speedup consists in the constant margin introduced in each slice during the SES calculation. At some point, while increasing the number of processors, the margin size will eventually become comparable to the effective size of the slice, thus vanishing the benefits of further parallelization.

4. Conclusion

We have effectively developed a tool which can produce voxelized macromolecular surfaces at very hight resolutions, even when faced with limited memory availability. For instance, the calculation of the surface for 1GZX at a resolution of 9000 voxels per Å^3 and while dividing the molecule in 10 slices, needs nearly 5GB of RAM, against the 2.6GB used while dividing the molecule in 20 slices (tests were made on a desktop computer with an Intel®Core™i7 860 CPU and 8GB of RAM (4×2GB DDR3-1333 banks) running Ubuntu 13.10 x64), which is an easily affordable amount of memory nowadays in desktop computers. By tuning the resolution and number-of-slices parameters various memory utilization rates can be achieved, depending on the users' needs.

The parallel implementation introduces advantages in terms of the overall speedup, however the uniform distribution of atoms per slice may not necessarily yield a balanced workload between processes. On the other hand, the constant slice margin represents the main limitation to this parallelization scheme as it introduces constant overhead regardless of the slice size. Addressing these issues in such an approach remains challenging.

Acknowledgements

This work has been partially supported by the University of Padova ex60% grant "Advanced Applications in Computer Science".

References

[1] D. Whitley, "Van der Waals surface graphs and molecular shape," *J. Math. Chem.*, vol. 23, no. 3-4, pp. 377–397, 1998.
[2] B. Lee and F. Richards, "The interpretation of protein structures: Estimation of static accessibility," *J. Mol. Biol.*, vol. 55, no. 3, pp. 379 – IN4, 1971.
[3] M. L. Connolly, "Analytical molecular surface calculation," *J. Appl. Crystallogr.*, vol. 16, no. 5, pp. 548–558, Oct. 1983.
[4] ——, "The molecular surface package," *J. Mol. Graph.*, vol. 11, no. 2, pp. 139–141, 1993.
[5] M. F. Sanner, A. J. Olson, and J.-C. Spehner, "Fast and Robust Computation of Molecular Surfaces," in *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, ser. SCG '95. New York, NY, USA: ACM, 1995, pp. 406–407.
[6] K. Kinoshita and H. Nakamura, "Identification of the ligand binding sites on the molecular surface of proteins," *Protein Sci.*, vol. 14, no. 3, pp. 711–718, 2005.
[7] L.-P. Albou, B. Schwarz, O. Poch, J. M. Wurtz, and D. Moras, "Defining and characterizing protein surface using alpha shapes," *Proteins: Struct. Funct. Bioinform.*, vol. 76, no. 1, pp. 1–12, 2009.

[8] J. C. Mitchell, R. Kerr, and L. F. T. Eyck, "Rapid atomic density methods for molecular shape characterization," *J. Mol. Graph. Model.*, vol. 19, no. 3-4, pp. 325–330, 2001.

[9] M. Bock, G. M. Cortelazzo, C. Ferrari, and C. Guerra, "Identifying Similar Surface Patches on Proteins Using a Spin-Image Surface Representation," in *Combinatorial Pattern Matching*, ser. Lecture Notes in Computer Science, A. Apostolico, M. Crochemore, and K. Park, Eds. Springer Berlin Heidelberg, 2005, vol. 3537, pp. 417–428.

[10] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. A. Friesem, C. Aflalo, and I. A. Vakser, "Molecular surface recognition: Determination of geometric fit between proteins and their ligands by correlation techniques," *Proc. Natl. Acad. Sci. USA*, vol. 89, no. 6, pp. 2195–2199, 1992.

[11] J. Esquivel-Rodriguez, V. Filos-Gonzalez, B. Li, and D. Kihara, "Pairwise and Multimeric Protein-Protein Docking Using the LZerD Program Suite," *Protein Struct. Predict.*, vol. 1137, pp. 209–234, Apr. 2014.

[12] J. Esquivel-Rodriguez and D. Kihara, "Effect of conformation sampling strategies in genetic algorithm for multiple protein docking," *BMC Proc.*, vol. 6, no. Suppl 7, p. S4, 2012.

[13] ——, "Evaluation of multiple protein docking structures using correctly predicted pairwise subunits," *BMC Bioinform.*, vol. 13, no. Suppl 2, p. S6, 2012.

[14] J. Esquivel-Rodriguez, Y. D. Yang, and D. Kihara, "Multi-LZerD: Multiple protein docking for asymmetric complexes," *Proteins: Struct. Funct. Bioinform.*, vol. 80, no. 7, pp. 1818–1833, 2012.

[15] D. Kihara, L. Sael, R. Chikhi, and J. Esquivel-Rodriguez, "Molecular Surface Representation Using 3D Zernike Descriptors for Protein Shape Comparison and Docking." *Curr. Protein Pept. Sci.*, vol. 12, no. 6, pp. 520–533, 2011.

[16] B. Li and D. Kihara, "Protein docking prediction using predicted protein-protein interface," *BMC Bioinform.*, vol. 13, no. 1, p. 7, 2012.

[17] L. Sael and D. Kihara, "Improved protein surface comparison and application to low-resolution protein structure data," *BMC Bioinform.*, vol. 11, no. Suppl 11, p. S2, 2010.

[18] V. Venkatraman, Y. Yang, L. Sael, and D. Kihara, "Protein-protein docking using region-based 3D Zernike descriptors," *BMC Bioinform.*, vol. 10, no. 1, p. 407, 2009.

[19] Z. A. Deeb, D. A. Adjeroh, and B.-H. Jiang, "Protein Surface Characterization Using an Invariant Descriptor," *Int. J. Biomed. Imaging*, vol. 2011, p. 15, 2011.

[20] S. Yin, E. A. Proctor, A. A. Lugovskoy, and N. V. Dokholyan, "Fast screening of protein surfaces using geometric invariant fingerprints," *Proc. Natl. Acad. Sci. USA*, vol. 106, no. 39, pp. 16 622–16 626, Sept. 2009.

[21] S. Grandison, C. Roberts, and R. J. Morris, "The Application of 3D Zernike Moments for the Description of "Model-Free" Molecular Structure, Functional Motion, and Structural Reliability." *J. Comput. Biol.*, vol. 16, no. 3, pp. 487–500, 2009.

[22] L. Sael, D. La, B. Li, R. Rustamov, and D. Kihara, "Rapid comparison of properties on protein surface." *Proteins*, vol. 73, no. 1, pp. 1–10, Oct. 2008.

[23] M. Weisel, E. Proschak, and G. Schneider, "PocketPicker: analysis of ligand binding-sites with shape descriptors," *Chem. Cent. J.*, vol. 1, no. 1, p. 7, 2007.

[24] D. G. Levitt and L. J. Banaszak, "POCKET: A computer graphics method for identifying and displaying protein cavities and their surrounding amino acids," *J. Mol. Graph.*, vol. 10, no. 4, pp. 229–234, 1992.

[25] M. Hendlich, F. Rippmann, and G. Barnickel, "LIGSITE: Automatic and efficient detection of potential small molecule-binding sites in proteins," *J. Mol. Graph. Model.*, vol. 15, no. 6, pp. 359–363, 1997.

[26] B. Li, S. Turuvekere, M. Agrawal, D. La, K. Ramani, and D. Kihara, "Characterization of local geometry of protein surfaces with the visibility criterion," *Proteins: Struct. Funct. Bioinform.*, vol. 71, no. 2, pp. 670–683, 2008.

[27] H. Berman, K. Henrick, and H. Nakamura, "Announcing the worldwide Protein Data Bank," *Nat. Struct. Mol. Biol.*, vol. 10, no. 12, p. 980, Dec. 2003.

[28] R. B. Corey and L. Pauling, "Molecular Models of Amino Acids, Peptides, and Proteins," *Rev. Sci. Instrum.*, vol. 24, no. 8, pp. 621–627, 1953.

[29] Point Cloud Library Documentation, "The PCD (Point Cloud Data) file format," Accessed 29 March 2015. [Online]. Available: http://goo.gl/6tNW0P

[30] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 1–4.

[31] D. Xu and Y. Zhang, "Generating Triangulated Macromolecular Surfaces by Euclidean Distance Transform," *PLoS ONE*, vol. 4, no. 12, p. e8140, Dec. 2009.

[32] P.-E. Danielsson, "Euclidean Distance Mapping," *Comput. Graph. Image Process.*, vol. 14, pp. 227–248, 1980.

[33] G. Borgefors, "Distance Transformations in Digital Images," *Comput. Vis. Graph. Image Process.*, vol. 34, no. 3, pp. 344–371, June 1986.

[34] O. Nilsson and A. Söderström, "Euclidean Distance Transform Algorithms: A comparative study," Linköping University, Department of Science and Technology, The Institute of Technology, Digital Media, Tech. Rep., 2007. [Online]. Available: http://goo.gl/MYALj6

[35] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, "2D Euclidean Distance Transform Algorithms: A Comparative Survey," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 2:1–2:44, Feb. 2008.

[36] G. Grevera, "Distance Transform Algorithms And Their Implementation And Evaluation," in *Deformable Models*, ser. Topics in Biomedical Engineering. International Book Series. Springer New York, 2007, pp. 33–60.

[37] O. Cuisenaire, "Region growing Euclidean distance transforms," in *Image Analysis and Processing*, ser. Lecture Notes in Computer Science, A. Bimbo, Ed. Springer Berlin Heidelberg, 1997, vol. 1310, pp. 263–270.

[38] M. Paoli, R. Liddington, J. Tame, A. Wilkinson, and G. Dodson, "Crystal Structure of T State Haemoglobin with Oxygen Bound At All Four Haems," *J. Mol. Biol.*, vol. 256, no. 4, pp. 775–792, 1996.

[39] L. Di Costanzo, G. Sabio, A. Mora, P. C. Rodriguez, A. C. Ochoa, F. Centeno, and D. W. Christianson, "Crystal structure of human arginase I at 1.29-Å resolution and exploration of inhibition in the immune response," *Proc. Natl. Acad. Sci. USA*, vol. 102, no. 37, pp. 13 058–13 063, 2005.

Distributed Two-Dimensional Guided Loop Self-Scheduling for Heterogeneous Computer Systems

Satish Penmatsa[1] **and Akash Laddha**[2]
[1]Department of Computer Science, Framingham State University, Framingham, MA, USA
[2]Staples Inc., Framingham, MA, USA

Abstract - *Modern distributed computing systems like the grid and cloud computing systems are a viable and less expensive alternative to parallel computers for executing computation intensive scientific applications. The performance of these applications can be maximized by providing application-level load balancing of loops inside them which are one of the largest sources of parallelism. In this paper, we implement a distributed two-dimensional guided self-scheduling (DGSS-2D) scheme whose objective is to efficiently schedule the loop iterations of an application for achieving a load balanced execution that minimizes the loop completion time. DGSS-2D is implemented and its performance evaluated using the Stampede high performance computing cluster at the Texas Advanced Computing Center of the University of Texas at Austin. Experimental results show that there is a substantial performance improvement in the loop execution time by using DGSS-2D when compared to the one-dimensional distributed guided self-scheduling scheme.*

Keywords: Loop scheduling, Distributed computing, Heterogeneous systems.

1 Introduction

Computation intensive applications often consist of parallel code which when scheduled for concurrent execution can reduce the total execution of the application significantly. Parallel code can be in the form of loops with or without any dependencies between the loop iterations. Parallel loops with dependencies are commonly known as DOALL loops and loops with dependencies among the iterations are known as DO ACROSS loops [3]. Scheduling the loop iterations for concurrent execution requires multiple computing resources (processors).

Parallel computers can be used for concurrent execution of an application. However, obtaining and maintaining them can be very expensive. Distributed computing systems are a viable and less expensive alternative to parallel computers. The performance of scientific applications on the underlying distributed system can be maximized by providing application-level load balancing of loops inside them. However, scheduling scientific applications in large-scale distributed systems for achieving a load balanced execution that minimizes the loop completion time is not straightforward. Factors such as the non-uniformity of iterate execution times, geographic distribution and heterogeneity of the computing and communication resources, communication and synchronization bottlenecks, failures such as processor or link failures, and sharing of the resources by multiple users lead to application performance degradation.

Loop scheduling schemes which do not take the heterogeneity of a distributed system into account are called 'simple' schemes whereas the schemes that take the heterogeneity of the system into account are called 'distributed' schemes. Also, depending on when the scheduling decisions are made, loop scheduling can be categorized into 'static' and 'dynamic' [3, 7]. Static scheduling schemes determine the task allocation to the processors prior to the execution of the application. Dynamic scheduling (or self-scheduling) is an automatic loop scheduling method in which idle processors request new loop iterations to be assigned to them during run time.

Various loop scheduling schemes have been proposed and analyzed in the past. For example, please see [1 – 6, 14, 15] and references therein. Most of the previously studied loop scheduling schemes partition only the outermost loop of a program loop structure and assign tasks (chunks of iterations) to the processors. This is not efficient for multi-dimensional nested loops.

Studies on two-dimensional (2D) loop scheduling can be found in [7, 9, 10] and references therein. Most of the 2D schemes do not take the heterogeneity of the system into account. A two-dimensional distributed trapezoid self-scheduling scheme has been studied in [7] and a two-dimensional distributed factoring self-scheduling scheme has been studied in [9]. In this paper, we implement a two-dimensional distributed guided self-scheduling (DGSS-2D) scheme and compare its performance with one-dimensional distributed guided self-scheduling scheme (DGSS-1D).

2 Two-Dimensional Distributed Guided Self Scheduling (DGSS-2D)

Guided Self – Scheduling (GSS) [4, 8] is a dynamic scheme with a non-linear chunk-size function. It assigns large chunks (set of iterations) initially, which implies reduced communication/scheduling overheads in the first scheduling steps. A modified version GSS(*l*) with minimum assigned chunk-size *l* attempts to improve on the weaknesses of GSS.

One-Dimensional Distributed Guided Self-Scheduling scheme (DGSS-1D) partitions only the outermost loop of a nested loop construct. Two-Dimensional Distributed Guided Self-Scheduling scheme (DGSS-2D) partitions both the outer loop and the inner loop of a two-level nested loop construct. The above schemes were implemented using *Master-Worker* architecture [3, 7].

In the following, we present the DGSS-2D algorithm. The methodology for computing the two-dimensional chunks is similar to the one described in [7]. The two-dimensional chunks will be allocated to the worker processors (PEs) by the master PE based on the worker *available powers* [3]. A worker with higher available power will be allocated more chunks than compared to a worker with lower available power.

ALGORITHM: Two-Dimensional Distributed Guided Self-Scheduling Scheme (DGSS-2D)

MASTER

1. (a) Receive processor speeds (P_j) from the worker PEs ($j = 1,\ldots,p$).

   (b) Compute processor *Available (Virtual) Powers*, V_j using worker PE workloads.

   (c) Send V_j to the worker PEs.

2. (a) While there are unassigned iterations, if a request comes, put it in a queue.

   (b) Compute the rectangular chunks and *istart1*, *istart2* [7] using DGSS-1D.

   (c) Pick a request from queue with virtual powers V_j and assign next V_j rectangular chunk along same or adjacent wavefront diagonals.

WORKER

1. (a) Send processor speed (P_j) to the Master PE.

   (b) Receive Virtual Power (V_j) from Master PE

2. Send a request for work (chunks of loop iterations).

3. (a) Wait for a response from Master.

   (b) If more tasks arrive, compute the new task, and go to Step 2. Else, *Terminate*.

3 Implementation and Results

The scheduling schemes (DGSS-2D and DGSS-1D) were implemented using the Message Passing Interface [11] on the Stampede [13] high performance computing cluster at the Texas Advanced Computing Center of the University of Texas at Austin. Stampede is one of the largest computing systems in the world for open science research.

The test problem used for the experiments is the Mandelbrot Computation [12]. The Mandelbrot Computation is a doubly-nested loop without any dependencies. The schemes are implemented with the number of worker processors (PEs) ranging from 1 to 16 and the Mandelbrot computation sizes ranging from 16000 x 16000 to 32000 x 32000.

To create a heterogeneous environment, we put an artificial load (one continuously running matrix multiplication process) in the background on half of the worker processors. The workers with one load in the background are assumed to have virtual power of 1 and the workers without any load are assumed to have virtual power of 2. Thus, we have half fast and half slow worker processors.

In the following, we present the experimental results for various problem sizes and number of worker processors. T_p denotes the total (parallel) execution time (for a given problem size) measured on the master processor. The times presented for the worker processors are their total compute times for the iterations assigned to them by the master processor. The units used in the following are as follows - *m*: minutes; *s*: seconds; and *ms*: milliseconds.

Table 1 presents the total execution time (T_p) for a problem size of 16000 × 16000 using only 1 worker PE. Since there is only one PE which does all the computation, the total time is almost the same for DGSS-1D and DGSS-2D (irrespective of the chunks computation and allocation by the scheduling schemes).

Table 1: Worker PEs: 1, Problem Size: 16000x16000

PE	DGSS-1D	DGSS-2D
1	1m 50s 173ms	1m 50s 139ms
T_p	1m 50s 173ms	1m 50s 139ms

Table 2 presents the T_p of DGSS-1D and DGSS-2D with 2 worker PEs for a problem size of 16000 × 16000. The table also shows the computation times of each worker PE. It can be observed that the T_p achieved by DGSS-2D is much lower than that of DGSS-1D. It can also be observed that the worker computation times in the case of DGSS-2D are well balanced compared to DGSS-1D. In the case of DGSS-2D, each worker PE has a computation time of about 55s where as in the case of DGSS-1D, one worker computation time is about 30s and the other worker computation time is about 80s.

Table 2: Worker PEs: 2, Problem Size: 16000x16000

PE	DGSS-1D	DGSS-2D
1	30s 665ms	55s 83ms
2	1m 19s 494ms	55s 83ms
T_p	1m 19s 494ms	55s 83ms

Table 3 presents the T_p and worker PE computation times of DGSS-1D and DGSS-2D for a problem size of 16000 × 16000 with 4 worker PEs. It can be observed that the T_p achieved by DGSS-2D is much lower than that of DGSS-1D. Also, the worker computation times in the case of DGSS-2D are well balanced (ranging from about 23s to 32s) compared to DGSS-1D (ranging from about 16s to 39s).

Table 3: Worker PEs: 4, Problem Size: 16000x16000

PE	DGSS-1D	DGSS-2D
1	23s 452ms	32s 761ms
2	16s 559ms	30s 99ms
3	30s 270ms	23s 636ms
4	39s 880ms	23s 636ms
T_p	39s 880ms	32s 761ms

Table's 4 and 5 present the T_p and worker PE computation times for problem sizes 24000 × 24000 and 32000 × 32000 with 4 worker PEs. It can again be observed that the T_p in the case of DGSS-2D is lower than that of DGSS-1D and the worker PE computation times of DGSS-2D are better balanced than that of DGSS-1D.

Table 4: Worker PEs: 4, Problem Size: 24000x24000

PE	DGSS-1D	DGSS-2D
1	52s 768ms	1m 13s 715ms
2	37s 250ms	1m 7s 717ms
3	1m 8s 85ms	53s 169ms
4	1m 29s 738ms	53s 169ms
T_p	1m 29s 738ms	1m 13s 715ms

Table 5: Worker PEs: 4, Problem Size: 32000x32000

PE	DGSS-1D	DGSS-2D
1	1m 33s 751ms	2m 11s 29ms
2	1m 6s 213ms	2m 0s 404ms
3	2m 1s 73ms	1m 34s 522ms
4	2m 39s 554ms	1m 34s 522ms
T_p	2m 39s 554ms	2m 11s 29ms

Table's 6 through 12 present the T_p and worker PE computation times of DGSS-1D and DGSS-2D for problem sizes ranging from 16000 × 16000 to 32000 × 32000 with worker PEs ranging from 8 to 16. In all cases, it can be observed that the T_p in the case of DGSS-2D is lower than that of DGSS-1D and the worker PE computation times of DGSS-2D are better balanced than that of DGSS-1D.

Table 6: Worker PEs: 8, Problem Size: 16000x16000

PE	DGSS-1D	DGSS-2D
1	14s 830ms	15s 618ms
2	11s 944ms	14s 979ms
3	9s 20ms	14s 145ms
4	9s 19ms	13s 466ms
5	9s 19ms	12s 901ms
6	17s 206ms	13s 321ms
7	18s 799ms	12s 901ms
8	20s 415ms	12s 999ms
T_p	20s 416ms	15s 618ms

Table 7: Worker PEs: 8, Problem Size: 24000x24000

PE	DGSS-1D	DGSS-2D
1	33s 298ms	35s 14ms
2	26s 848ms	30s 174ms
3	20s 267ms	34s 982ms
4	20s 285ms	29s 484ms
5	20s 286ms	29s 964ms
6	38s 690ms	29s 484ms
7	42s 281ms	29s 484ms
8	45s 955ms	29s 484ms
T_p	45s 959ms	35s 6ms

Table 8: Worker PEs: 8, Problem Size: 32000x32000

PE	DGSS-1D	DGSS-2D
1	59s 234ms	1m 2s 223ms
2	47s 745ms	53s 673ms
3	36s 43ms	1m 2s 180ms
4	36s 40ms	52s 346ms
5	36s 40ms	53s 246ms
6	1m 8s 743ms	52s 346ms
7	1m 15s 104ms	52s 346ms
8	1m 21s 729ms	52s 346ms
T_p	1m 21s 729ms	1m 2s 220ms

Table 9: Worker PEs: 12, Problem Size: 16000x16000

PE	DGSS-1D	DGSS-2D
1	10s 189ms	9s 466ms
2	9s 90ms	9s 686ms
3	7s 976ms	9s 335ms
4	6s 108ms	9s 290ms
5	6s 281ms	8s 839ms
6	5s 864ms	9s 392ms
7	5s 17ms	8s 839ms
8	9s 319ms	8s 839ms
9	11s 680ms	9s 399ms
10	11s 874ms	8s 839ms
11	13s 573ms	9s 195ms
12	13s 529ms	9s 309ms
T_p	13s 548ms	9s 680ms

Table 10: Worker PEs: 12, Problem Size: 24000x24000

PE	DGSS-1D	DGSS-2D
1	22s 890ms	19s 762ms
2	20s 385ms	20s 389ms
3	17s 884ms	21s 757ms
4	13s 762ms	22s 324ms
5	14s 81ms	20s 229ms
6	13s 153ms	23s 545ms
7	11s 261ms	20s 282ms
8	20s 953ms	19s 762ms
9	26s 228ms	19s 804ms
10	26s 651ms	20s 708ms
11	30s 477ms	19s 762ms
12	30s 401ms	19s 762ms
T_p	30s 480ms	23s 545ms

Table 11: Worker PEs: 12, Problem Size: 32000x32000

PE	DGSS-1D	DGSS-2D
1	40s 660ms	35s 92ms
2	36s 228ms	36s 202ms
3	31s 765ms	38s 674ms
4	24s 417ms	39s 668ms
5	24s 980ms	35s 957ms
6	23s 355ms	41s 868ms
7	19s 940ms	36s 47ms
8	37s 153ms	35s 92ms
9	46s 611ms	35s 223ms
10	47s 340ms	36s 821ms
11	54s 147ms	35s 92ms
12	54s 38ms	35s 92ms
T_p	54s 149ms	41s 868ms

Table 12: Worker PEs: 16, Problem Size: 16000x16000

PE	DGSS-1D	DGSS-2D
1	7s 791ms	6s 941ms
2	6s 351ms	6s 726ms
3	7s 223ms	7s 228ms
4	5s 981ms	6s 726ms
5	5s 76ms	7s 424ms
6	4s 44ms	7s 186ms
7	4s 766ms	7s 17ms
8	4s 237ms	7s 402ms
9	8s 641ms	6s 726ms
10	9s 72ms	6s 788ms
11	10s 886ms	6s 726ms
12	4s 33ms	6s 726ms
13	5s 560ms	6s 728ms
14	9s 982ms	6s 726ms
15	7s 843ms	6s 726ms
16	8s 795ms	6s 726ms
T_p	10s 889ms	7s 424ms

4 Conclusions

In this paper, we implemented a distributed two-dimensional guided self-scheduling (DGSS-2D) scheme whose objective is to efficiently schedule the loop iterations of an application by considering the system heterogeneity for achieving a load balanced execution that minimizes the loop completion time. DGSS-2D is implemented and its performance evaluated using the Stampede high performance computing cluster. Results showed that DGSS-2D performs better compared to DGSS-1D and also present a more balanced load distribution of the workload among the computers in the cluster.

5 References

[1] I. Banicescu, V. Velusamy, and J. Devaprasad, "On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring", Cluster Computing, vol. 6, pp. 215–226, 2003.

[2] A. Kejariwal, A. Nicolau, and C. Polychronopoulos, "Historyaware self-scheduling", in International Conference on Parallel Processing, Columbus OH, Aug 2006, pp. 185–192.

[3] A. T. Chronopoulos, S. Penmatsa, J. Xu, and S. Ali, "Distributed loop-scheduling schemes for heterogeneous computer systems," Concurrency and Computation: Practice and Experience, vol. 18, no. 7, pp. 771–785, 2006.

[4] C. D. Polychronopoulos and D. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers", IEEE Transactions on Computers 1987; 36:1425–1439.

[5] F. M. Ciorba, I. Riakiotakis, T. Andronikos, G. Papakonstantinou, and A. T. Chronopoulos, "Enhancing self-scheduling algoritms via synchronization and weighting," Journal of Parallel and Distributed Computing, vol. 68, no. 2, pp. 246–264, 2008.

[6] J. Herrera, E. Huedo, R. S. Montero, and I. M. Llorente, "Loosely-coupled loop scheduling in computational grids," in Proc. of the 20th IEEE Intl. Parallel and Distributed Processing Symp., Rhodes Island, Greece, 2529 April 2006.

[7] A. T. Chronopoulos, L. M. Ni, and S. Penmatsa, "Multidimensional dynamic loop scheduling algorithms," in IEEE International Conference on Cluster Computing, Austin, TX, 17-20 Sept. 2007, pp. 241 – 248.

[8] T. L. Freeman, D. J. Hancock, J. M. Bull, and R. W. Ford, "Feedback guided scheduling of nested loops", Proceedings of the 5th International Applied Parallel Computing (PARA) Workshop, Bergen, Norway, 2000 (Lecture Notes in Computer Science, vol. 1947), Springer: Berlin, 2001; 149–159.

[9] S. Penmatsa and A. Oji, "Performance Evaluation of Two-Dimensional Distributed Factoring Self-Scheduling Scheme for Heterogeneous Computer Systems", Proceedings of the 19th International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, USA, July 22-25, 2013.

[10] S. Penmatsa, S. Grover, and G. S. Hura, "Load Distribution of Parallelizable Jobs in Computer Clusters", Journal of Global Information Technology, Vol. 6, No. 1, pp. 13-20, 2011.

[11] P. Pachecho, Parallel Programming with MPI. Morgan Kauffman, 1997.

[12] M. F. Bransley, R. L. Devaney, B. B. Mandelbrot, H. O. Peitgen, D. Saupe, R. F. Voss, Y. Fisher, and M. McGuire, The Science of Fractal Images. NY: Springer-Verlag, 1988.

[13] www.tacc.utexas.edu/stampede

[14] Y. Han and A. T. Chronopoulos, "A Resilient Hierarchical Distributed Loop Self-Scheduling Scheme for Cloud Systems", IEEE 13th International Symposium on Network Computing and Applications, pp. 80 – 84, 2014.

[15] Y. Han and A. T. Chronopoulos, "Distributed Loop Scheduling Schemes for Cloud Systems", IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp. 955 - 962, 2013.

GRALT: A MULTI-THREAD TOOL FOR TEXT VERSIONS ANALYSIS

Katia Mayfield
Dept. of Math, Computer and Natural Sciences,
Athens State Univ.,
Athens, AL USA

Ronald Marsh
Department of Computer Science
University of North Dakota,
Grand Forks, ND USA

Crystal Alberts
Department of English
University of North Dakota,
Grand Forks, ND USA

Abstract - *The existence of multiple versions of a text, regardless of how they were created (in print or digital), and the possibility of using a computer as an analytical tool has led to the development of software systems used by textual criticism scholars. Following the tradition of this type of work from initial efforts by IBM to today's Google, this study focuses on the multi-thread implementation of a tool supporting the study of text variations by textual critics.*

Keywords: multi-thread, wavefront, evolution, version, performance

1 INTRODUCTION

Textual critics examine changes caused by editing, authors' modifications, and even printing errors to discover the evolution of text variants. To distinguish between versions, scholars collate texts, which according to Williams and Abbot, is the process where one text is compared with another to discover textual variants [14]. Along with the comparison between texts, a scholar must also take into account that an author's manuscripts may also have notations or marginalia, such as dates, numbers, biographical or historical events, that make the distinction between versions possible. When the texts do not provide a clear idea of such a sequence, then the textual critic may depend on bibliographical studies to provide extra information. Historical, analytical and descriptive bibliographical studies examine published documents, like a book, in order to identify several attributes of those publications.

For example, analytical bibliographic scholars investigate various physical characteristics of a work, including manufacturing processes, ink components, and book binding techniques, to find significant information that will help them with their study. These studies can also be conducted with the goal of trying to establish a chronological sequence of document creation [14]. However, with the advancement of digital technology, new obstacles have surfaced in bibliographical studies. While digital technology leads to greater access to a manuscript's content, its physical characteristics cannot be accessed in the digital realm, at least not in the same way. As such, digital tools for digital objects are necessary. Due to the amount of data to be analyzed and the target user, these tools require high performance solutions to be applicable to office solutions. A parallel implementation based on multithread techniques is therefore the basis of this study.

2 BACKGROUND

A series of recently developed investigative tools, complement Shillingsburg's argument that a new set of characteristics need to be analyzed in documents that are represented in digital form, including hardware (from the processor to the monitor), the communication networks, the character sets, type fonts, mark-up, and formatting of the text [9]. Even blank spaces and printer paper--if the document is printed--become part of this analysis.

Several scholars, Michael Wise among them, have conducted studies on textual variants concentrating in the area of plagiarism [15]. However in the study of plagiarism, the texts are compared in search of segments that are plagiarized. These studies do not consider the possibility of targeting versions of the same work, which is the goal of this research.

Two well-known tools among humanists, the Versioning Machine, built by Susan Schreibman et al, and Juxta, created by the Applied Research in Patacriticism group at the University of Virginia, have been used in a number of projects [3, 12]. Scholars, such as Lara Vetter and Jarom McDonald in "Witnessing Dickinson's Witness," apply the Versioning Machine to demonstrate the complexity of Emily Dickinson's work with respect to the number of published variants [13]. The text is displayed through the Versioning Machine to allow for a comparison between versions of Dickinson's work and to show how Dickinson varied words, phrases, lines and stanzas. Juxta provides the user with histograms, which display the frequency of differences between the initial text to a specific

variant, allowing the users to identify sections of the text that have been heavily modified [3]. These tools provide a visualization of the differences between texts, but still require the manual intervention of the textual critic to establish the text evolution, which implies that even though the differences between documents can be seen by the critic, the tool does not provide any information on the evolution of the text.

3 GRALT: A MULTI-THREADED TOOL

This section presents GRALT, Graph Analyzer of Literary Text Versions, a multi-thread solution based on graph techniques, which has a polynomial time complexity. In order for GRALT to estimate the evolutionary sequence of the text versions, a text comparison graph is constructed to represent the differences between the texts [6, 7].

In order to measure the difference between texts, an editing metric technique is required. One of the most widely known text editing metrics is Levenshtein's Distance [5]. It calculates the least number of operations that are required to modify a text, at character level, and obtain a new one. The Levenshtein's algorithm shown in Figure 1, utilizes a dynamic programming technique in a two dimensional matrix to perform the required calculations. The texts being compared are aligned with the rows and columns of the matrix. The matrix is then populated from the upper left corner to the bottom right corner. The number found in the lower right corner is the Levenshtein distance between the documents.

The first approach to finding the evolutionary sequence by using graph traversal algorithms is based on concepts associated with a Hamiltonian path. Such graph nodes represent the text variants while the edges indicate the differences found in the text. In this study, it is assumed that the user has enough information to establish which node corresponds to the original (first draft) of the text, and then a greedy algorithm named Single Path Evolution (SPE), shown in Figure 2, is applied to the graph, adding edges to the solution path in a single direction. The algorithm assumes that the graph is fully connected.

A second approach involves the use of a Minimum Spanning Tree algorithm. Two common algorithms used to find a MST are the Kruskal's algorithm and the Prim's algorithm [8]. A simple pseudo code representation of the Kruskal's algorithm is shown in Figure 3.

Levenshtein's Algorithm:
input: text1 and text2
output: Levenshtein's distance
lev[0][j]=j for 0≤j≤ length(text2)
lev[i][0]=i for 0≤i≤ length(text1)
for (i = 1; i ≤ length(text1); i++)
{
for (j = 1; j ≤ length(text2); j++)
{
if(text1 [i -1] == text2 [j-1])
{
lev[i] [j] = lev [i – 1] [j – 1]
}
else
{
lev[i] [j] = minimum(lev[i-1][j],
lev[i][j-1], lev[i-1][j-1]) + 1
}
}
}
return lev[length(text1)] [length(text2)]

Figure 1 Levenshtein's Algorithm

SPE Algorithm
input: graph G=(V,E, W), original node s
output: shortest Hamiltonian path, H = (M, D)
D←Ø
M← {s}
V ←V - {s}
end_of_M ← s
While V ≠ Ø
select v in V with minimum weight from end_of_M
D ← D ∪ {(end of M - v)}
M←M ∪ {v}
V ←V - {v}
end_of_M ← v

Figure 2 SPE algorithm

Kruskal Algorithm
input: graph G=(V,E, W), original node s
output: Minimum Spanning Tree F=(M,T)
// create a forest (unconnected graph) F such that F= (M,T)
T←Ø
M ←V
sort E into non-decreasing order by weight w
for each (u, v) ∈ E taken from the sorted list
if (u,v) connects 2 different trees in F
then T ← T ∪ {(u, v)}
Return MST F = (V,T)

Figure 3 Kruskal's MST algorithm

The combined functionality of a modified Levenshtein algorithm and the version sequence estimation available through the SPE path and the MST algorithms, results in the GRALT algorithm shown in Figure 4. The algorithm produces an undirected graph and considering the assumption that the original is known, it can be transformed into a direct graph, introducing the directions necessary to obtain the estimated version evolution.

4 MULTI-THREAD DESIGN

Significant computer programming challenges, involving memory utilization and computational time, exist in the implementation of GRALT. It is important to implement a multithreaded version of the algorithm with dynamic memory allocation to support the different needs of data storage. The comparison of two texts using the modified Levenshtein's method requires the execution of a nested pair of loops whose total number of iterations may result in a very long execution time. In this study, given the size of some of the problems that are investigated, multi-thread programming based on a technique proposed by Lamport in 1974 to improve the execution of nested loops is used [4].

GRALT algorithm
Input: list of texts L, original text s ∈ L
Output: estimated evolutionary sequence of L
// build complete graph G=(V,E,W)
V←L
E←Ø
For all v ∈ V
E ← E ∪ {(v→u)| u ∈(V-{v})}
// compute editing distance between texts
For all v_i ∈V
    For all v_j ∈V, j>i
        $W(v_j,v_i)$ ←Levenshtein(v_i,v_j)
// find the SPE path of G
Estimated sequence ← SPE (G, s)
// or find the minimum spanning tree
Estimated sequence ← Kruskal (G,s)

Figure 4 GRALT Algorithm

This technique, known as the wavefront approach, uses a hyperplane concept to identify parallel code in uniform nested loops [4]. In particular, the modified Levenshtein's algorithm has a double nested loop equivalent to a two-dimensional space. Usually, the execution of such loop iterations follows a row-wise or column-wise sequence. In the modified Levenshtein's algorithm, each iteration depends on three values previously calculated: the first value is found in the same column and a previous row, the second on the same row and a previous column and the third in a diagonal, as represented by the arrows seen in Figure 5. The few values that are shown in the matrix are examples of the calculated distance based on the modified Levenshtein's algorithm.

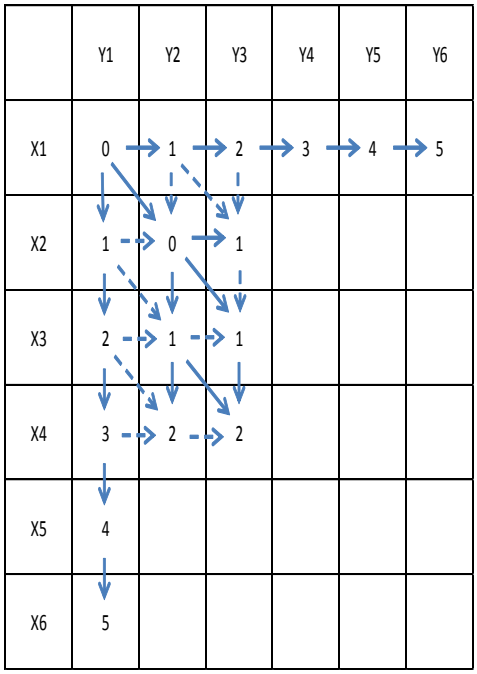


Figure 5 Levenshtein algorithm dependencies

In the Levenshtein's algorithm, it is clear that after iteration (1,1) has been completed, iterations (2,1) and (1,2) could be run in parallel, followed by iterations (3,1), (2,2), and (1,3). In this study, considering the synchronization required by the iterations (to obtain data previously computed), any attempt to parallelize individual iterations might cause undesirable overhead. A better solution is to apply this concept to groups of iterations as represented in Figure 6. A single thread runs the first block of iterations; the second and third blocks are assigned to two threads that will run concurrently, and depending on the number of threads available, the number of concurrent executions will increase. In a simple solution, as seen in Figure 9, if six threads are available, the rows would be split in 6 blocks, and each block of columns would be assigned to one of the threads. Therefore, when getting to the third hyperplane, the group (X3,Y1) would run concurrent with (X2,Y2) and (X1,Y3).

Another computational problem is the amount of memory required by the modified Levenshtein's algorithm. In a simple example, two texts of size

200,000 characters being compared would require a matrix of 200,000 by 200,000 integer values or 40,000,000,000 integer values or approximately 160 Gbytes of memory. Well beyond the typical physical memory available in current desktop computers.

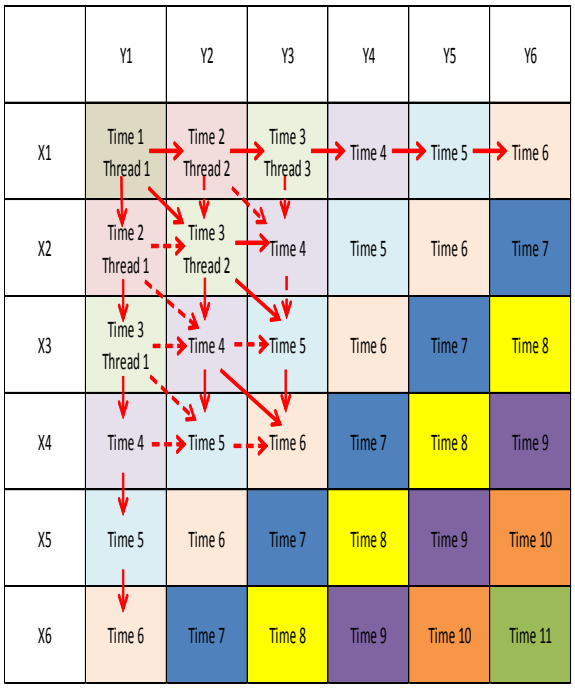


Figure 6 Parallel execution using threads

A simple solution was adopted by using dynamic allocation of a pool of memory, where just a few rows of the matrix are initially allocated, as seen in Figure 7, and the low index rows are reutilized as the wavefront advances through the array. In other words, when row X8 in Figure 7 needs to run, it reutilizes the memory initially allocated for row X1, since the results for X1 have already been propagated to row X2, and X1 is no longer needed. In the example mentioned earlier, involving 200,000 character documents, and assuming six threads running in parallel the initial memory allocation requirement becomes only 6.4 Mbytes. A larger number of threads will require more memory and synchronization while increasing the parallelism and reducing computational time.

5 EXPERIMENTS

To be able to properly test the proposed solution in this research, it was very important that the test data results could be verified, according to its known evolutionary sequence. Unfortunately, textual evolution has usually been recorded based on when the variant became public by being published or inscribed in some document, such as a letter to a friend or a dated draft kept in the author's files, and not work modification. Therefore, the most recently dated version must always be considered a version of one that was dated before it. Most of the dates used are those in which the literary documents were published or recorded in some publication or digital archive.

Considering that the main focus of this research is in the field of textual criticism, the sources chosen were of a literary nature and a realistic representation of works that would be studied by a literary scholar. These sources can be categorized in three different groups. The first group is comprised of documents, specifically poems, that were written and published in books and later digitized for Internet accessibility, obtained through the Elizabeth Barrett Browning (EBB) project at the University of North Dakota (UND), accessible online through UND's implementation of the Versioning Machine [10, 12]. These test cases range between having three to four versions beyond the original. The sizes of these poems are in a range of 508 to 2,699 characters each.

	Y1	Y2	Y3	Y4	Y5	Y6
X1	Time 1 Thread 1	Time 2 Thread 2	Time 3 Thread 3	Time 4 Thread 4	Time 5 Thread 5	Time 6 Thread 6
X2	Time 2 Thread 1	Time 3 Thread 2	Time 4	Time 5	Time 6	Time 7
X3	Time 3 Thread 1	Time 4	Time 5	Time 6	Time 7	Time 8
X4	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9
X5	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10
X6	Time 6	Time 7	Time 8	Time 9	Time 10	Time 11
X7	Time 7	Time 8	Time 9	Time 10	Time 11	Time 12
X8	Time 8	Time 9	Time 10	Time 11	Time 12	Time 13
X9	Time 9	Time 10	Time 11	Time 12	Time 13	Time 14

Figure 7 Memory allocation, row X8 reutilizes row X1

The second data set, represented as "WS" for William Shakespeare in Table 1, was obtained from the Internet Shakespeare Editions (ISE) website that is supported by the University of Victoria, Friends of ISE, and the Social Sciences and Humanities Research Council of Canada [2]. This dataset is made up of the quartos, folios and modern published versions of writings by William Shakespeare. This dataset consists of documents that range in size from 2,812 to 144,878 characters each. In particular, the data set labeled *Hamlet* was redefined in four different subsets to allow for an in-depth examination of the obstacles to establishing the evolutionary sequence of the text. The original *Hamlet* set contains 7 texts, while the remaining subsets contain only 3 to 5 files.

Lastly, a third test set is the poem "*Song of Myself*" found in the collection of poetry, *Leaves of Grass*, written by Walt Whitman and found in the Walt Whitman Archive housed by the University of Nebraska, Lincoln [11]. The reason of using this third case was to have a second set of data with more than five files. This dataset consists of documents that range in size from 70,239 to 73,473 characters each.
Below is a discussion of the computational time results associated with running the sequential and the multithreaded algorithms with the use of four threads in the parallel implementation. Table 1 shows the significant performance improvement obtained by the multithreaded configuration. The most significant gain, due to the size of the test case, was registered when running the William Shakespeare *Henry V* test case, which showed a speed up factor of 3.95 on the total execution time of the algorithm when compared to the sequential execution [1]. The smaller test cases showed minor improvements. In such cases, the overhead of activating the threads added to the synchronization delays between threads resulted in the smaller reduction of the execution time when compared to the sequential cases.

One of the main goals of this experiment was to verify that this technique can be used as a guide to determine the evolutionary sequence of versions of documents. In order to be able to determine the accuracy of the proposed method, a numerical scoring system had to be applied. Woon and Wong proposed the use of a new scoring system, restricted to graphs consisting of a single linear path and establishing windows of comparison along such path, which allowed one correct result to be counted multiple times according to the size of the window when a node preceded any of its actual successors in the path [16].

In this study, a window of size one was assumed, where an original is checked only against its immediate succeeding version, so a one point score is awarded to every edge that correctly matches the order of the actual text dates and zero points to those edges that fail the match. Table 2 shows the accuracy of the GRALT algorithm working with the MST and the SPE path options in the benchmark tests.

Table 1 Computational time for each test case.

Test Case	Sequential Processing Time (in seconds)	Multi-thread Processing Time	Multi-thread/Sequential Processing Speedup
EBB Child	0.898	0.515	1.74
EBB Bettine	0.566	0.343	1.65
EBB Sea	0.766	0.421	1.82
EBB Loved	1.092	0.390	2.80
EBB Clouds	1.282	0.609	2.11
EBB Dog	1.045	0.390	2.68
EBB Mitford	0.237	0.109	2.17
WW Leaves	1599.000	432.734	3.70
WS Hamlet	7224.080	2247.050	3.21
WS Henry IV Part I	3.354	0.905	3.71
WS Henry V	13.354	3.385	3.95
WS Richard II	4.321	1.201	3.60
WS Richard III	2.371	0.687	3.45
WS Romeo & Juliet	1.950	0.686	2.84
WS Troilus and Cressida	4.462	1.233	3.62

Table 2 Accuracy of GRALT with MST and SPE implementations

Test Case	Minimum Spanning Tree	SPE
EBB Child	100%	100%
EBB Bettine	66%	66%
EBB Sea	33%	33%
EBB Loved	100%	100%
EBB Clouds	100%	100%
EBB Dog	50%	50%
EBB Mitford	100%	100%
WW Leaves	100%	100%
WS Hamlet	66%	66%
WS Hamlet (exclude Quarto 1 files & Modern versions)	100%	100%
WS Hamlet (exclude Modern Versions)	66%	66%
WS Hamlet (exclude Quarto 1 & Modern Quarto 1)	75%	50%
WS Henry IV Part I	100%	66%
WS Henry V	66%	66%
WS Richard II	100%	100%
WS Richard III	100%	100%
WS Romeo & Juliet	100%	100%
WS Troilus and Cressida	100%	100%

6 CONCLUSION

In the analysis of literary works from the same author, it is important to know the evolutionary sequence, or versioning sequence, of the texts. In many situations, the actual sequence is unknown. The result of this study was the design and implementation of a system that can automatically make an estimation of the evolutionary sequence of digitized data that has not had its evolutionary derivation information recorded. The solution was based on the use of graph theory, in particular by embedding modified implementations of the Kruskal's Minimum Spanning Tree (MST) and Hamiltonian path (SPE for Single Path Evolution) algorithms to make the final determinations, which resulted in the design of the GRALT algorithm. The graph techniques were applied, resulting in a complete system design, which utilized a parallel wavefront implementation of a modified Levenshtein's algorithm to calculate text editing distances. The experiments showed a significant improvement on execution time when running a multithreaded implementation. There are still other approaches to be further investigated, including the improvement of the algorithm implementation computational time, which may be accomplished by increasing the number of threads and the number of processing elements or running on a distributed computer system.

7 REFERENCES

1. Hennesy, J.L., and Patterson, D.A., Computer Organization and Design The Hardware/Software Interface, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1994.

2. (2013, March). Internet Shakespeare Editions [Online]. Available: internetshakespeare.uvic.ca.

3. (2013, June). Juxta[Online]. Available:http:// www.juxtasoftware.org.

4. Lamport, L. "The Parallel Execution of DO Loops," in the *Communications of the ACM Special Interest Group on Programming Languages*, February 1974, pp. 82-93

5. Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

6. Mayfield, K., Grant, E., Alberts, C. "Determining the Evolutionary Sequence of Versions of a Text with the Minimum spanning Tree Algorithm," Proceedings: 25th International Conference on Computer Applications in Industry and Engineering, 2012.

7. Mayfield, K. Alberts, C. "GRALT: Graphical Analyzer of Literary Text Versions," in the North Dakota Quarterly, Vol. 79, No 1, 2014.

8. Neapolitan, R., Naimipour, K. "Foundations of Algorithms Using C++ Pseudocode," Jones and Bartlett, Inc., New York, 2004.

9. Shillingsburg, Peter L. *From Gutenberg to Google: Electronic Representations of Literary Texts*. Cambridge, UK: Cambridge University Press, 2006.

10. (2011, November 15). The University of North Dakota: Elizabeth Barrett Browning Project [Online]. Available: http://und.edu/instruct/ sdonaldson/index.html.

11. (December 2013). The Walt Whitman Archive [Online]. Available: http://www.whitman archive.org/.

12. (2013, January 15). Versioning Machine v.4.0 (2011)[Online]. Available: http://v-machine.org/index.php.

13. Vetter, L., and McDonald, J. "Witnessing Dickinson's Witnesses," in *Literary and Linguistic Computing*, Vol. 18, No 2, 2003, pp. 151-165.

14. Williams, W.P., Abbott, C.S., "An Introduction to Bibliographical and Textual Studies." The Modern Language Association of America, New York, 1985.

15. Wise, M. "YAP3: Improved Detection of Similarities in Computer Program and Other Texts," in the *Proceedings of the ACM Special Interest Group on Computer Science Education*, 1996, pp. 130-134.

16. Woon, W.L., Wong, K.D. "String Alignment for Automated Document Versioning," Knowledge Information Systems, 2009, pp. 293-309.

I/O performance evaluation of parallel scientific applications in a Cloud Environment

Pilar Gomez-Sanchez[1], Sandra Méndez[2], Dolores Rexachs[1] and Emilio Luque[1]
[1]Computer Architecture and Operating Systems Department, Universitat Autònoma de Barcelona (UAB), Bellaterra (Barcelona), Spain
[2]High Performance Systems Division, Leibniz Supercomputing Centre (LRZ), Garching (Munich), Germany

Abstract— *Performance evaluation of parallel application plays an important role on High Performance Computing (HPC), in cloud computing as well as in the classical systems. As the scientific community selects the cloud environment to access resources of HPC, performance evaluation brings new challenges such as defining methods to analyze the performance on a more configurable environment than traditional cluster computers. In this paper, we present a methodology to evaluate the I/O performance of parallel application over the Virtual Clusters in a cloud environment. We represent the I/O requirements of the parallel application in an I/O model and define a percentage required of the memory and storage to determine if the application is I/O intensive. If the application is I/O intensive, we define boundaries for the data transfer rate to identify if the application is limited by the I/O subsystem. We have evaluated on different Virtual Clusters configurations for the kernels of scientific applications S3DIO and BT-IO. Experimental results show that our methodology allows us to identify when an application is I/O intensive and to evaluate if it is limited by the I/O subsystem.*

Keywords: Parallel I/O System, Access Pattern, I/O Configuration, I/O Modeling, I/O phases

1. Introduction

The increasing interest in cloud computing by the High Performance Computing (HPC) community offers new challenges for the users of HPC. Efficient use of the Virtual Cluster on Cloud (VCC) is crucial because the user must pay fees based on use of resources and services. Due to the configurability and elasticity of cloud environments, the user can create and configure a VCC that meets the needs of their parallel application by avoiding the unnecessary resources use. Parallel applications that use parallel I/O need to be evaluated to know if the I/O subsystem is limiting the application performance. This allows the user to identify possible bottleneck in the I/O subsystem which provides guidelines to change the configuration of the I/O subsystem to reduce or avoid the effect of the bottlenecks.

In this paper, we present a methodology to evaluate the expected performance in a new VCC quickly taking into account the I/O subsystem configuration and the application behavior on cloud computing. The methodology has the following steps: 1) Application Parallel I/O Characterization. The I/O model, which characterizes the application behavior, is used to identify if the application performance is limited by the selected I/O system configuration. 2) Virtual Clusters I/O Characterization. 3) Performance Evaluation on the Virtual Clusters for the application I/O model. 4) Performance Analysis based on the relation of the I/O Time-Cost.

We show the results to apply the proposal methodology in different configurations VCC for the kernels of parallel application S3D-IO and BT-IO with different inputs and number of processes.

The cloud platform selected is Amazon EC2 because this is the leading Infrastructure-as-a-Service (IaaS) provider.The tool selected to create a Virtual Cluster on Amazon EC2 is StarCluster [1]. To facilitate the I/O configuration, we have implemented a tool for configuration and installation of the parallel filesystem PVFS2.

The rest of this article is organized as follows; in Section 2, we review the related work, Section 3 introduces our methodology. In Section 4, we present the experimental results. Finally, in Section 5, we present the conclusions and future work.

2. Related Work

The cloud platform introduces a new level system, virtualization. This level increases the complexity of computing, networking and I/O system. There are several studies that focus on performance evaluation. Noorshams et al. present in [2] an I/O performance modeling approach for virtualized storage systems based on queueing theory. Sivathanu et al. in [3] present a measurement study of I/O performance. They focused on the influence of careful disk provisioning and placement on I/O performance workload interference and the implications of virtualization on different types of workloads. In [4], [5], [6], the storage subsystem on the Amazon EC2 platform is evaluated using different instances type to analyze if cloud platform is convenient for scientific applications with high I/O performance requirements.

Unlike the previous authors, we present a methodology that defines the I/O requirements of the parallel application

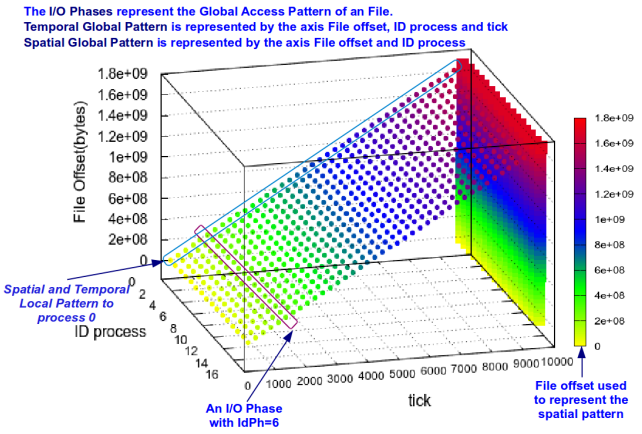


Fig. 1: The phase concept used by the I/O model to represent the global access pattern of the parallel scientific applications

based on an I/O model and in the I/O characteristics of the VCC. This allows the user to determine if his I/O intensive application is limited by the I/O subsystem of the VCC.

3. Methodology of Analysis

In this section, we present a methodology for the I/O performance evaluation of the parallel scientific applications on a cloud environment. Next, we explain the steps of our methodology.

3.1 Application Parallel I/O Characterization

The I/O characteristics of the parallel application is expressed by an I/O model which is extracted with the tool PAS2P-IO [7]. The application model is defined by three characteristics: meta-data, spatial global pattern and temporal global pattern. The concept of I/O phase is defined to represent a global pattern, both temporal and spatial, for each file of the parallel application. Figure 1 shows the I/O phase concept for the model.

An I/O phase is represented by:

- direct values extracted from the application,
  - $IdPh$, identifier of a phase.
  - np, number of processes that composes the phase.
  - The I/O pattern of the phase that is composed of the request size (rs), the type of operation (w/r) and the number of I/O operations ($\#iop$).
  - Number of repetitions rep of the phase $IdPh$.
- indirect values,
  - Data Volumen: $weight_{(IdPh)}$ for the phase $IdPh$. This represents the data transferred during the phase, it is expressed in Bytes and it is calculated by expression 1.

$$weight_{(IdPh)} = np \times rs \times rep \qquad (1)$$

A file of the application is represented by:

- A set of identifiers of phases.
- Memory required per process: this represents the memory required (expression 2) for the I/O operations per process. This includes the I/O operations of the phases (Num_Phs) in which the process takes part.

$$MRxP = \sum_{i=1}^{Num\_Phs} rs_i \times \#iop_i \times rep_i \qquad (2)$$

- Access Mode: Strided, Sequential and Random.
- Access Type: Unique (a file per process) or Shared (a shared File between the processes).
- Metadata: this is associated to positioning, synchronism, and coordination of the data access operations defined for MPI-IO.
- File Size: for the I/O files and output files, the size is calculated by expression 3 where phs_write is the number of phases with write operations at different offsets of the file, and for input files by expression 4 where phs_read is the number of phases of read operations.

$$IOFSize = \sum_{i=1}^{phs\_write} weight_i \qquad (3)$$

$$INFSize = \sum_{i=1}^{phs\_read} weight_i \qquad (4)$$

Finally, the number of files (nf) and the storage capacity required by the application are added to the I/O characteristics. The storage capacity (ST_{app}) is calculated by expression 5, where nf_IOF is the number of I/O files, nf_INF is the number of input files.

$$ST_{app} = \sum_{i=1}^{nf\_IOF} IOFSize_i + \sum_{i=1}^{nf\_INF} INFSize_i \qquad (5)$$

3.2 Characterization of the Virtual Clusters

For the cluster selection of components for the VCC, we have applied the methodology presented in [8]. We apply IOzone[9] benchmark in a node of the VCC to obtain the average transfer rate at local filesystem level. IOzone is a filesystem benchmark tool that generates and measures a variety of file operations. It is used in automatic mode with a file size equal to double RAM size, minimum request size (rs) equal to 4 KB and a maximum request size equal to 1 GB or 2 GB, depending on the network performance. We have selected only write/rewrite and read/reread operations. From this, we obtain the average transfer rate (`Avg_ST_bw`) for request sizes between minimum and maximum. IOzone supplies us with the dynamic characteristics for the nodes of Clusters. It allows us to evaluate the instance variability that will depend on date and hour, and thus what we can expect for a specific application.

From the IOzone measures, we calculate the storage bandwidth (`ST_BW`) for the global file system of the VCC.

This is calculated by expression 6, where Num_DFs is the number of data servers of the global filesystem.

$$ST\_BW = Avg\_ST\_bw \times Num\_DFs \quad (6)$$

3.3 Performance Evaluation on the Virtual Clusters for the application I/O model

We represent, in Figure 2, the I/O requirements of the parallel application in an I/O model and we define a percentage required of the memory and storage to determine if the application is I/O intensive. Below, the steps are described to evaluate the performance on each VCC.

- Percentage calculation of I/O Requirements: The percentage of the memory and storage capacity required is calculated to determine the application I/O requirements on a specific system. We consider that a parallel application with less than 15% of the memory and storage requirements is not I/O intensive in I/O system of the VCC. Applications with more than 20% are considered I/O intensive. The equation 7 is applied to calculate the percentage of memory required for the I/O operations per compute node.

$$\%Mem.Req.CN = \frac{(NPxCN \times MRxP) \times 100}{RAM(CN)} \quad (7)$$

  Where $NPxCN$ is the number of processes (np) per compute node (CN), $MRxP$ is the memory required for the I/O operations per process, and the $RAM(CN)$ is the RAM size of the compute node.
  The percentage of the storage capacity required by the application is calculated by expression 8.

$$\%ST.Req = \frac{ST_{app} \times 100}{ParallelStorageCapacity} \quad (8)$$

  Where $ParallelStorageCapacity$ is the capacity of global filesystem configured in the VCC.
- I/O Evaluation: To reduce the time consumed in this step and the associated cost, we use an I/O Time prediction. For that we obtain the transfer rate ($BW_{(CH)}$) expressed in MB/sec, and I/O time considering the application behavior obtained in the model. To do this, we can use 1) a synthetic program that represents the I/O model or 2) an benchmark that can be configured for the I/O model parameters. In this paper, we use the second option. We apply IOR[10] benchmark to execute the I/O model of a parallel application on a specific I/O system.
  We have defined the input parameters IOR by representing the I/O model with the following values:
  - Access Mode (Strided, Sequential and Random).
    - Sequential: for $np = np_{(IdPh)}$ processes, $s = 1$, $b = weight_{(IdPh)}$ and $t = rs_{(IdPh)}$
    - Strided: for $np = np_{(IdPh)}$ processes, $s = rep$, $b = rs_{(IdPh)}$ and $t = rs_{(IdPh)}$.
  - Access Type: Unique $-F$ (1-file-per-process) or Shared $default$ (shared-file between the processes).
  - I/O library: $mpiio$, $-c$ for the MPI with Collective I/O, and $posix$. Collective I/O is performed at user level through the ROMIO.
  - Storage Capacity required: nf, number of files. The storage capacity required for the parallel application is calculated by $\sum_{i=1}^{nf} s_i \times b_i \times np_i$.

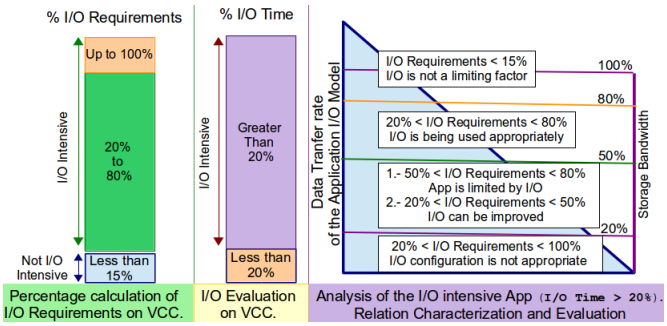


Fig. 2: Performance Evaluation on the Virtual Clusters for the application I/O model

- Analysis of the I/O intensive application: in Figure 2, the relation between the I/O requirements can be observed, as well as I/O time and the storage bandwidth to evaluate when an application can be limited by the I/O system. We analyze the I/O impact on the parallel application performance by considering the storage bandwidth defined in the VCC characterization. To do this, we define the following rules for the transfer rate (`BW`) obtained, storage bandwidth (`ST_BW`) and the I/O requirements (`IO_Req`):
  - If (`BW >80% of ST_BW`) then the I/O is no a limiting factor for the application in that system.
  - If (`50% of ST_BW< BW <80% of ST_BW`) and (`20%< %IO_Req <80%`) then I/O is being used appropriately.
  - If (`20% of ST_BW< BW <50% of ST_BW`) and (`50%< %IO_Req <80%`) then the application is limited by I/O.
  - If (`20% of ST_BW< BW <50% of ST_BW`) and (`20%< %IO_Req <50%`) then I/O performance can be improved.
  - If (`BW< 20% of ST_BW`) and (`20%< %IO_Req <100%`) then selected configuration not is appropriate for application requirements.

3.4 Performance Analysis based on the relation of the I/O Time-Cost.

Measures obtained with IOR for the application I/O model are used to calculate the utilization cost of I/O system. The total cost for a specific VCC is composed of a variable cost ($cost\_var$) and a fixed cost ($cost\_fix$). This is computed

by (9).

$$cost\_tot = cost\_fix + cost\_var \quad (9)$$

The metric selected for IOR is the transfer rate ($BW_{(CH)}$), expressed in MB/sec. The variable cost estimated for the application I/O model is calculated by (10). This variable is proportional to utilization time of I/O, and hence to the quantity of transfered data. The billing is per utilized hours.

$$cost\_var = \sum_{i=1}^{phases} cost(phase[i]) \times num\_inst \quad (10)$$

Where num_inst is the number of instances used for the execution, the $cost(phase[i])$ is calculated by (11), $cost_{inst}$ is the cost per hour for the instance type $(inst)$ and EBS which represents the persistent storage per month. This last parameter is optional, but if you can't execute the application with temporal storage you can add the persistent to increases the storage capacity or simply to have the storage persistent.

$$cost(phase[i]) = costphase[i] + [EBS] \quad (11)$$

$$costphase[i] = (\frac{weight_{(phase[i])}}{BW_{(CH)}(phase[i])}/3600) \times cost_{inst} \quad (12)$$

The time-cost relation of the different VCCs is calculated by the equation 13.

$$perf\_cost_{ci} = \frac{Time_{ci}}{cost_{ci}} \quad (13)$$

Where $Time$ is expressed in sec, ci represent one VCC, and $i \in \{1..NumVCC\}$ where $NumVCC$ is the total number of distinct configurations of VCCs to analyze.

To compare the Cluster ck and Cluster cj, we can say ck has a more efficient performance-cost relation than cj, if $perf\_cost_{ck}$ is higher than $perf\_cost_{cj}$.

4. Experimental Results

In this section, we present the performance evaluation and the cost analysis for two well-known scientific applications I/O kernels, NAS BTIO[11] and S3D-IO[12], that present different I/O access patterns. We work with the S3D-IOv1.1, blocking API's and without restart.

BT-IO and S3DIO have been traced using PAS2P-IO to extract their I/O models.This process was carried out in physical computer clusters Finisterrae of the Centre of Supercomputing of Galicia(CESGA)[13] and Supernova of the Wroclaw Centre for Networking and Supercomputing[14].

We have selected two instance types from Amazon EC2 [15], taking into account the I/O requirements of application and the instance price. The Amazon instances selected are shown in Table 1. The C3 instances are compute-optimized instances, featuring the highest performing processors and the lowest price/compute performance in EC2. The M3 instance types provides a balance of compute, memory, and network resources. Using the instances of Table 1, we have created two VCCs and Table 2 shows the components of the VCCs.

Table 1: Characteristics of the Amazon's Instances Selected

Instances	Processor Intel Xeon	RAM (GB)	Storage (GB)	Network	$xHora
c3.xlarge	4 Cores; E5-2680 v2 2.8 GHz	7.5	2x40 SSD	Moderate	0.210
m3.xlarge	4 Cores; E5-2670 v2 2.5 GHz	15	2x40 SSD	High	0.280

Table 2: Descriptive Characteristics of the Virtual Clusters on cloud (VCC) configured equal for the experiments.

I/O components	VCC 1	VCC 2
Instance Type	c3.xlarge	m3.xlarge
Number of Instances	10	10
Storage Type Temporal	Ephemeral	Ephemeral
Storage Type Persistent	EBS	EBS
Device Type Temporal	SSD	SSD
Device Type Persistent	HDD	HDD
Capacity of Persistent Storage	16GB	16GB
File system Local	ext3	ext3
File system Global	PVFS2	PVFS2
Parallel Storage Capacity	320GB	320GB
Number of data servers	8	8
Number of Metadata Server	1	1
Stripe Size	64KB	64KB
MPI library	mpich2-1.5	mpich2-1.5
I/O library	pnetcdf 1.4.1	pnetcdf 1.4.1

4.1 BT-IO Charaterization

The BTIO benchmark performs large collective MPI-IO writes and reads of a nested strided datatype, and it represents a significant workload test to evaluate the performance that a system can provide for non-contiguous workloads.

We have obtained the following meta-data of NAS BT-IO in the FULL subtype with our tool PAS2P-IO: Explicit offset, Blocking I/O operations, Collective operations, Strided access mode, Shared access type and a shared File accessed by all the MPI processes.

Table 3 presents the I/O phases for the I/O model using 25 and 36 processes for the classes C and D. We also present the storage capacity required by BT-IO for the different classes and memory required for each process.

4.2 S3D-IO Charaterization

S3D-IO uses parallel NetCDF for checkpointing. A checkpoint is performed at regular intervals, and its data consist of 8-byte three-dimensional arrays. We have obtained the following metadata for the S3D-IO with our tool PAS2P-IO: Collective write, Individual file pointer, Blocking I/O operations, Strided access mode, Shared access type and five Shared files accessed by all MPI processes.

Table 3: I/O phases for the BT-IO model using 25 and 36 processes for the classes C and D

Class	IdPh	np	Operation	rs (MB)	rep	weight(IdPh)MB (rep× np × rs)	weight(app)	STapp(GB)	MRxP (MB)	Files
C	1 to 40	25	Write_at_all	6.49	1	1 × 25 × 6.49	12.9GB	6.4	518	1
	41	25	Read_at_all	6.49	40	40 × 25 × 6.49				1
C	1 to 40	36	Write_at_all	4.50	1	1 × 36 × 4.50	12.9GB	6.4	360	1
	41	36	Read_at_all	4.50	40	40 × 36 × 4.50				1
D	1 to 50	36	Write_at_all	72	1	1 × 36 × 72	259.2GB	129.6	7,196	1
	51	36	Read_at_all	72	50	50 × 36 × 72				1

Table 4 presents the I/O phases for the I/O model using 16 and 32 processes for the workloads 200x3 and 400x3. In addition, we show the storage capacity required by the application and memory required for each process.

4.3 Characterization of the Virtual Clusters

Considering that we have two clusters with the parallel file system PVFS2, because the two benchmarks selected use shared files for all MPI processes. We execute IOzone to evaluate the storage bandwidth for transfer rate provided by the ephemeral disk for one node. We only evaluate the ephemerals used for the PVFS2 parallel file system. The PVFS2 was configured with a metadata server (MDS), 8 datafiles (DF) and a 64KB stripe size. The number of compute node is 9 for the BT-IO and 8 for S3D-IO. In both cases, nodes 1 to 8 perform I/O and compute. In total 10 instances are used on each VCC, where the master node is used as MDS server, with the objective being to have different performance behavior and cost, some intensive and others not. The nodes are sharing compute and I/O.

We evaluate transfer rate of write and read operations for request sizes from 64KB to 1GB. Table 5 presents the storage bandwidth, the operation average and the deviation standard for the two VCCs.

4.4 BT-IO Performance Evaluation

Table 6 shows the I/O requirement for BT-IO on VCC 1 and VCC 2. We can observe that the Class C on two VCCs requires less than 10% of memory and storage capacity. The I/O performance expected could overcome the storage bandwidth defined in Table 5. Class D, in Table 6, presents memory requirement higher than 80%, and a storage capacity greater than 20%, which is the umbral to consider this application as a candidate to be I/O intensive.

Figure 3 shows the transfer rate for IOR and write operation, configured for the BT-IO, on VCC 1 (c3.xlarge) and VCC 2 (m3.xlarge). The values correspond to the average and standard deviation to consider the variability of the cloud environments in the evaluation. The tests run 3 times consecutively. It can be observed that Class C overcomes the write operation storage bandwidth of VCC 1 and is near to the write operation storage bandwidth of the VCC 2.

Class D, Figure 3, as was expected, is I/O intensive in two VCCs. The variability is higher for the VCC 1 and is using around 75% of storage bandwidth, which is indicating

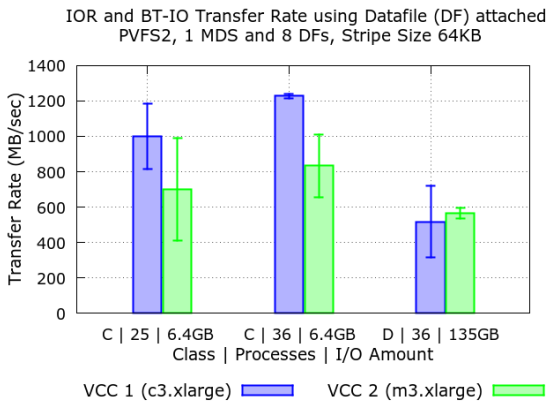


Fig. 3: IOR Transfer rate tuned for the I/O model parameters of the BT-IO on VCC1 (c3.xlarge) and VCC2 (m3.xlarge).

an appropriate use of the I/O system. For the VCC 2, the transfer rate reported is near to 60%, a value sufficient to be considered not I/O limited.

Although, the BT-IO presents both write phases as read, the application performance is more affected for the write operations due to the write phases being 40 times more than read phases for Class C and 50 times more for Class D.

4.5 S3D-IO Performance Evaluation

Table 7 shows the I/O requirement for S3D-IO on VCC 1 and VCC 2. In this case, the I/O requirements are focused on the memory required because storage required, in both VCCs, is less than 20%.

In Table 7, it can be observed that S3DIO with a workload 200x3 is not I/O intensive due to that its memory required and storage represent less than 20%.

To analyze the memory required impact, we evaluate the I/O model of S3DIO using IOR. Figure 4 shows the transfer rate on VCC 1 (c3.xlarge) and VCC 2 (m3.xlarge).

The S3DIO memory requirements for the 400x3 workload (I/O Amount = 39 GB) represent 64% of the RAM per compute node on VCC 1 and 32% on VCC 2. In VCC 1, the value is greater than 50%, which could avoid to take advantage of the performance capacity of the I/O system. In Figure 4, it can be observed for the I/O amount of 39 GB (workload 400x3) that VCC 2 has 20% more performance than VCC 1. This shows that the application obtained more performance in the VCC 2, where the memory required is 32% which is a 50% less than on VCC 1.

Table 4: I/O phases for the S3D-IO model using 16 and 32 processes for the workloads 200x3 and 400x3.

Workload	IdPh	np	Operation	rs (MB)	rep	weight(IdPh)MB (rep × np × rs)	weight(app)	STapp(GB)	MRxP (MB)	Files
200x3	1 to 5	16	write_all	61	1	976	4.8GB	4.8GB	305	1 per IdPh
	1 to 5	32	write_all	30.5	1	976	4.8GB	4.8GB	153	1 per IdPh
400x3	1 to 5	16	write_all	488	1	7808	39GB	39GB	2,442	1 per IdPh
	1 to 5	32	write_all	244	1	7808	39GB	39GB	1,221	1 per IdPh

Table 5: Storage bandwidth calculated from IOzone measures for Virtual Clusters on cloud (VCC) with different sizes.

Cluster Virtual	Avg.Write (MB/s)	Std Write (MB/s)	Avg.Read (MB/s)	Std Read (MB/s)
VCC 1	774	6	2,253	12
VCC 2	945	14	2,167	20

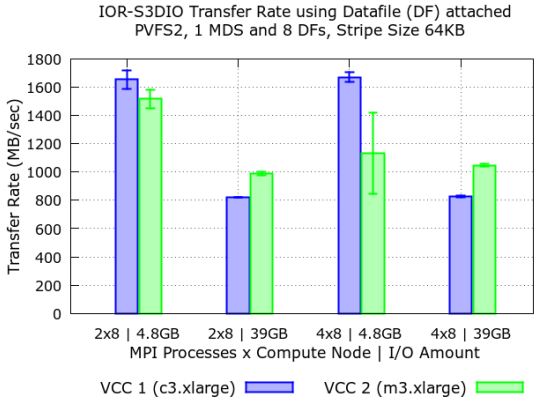


Fig. 4: IOR Transfer rate tuned for the I/O model parameters of the S3DIO on VCC1 (c3.xlarge) and VCC2 (m3.xlarge).

Finally, we compare the execution time of the BT-IO and S3D-IO application and their I/O models. In Figure 5, we show execution time for the IOR and BTIO. Figure 6 depicts time execution for S3D-IO and its version IOR. We can observe that the time for the I/O models are less than the execution time of real application both BTIO as S3DIO. As we pay per used instance and per used hour, the cost will be minor. In this way, we provide a method to evaluate performance for the I/O pattern of the application in less time and with less cost.

5. Conclusions

The performance evaluation is an important issue for the analysis of the HPC application, in cloud computing as well as in the classical HPC systems.

Parallel applications, which use I/O, need to have some additional information to determine when the application will be limited by the I/O system of VCCs. We have presented a methodology that represents the I/O characteristics of the application and the VCCs to determine when an application is I/O intensive and to evaluate if this is limited by the I/O subsystem.

Experimental results show cases for different patterns that reflect how the I/O performance can be affected, both by the percentage of storage required as by the memory requirement. Due to the elasticity of the cloud environment, the user is provided with information that can help to define the distribution MPI processes on the compute nodes considering the memory used per the I/O processes. Furthermore, for the I/O intensive applications, the user can reduce the percentage of storage required by adding more data servers to reduce the impact of the I/O utilized.

As future work, we are analyzing the implementation of the synthetic program that replicates complex I/O patterns that IOR cannot represent appropriately. Furthermore, we will continue analyzing the impact of the different components for the VCC configuration on cost and performance. Moreover, we will work with different file systems and other cloud platforms to evaluate the applicability of our methodology.

Acknowledgment

This research has been supported by the MINECO (MICINN) Spain under contract TIN2011-24384. The research position of the PhD student P. Gomez has been funded by a research collaboration agreement, with the "Fundación Escuelas Universitarias Gimbernat".

Appreciation to The Centre of Supercomputing of Galicia (CESGA) under the Science and Technology Infrastructures program (in spanish ICTS) and The Wroclaw Centre for Networking and Supercomputing (WCSS), Poland.

References

[1] StarCluster. (2014) An Open Source Cluster-Computing Toolkit for Amazon's Elastic Compute Cloud (EC2). [Online]. Available: http://star.mit.edu/cluster/

[2] Q. Noorshams, S. Kounev, and R. Reussner, "Experimental Evaluation of the Performance-Influencing Factors of Virtualized Storage Systems," in *Computer Performance Engineering. 9th European Workshop, EPEW 2012, Munich, Germany, July 30, 2012, and 28th UK Workshop, UKPEW 2012, Edinburgh, UK, July 2, 2012, Revised Selected Papers*, ser. Lecture Notes in Computer Science, M. Tribastone and S. Gilmore, Eds. Springer Berlin Heidelberg, 2013, vol. 7587, pp. 63–79.

[3] S. Sivathanu, L. Liu, M. Yiduo, and X. Pu, "Storage management in virtualized cloud environment," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 204–211.

[4] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2," *J. Grid Comput.*, vol. 10, no. 1, pp. 5–21, Mar. 2012.

[5] M. Liu, J. Zhai, Y. Zhai, X. Ma, and W. Chen, "One Optimized I/O Configuration Per HPC Application: Leveraging the Configurability of Cloud," in *Proceedings of the Second Asia-Pacific Workshop on Systems*, ser. APSys '11. New York, NY, USA: ACM, 2011, pp. 15:1–15:5.

Table 6: I/O requirement for BT-IO on VCC 1 (c3.xlarge) and VCC 2 (m3.xlarge). Eight nodes perform compute and I/O. MetaData Server (MDS) and Number of DataFiles(DF)

Class	Number of Processes	Compute Nodes (CN)	Processes x CN	Mem.Req. x CN (MB)	%Mem.Req. x CN (VCC 1)	%Mem.Req. x CN (VCC 2)	MDS+DF	Filesystem Capacity(GB)	%Storage Used
C	25	9	3	721	9%	5%	1+8	320	2%
C	36	9	4	721	9%	5%	1+8	320	2%
D	36	9	4	14394	187%	94%	1+8	320	40%

Table 7: I/O requirement for S3DIO on VCC 1 (c3.xlarge) and VCC 2 (m3.xlarge). Eight nodes perform compute and I/O. MetaData Server (MDS) and Number DataFiles(DF)

Class	Number of Processes	Compute Nodes (CN)	Processes x CN	Mem.Req. x CN (MB)	%Mem.Req. x CN (VCC 1)	%Mem.Req. x CN (VCC 2)	MDS+DF	Filesystem Capacity(GB)	%Storage Used
200x3	16	8	2	610	8%	4%	1+8	320	1%
400x3	16	8	2	4883	64%	32%	1+8	320	12%
200x3	32	8	4	610	8%	4%	1+8	320	1%
400x3	32	8	4	4883	64%	32%	1+8	320	12%

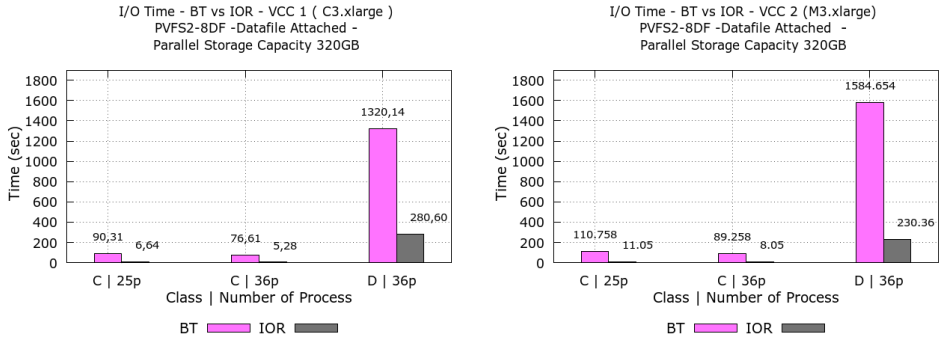


Fig. 5: Time Comparison between BT-IO and its IOR version on VCC 1 and 2

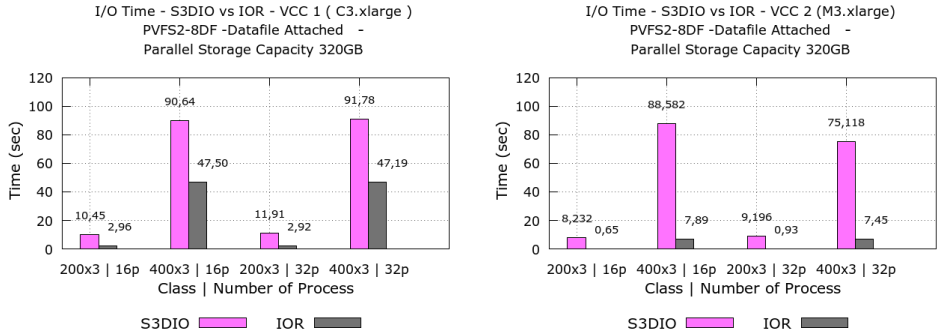


Fig. 6: Time Comparison between S3D-IO and its IOR version on VCC 1 and 2

[6] R. Expósito, G. Taboada, S. Ramos, J. González-Domínguez, J. Touriño, and R. Doallo, "Analysis of I/O Performance on an Amazon EC2 Cluster Compute and High I/O Platform," *Journal of Grid Computing*, vol. 11, no. 4, pp. 613–631.

[7] S. Méndez, J. Panadero, A. Wong, D. Rexachs, and E. Luque, "A New approach for Analyzing I/O in Parallel Scientific Applications," in *CACIC 12, Congreso Argentino de Ciencias de la Computación*, 2012, pp. 337–346.

[8] P. Gomez-Sanchez, S. Méndez, D. Rexachs, and E. Luque, "Hopes and facts in evaluating the performance of HPC-I/O on a cloud environment," *Journal of Computer Science & Technology*, vol. 15, no. 1, pp. 23–29, 2015.

[9] W. D. Norcott. (2006) IOzone Filesystem Benchmark. [Online]. Available: http://www.iozone.org/

[10] W. Loewe, T. McLarty, and C. Morrone. (2012) IOR Benchmark. [Online]. Available: https://github.com/chaos/ior/blob/master/doc/USER_GUIDE

[11] P. Wong and R. F. V. D. Wijngaart, "Nas parallel benchmarks i/o version 2.4," Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division, Tech. Rep., 2003.

[12] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo, "Terascale direct numerical simulations of turbulent combustion using S3D," *Computational Science & Discovery*, vol. 2, no. 1, p. 015001, 2009.

[13] CESGA. (2014) Finisterrae of the centre of supercomputing of galicia (CESGA). [Online]. Available: https://www.cesga.es

[14] WCSS. (2014) Supernova of the Wroclaw Centre for Networking and Supercomputing (WCSS). [Online]. Available: https://www.wcss.pl

[15] AWS-EC2. (2014) Amazon Elastic Compute Cloud, Instance Types. [Online]. Available: http://docs.aws.amazon.com/AWSEC2-/latest/UserGuide/Instances.html

Analysis of Fault Injection Approaches in Apache Hadoop

P. P. Barcelos, A. S. Charão, R. Boufleuer, J. C. Lima
Dept of Languages and Computing Systems, Federal University of Santa Maria, Santa Maria, RS, Brazil
{pitthan, andrea, rboufleuer, caio}@inf.ufsm.br

Abstract – *The Hadoop platform has been claimed as a solution for distributed processing of big data. This platform provides some fault tolerance mechanisms and there are a lot of studies that aim to improve them. Fault tolerance is a key feature of Hadoop, and it is essential to test the Hadoop´s dependability. In this context, validation mechanisms, which use fault injection techniques implemented through instrumentation code, arise as a need. This paper analyses some fault injection alternatives available in Hadoop platform. The results fit out as support for future work involving fault tolerance on this platform.*

Keywords: Fault Tolerance, Validation, Fault Injection, Big Data, Hadoop, MapReduce.

1 Introduction

Apache Hadoop [6] is an open-source distributed computing platform, developed in Java and targeted to the processing of large volumes of data. Hadoop comprises a distributed file system (HDFS - Hadoop Distributed File System), which can store data on a large number of servers and implements the MapReduce programming model [9], which easily enforces parallel operations on a set of distributed data.

This platform aims to employ computer clusters in a realiable, scalable and fault tolerant way. Both HDFS and MapReduce are designed to support and handle failures in the cluster nodes of the execution environment. Even though, Hadoop has been target of research papers whose goal is to improve its fault tolerance characteristics [5, 13].

In fault tolerant systems, it is necessary to ensure the system's ability to provide the specified service. Validation provides this guarantee as a goal [2]. To test a system we must submit it to certain states to ensure that some execution ways are achieved. Besides, the system may be submitted to a behavior that under normal conditions would not occur. These are the main goals to use validation techniques [8].

Fault injection is a widely used validation technique. It consists on the introduction of a failure in the system in a controlled way to observe its behavior [12]. This technique accelerates the occurrence of failures in a system. Thus, rather than waiting for the spontaneous occurrence of failures, we can intentionally introduce them by controlling the type, location and duration of the failures [10].

In this work, we analyse the features and response provided by Hadoop to fault injection. This is an exploratory research, based on literature review and practical experimentation in order to deeply know the possibilities of fault injection in the platform. The remainder of this paper is organized as follows: section 2 introduces Apache Hadoop, describing its architecture, its processes and the fault tolerance mechanisms implemented in the platform. The section 3 describes the two fault injection approaches explored in this work for Hadoop platform. Section 4 shows the experimentation realized with the fault injection tools described in Section 3. Finally, Section 5 presents some conclusions and final considerations about the work.

2 Apache Hadoop

The Apache Hadoop Project achievement is due largely to MapReduce Programming Model [9]. This programming model was originally presented by Google and has as main goals the processing and analysis of big data distributed over large scale computing system. Besides Google, a growing number of companies uses MapReduce to analyse the data generate by several applications, such as social networks, data mining, machine learning, log processing, and business intelligence [6].

Programs written in this functional style are automatically distributed and are able to execute on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system [9].

The MapReduce model abstracts details such as parallelization, fault tolerance, data distribution and load balancing of the application. The model allows to easily express computations which must be performed in parallel over a large volume of data. The input data are splitted into different tuples (key, value), which are submitted to a mapping function (Map) that generates an intermediate set of pairs (key, value). The intermediate pairs are then submitted to a reduction function (Reduce), responsible for processing all the values of a same key, creating a new final set of pairs (key, value) [9].

The development of the Apache Hadoop platform is inspired by the original publication of the Google MapReduce technology. The platform architecture is based on two main components: the Hadoop Distributed File System (HDFS) [3], which provides a distributed storage layer, and the Hadoop MapReduce [4], which provides a distributed execution layer. These components are described in the next paragraphs.

2.1 Hadoop Architecture

Figure 1, adapted from [6] and [11], shows how the processes of the Hadoop architecture are interrelated. Initially notice a separation of processes between master and slave nodes. The master node contains the NameNode, the JobTracker and the SecondaryNameNode. Each slave node includes a TaskTracker and a DataNode linked respectively to JobTracker and NameNode of the master node.

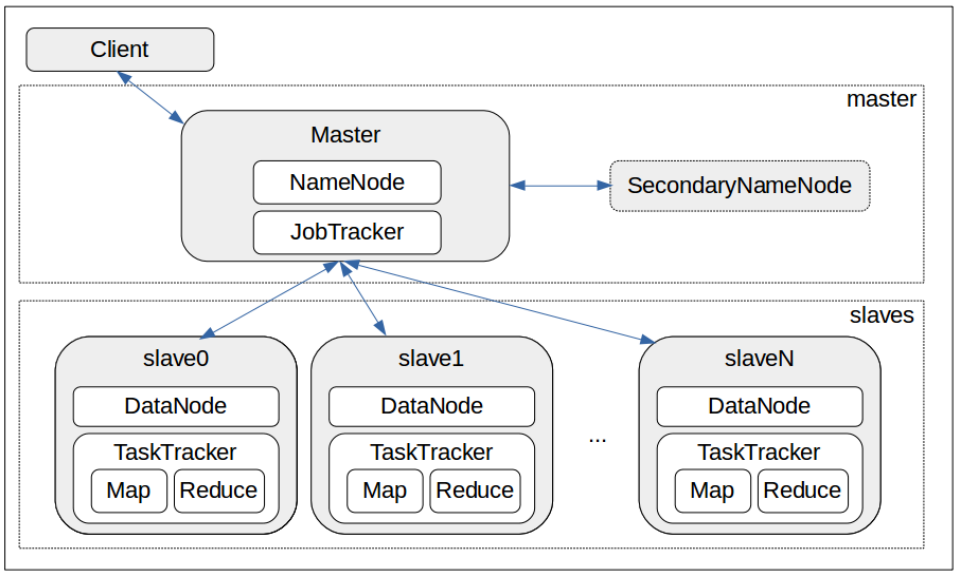


Figure 1: The Hadoop Processes (adapted from [6] and [11])

A client of an application connects to the master node and calls for execution. At this point, the JobTracker creates an execution plan and determines what, when and how often the slaves will process the application data. Meanwhile, the NameNode, based on parameters already defined, is responsible of storing and managing information from the files being processed. On the slave side, the TaskTracker executes the tasks assigned to it, which can be map or reduce tasks, and the DataNode stores one or more file blocks. During the execution, the slave node also needs to communicate with the master node, sending information about its local situation.

Alongside all this running, the SecondaryNameNode records checkpoints of NameNode in log files, useful in case of NameNode faults. These components are detailed in the next section.

2.2 Hadoop Processes Description

As we said in previous section, a Hadoop cluster follows a master-slave architecture where each node runs some processes that execute HDFS or MapReduce features. These processes are *NameNode*, *DataNode*, *JobTracker* and *TaskTracker*. *NameNode* and *JobTracker* run on a master node, while *DataNode* and *JobTracker* processes mainly run on slave nodes.

An HDFS cluster consists of a single *NameNode*, a master server that manages the file system namespace and regulates access to files by clients [3]. The *NameNode* in Hadoop is the node where Hadoop stores all the location information of the files in HDFS. In other words, it holds the metadata for HDFS. Whenever a file is placed in the cluster, a corresponding entry of its location is maintained by the *NameNode*.

In addition, there are a number of *DataNodes*, usually one per node in the cluster, which manage storage attached to the nodes where they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of *DataNodes*. The *NameNode* executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to *DataNodes*. The *DataNodes* are responsible for serving read and write requests from the file system's clients. The *DataNodes* also perform block creation, deletion and replication upon instruction from the *NameNode* [3].

The *JobTracker* is responsible for taking in requests from a client and assigning *TaskTrackers* with tasks to be performed. The *JobTracker* tries to assign tasks to the *TaskTracker* on the *DataNode* where the data is locally present. If that is not possible, it will at least try to assign tasks to *TaskTrackers* within the same rack. If for some reason the node fails, the *JobTracker* assigns the task to another *TaskTracker* where the replica of the data exists since the data blocks are replicated across the *DataNodes*. This ensures that the job does not fail even if a node fails within the cluster.

The *TaskTracker* is a daemon that accepts tasks (Map, Reduce and Shuffle tasks) from the *JobTracker*. The *TaskTracker* keeps sending a heart beat message to the *JobTracker* to notify that it is alive. *TaskTracker* starts and monitors the Map and Reduce tasks and sends progress or status information back to the *JobTracker*.

2.3 Fault Tolerance in Hadoop

Hadoop can achieve fault tolerance through the restart of the *TaskTracker* and *JobTracker* tasks. *TaskTrackers* are always communicating with the main node system, the *JobTracker*. In case of communication failure, the *JobTracker* assumes that the *TaskTracker* failed. As *JobTracker* knows which Map/Reduce tasks were assigned to each *TaskTracker* process, if the job is still in the Map/Reduce phase, then other *TaskTrackers* will be assigned to re-execute all Map/Reduce tasks previously performed by the *TaskTracker* failed.

Once completed, Reduce tasks are written in HDFS. Thus, if a *TaskTracker* finishes some of these Reduce tasks, only the tasks not yet completed need to be executed. With respect to Map tasks, it occurs in a different way. Even if a node has completed a certain amount of Map tasks, the nodes that execute Reduce may not have copied the Map tasks outputs to their entries. In this case, the failure of a Map process makes unavailable its output. So, completed Map tasks must be re-executed in order to make available their results to other reducing processes.

Another way to achieve fault tolerance is through HDFS, which implements redundancy through file replication over several *DataNodes*. The *DataNodes* store and retrieve data blocks. The mapping of file to blocks is kept by *NameNode*, which also controls the block mapping in *DataNodes*. The HDFS standard replication factor is three, which means that the data blocks are replicated three times. Moreover, every three seconds, the existing *DataNode* processes send "heartbeat" messages to *NameNode*. Faulty nodes are detected after the absence of heartbeat messages during some time. In this case, data blocks in the faulty node are identified, recovered from an operational node and then replicated to another cluster node. The corruption of data blocks is detected through checksums. At this rate, the failures are reported to *NameNode*, which triggers the replication of these data blocks. Considered a single point of failure, the *NameNode* has a Secondary *NameNode*, whose goal is to log the *NameNode* operations.

3 Hadoop Fault Injection

This work deals with fault injection in Hadoop through code instrumentation techniques. One of the main concerns of code instrumentation for fault injection approaches is to avoid significant changes in the code of the target application.

Code instrumentation consists of inserting auxiliary code in the target application, allowing to observe the behavior of the application and to get information or measures concerning its implementation.

In latest stable Hadoop versions, we found two different means to inject failures on Hadoop, using code instrumentation. The first resides on the *MRReliabilityTest*, a progam to test the reliability of the MapReduce. The second is the Fault Injection (FI) Framework, which aims to support the development of fault injectors on the platform. These tools will be described in the next sections.

3.1 MRReliabilityTest

As Hadoop is an open-source platform developed by a community, there is a strong concern to maintain a set of tests to help the validation of the platform features. These tests are distributed with the tool and comprise two types: unit testing and integration and system testing. Unit tests are intended to check individual classes, while integration and system tests aim to verify the operation of one or more modules that interact with each other.

Concerning to fault tolerance, we found only one system test to verify the platform reliability. This test, called *MRReliabilityTest*, is available since version 0.20.0 (launched in 2009), and aims to run some synthetic MapReduce applications under faults. The test runs in a distributed environment (with several nodes) or in a pseudo-distributed environment (with several processes on a node). It was necessary to analyse the source code to understand its operation, as we have no additional documentation.

To verify the system behavior, *MRReliabilityTest* injects faults in the nodes that execute the applications. The operation is as follow: initially the test triggers jobs (objects of Job class) whose tasks are spread on every available node. These tasks correspond to Map phase of the MapReduce paradigm. Then, the test injects faults in some of these tasks using the own interface of the objects whose represents the tasks under execution (R*unningJob* class from Hadoop platform). The platform must restart the tasks that fail. Further, the test causes failures in processes of TaskTracker class, through 'kill' operating system command. Thus, the platform must re-schedule the tasks which run in the nodes whose *TaskTrackers* failed. These operations are repeated for three task types attached in the Sort job sample of the platform (RandomWriter, Sort and SortValidator tasks). All tasks must successfully complete even after the failures.

3.2 Fault Injection (FI) Framework

The Fault Injection (FI) Framework arises in 2010 on Haddop platform, in order to support the development of fault injection code [1]. This framework is based on AspectJ, an aspect-oriented programming (AOP) tool built in Java [7]. The aspect-oriented programming easily separates the fault injection code from the implementation target, the original code. Instead of writing the code for a fault tolerance mechanism as part of the target program functions, it is written as independent code segments called *advices*.

The Aspect-Oriented Programming (AOP) has been an interesting approach to fault injection, since the AOP paradigm allows methods to be intercepted and changed at runtime, without interference in the application code under test. The instrumentation code for AOP allows classes of elements to be intercepted and instrumented, requiring only knowledge of their APIs.

Through encapsulation of the fault injection using aspects, the fault injection reaches several modules of the

target application, which may affect methods executed in different classes of the application under test.

The failures emulated with FI Framework are no deterministic, that is, no one previously knows when a failure will occur. Besides, the failure occurrence is controlled by a classical probabilistic model, in which we set a probability of a fault occurrence. To ensure that a failure will really occur during an execution, it is advised to set the probability percentage to 100%.

To develop a fault injection, we must create an aspect in *AspectJ* (file .aj). This aspect must import the framework class of the probability model (org.apache.hadoop.fi. ProbabilityModel) and must define where and how the failure will occur. The fault location is defined by a *pointcut* in *AspectJ*, which selects the points of the original code which will be intercepted during the execution. For these points of the code we define advices in *AspectJ* to specify the additional code to be executed when the interception occurs. The fault injection examples provided by the fault injection framework often use the advice called *before.* This advice executes just before the execution of the selected point of the code. The fault itself is specified in this part of the aspect, and usually consists on an exception trigger.

4 Experimental Analysis

The mechanisms described in the previous section were analyzed in detail and tested. The following subsections present an evaluation and validation through MRReliabilityTest and FI Framework tools.

4.1 *MRReliabilityTest* Experimentation

To MRReliabilityTest experimentation, we use a Hadoop stable version (0.20.203.0). In this test, Hadoop was configured to operate in a pseudo-distributed mode, running in a server of the Computer Systems Laboratory (LSC-UFSM). The test resides in a .jar file for testing MapReduce and the applications executed by the test were in the set of examples of MapReduce. We notice that the test jobs were divided into four parts, in which each part was subsequently divided into other four parts and executed in pairs, totaling four jobs in the Map and one in the Reduce phases.

At first, there was a stage in which the tasks were forced to fail, with tasks being abruptly terminated and later recovered by the Hadoop fault tolerant mechanism. We notice, as expected, that the tasks were aborted once a run and recovered.

Secondly, there was a stage in which the *TaskTracker* processes were forced to fail. In the beginning of this phase, some jobs could not be recovered and the replication causes an excessive use of processing and storage, leading to locking the machine running the test. The process had to be manually aborted to contain the attempts of the system to further replicate the tasks that had been aborted. It was concluded that, in this case, the test can not be totally used to determine the MapReduce reliability, since only one of the four jobs had success in ending and tasks recovering.

4.2 FI Framework Experimentation

This experimentation aims to explore the FI Framework, seeking to corroborate the instructions in its documentation to identify fault injection opportunities provided by the tool and to find limitations in its use. For this, we must first know the Apache Hadoop development details (source code, compilation environment and available tests). Next, we develop and test fault injection features in the four main components of Apache Hadoop: *DataNode*, *NameNode*, *TaskTracker* and *JobTracker*.

For the experiments, we considered two versions of Apache Hadoop: 0.21, a development version, in which FI Framework was originally included; and 1.0.3, a Hadoop stable version. Both versions were obtained from the repository to Hadoop developers. The phases of each experiment were: (i) analysis of the component source code, in order to define the fault injection point; (ii) definition of an exception to characterize the failure; (iii) selection of a test potentially affected by the failure; and (iv) execution of the test with and without fault injection, observing the component behavior. For reasons explained below, it was only possible to test the fault injection in the *DataNode* and in the *NameNode*. The tests were performed using the Hadoop pseudo-distributed mode (one machine running all processes). Moreover, we use the original compilation environment of the tool (original configuration present in `build.xml` file of each version).

4.2.1 Fault Injection in *DataNode* and *NameNode*

The first aspect created focused on fault injection at startup of a *DataNode*. From what we know of the Hadoop architecture, this type of failure is not suported in the failure model of HDFS. However, we decided to emulate it to observe the behavior of the tool in this unrecoverable situation. Analyzing the source code of the *DataNode* class, we found a method, called `startDataNode`, responsible to its initialization. So, we created an aspect file (DataNodeStart.aj), located at tests directory, which collects all files ".aj" to weave with the original code in the compilation step. Seeking to emulate a software external failure, the aspect adds an exception in the code signaling a memory failure (OutOfMemoryError) (Figure 2). To test the injection of the failure, we choose the TestFileCreation.java program, which considers several test cases during the creation of a file in HDFS. With this failure injected, we notice that Hadoop can not handle such error, causing a widespread failure (the program test remains trying to run operations on *DataNode*, which does not exist, and thus, the errors accumulate, resulting in the application interruption).

```
privileged public aspect DataNodeStart {
    public static final Log LOG =
       LogFactory.getLog(DataNodeStart.class);

    pointcut execstartDataNode():
       execution(* startDataNode(..));
    before() throws OutOfMemoryError :
       execstartDataNode() {
          throw new OutOfMemoryError("FI:" + " OutOfMemoryError");
       }
}
```

Figure 2: Fault injection aspect related to *DataNode*

The second aspect created was focused on *NameNode*, at the point where this component allocates blocks in the file system (`allocateBlock` method from `FSNamesystem` class). We know that failures in *NameNode* are not supported by the tool. We have emulated a concurrency problem, provided on HDFS protocol (org.apache.hadoop.hdfs. protocol.AlreadyBeingCreatedException), in which it detects that a block is already being created (Figure 3). We choose an HDFS test program, called TestLoadGenerator.java, which tests the balancing of replicas of a file. The test execution had also produced results as expected.

```
privileged public aspect NameNodeAllocate {
    public static final Log LOG =
      LogFactory.getLog(NameNodeAllocate.class);

    pointcut execallocateBlock():
      execution(* allocateBlock(..));

    before() throws IOException : execallocateBlock() {
      throw new AlreadyBeingCreatedException("FI:" + "Failure inserted in " +
                thisJoinPoint.getStaticPart().getSourceLocation());
    }
}
```

Figure 3: Fault injection aspect related to *NameNode*

4.2.2 Fault Injection in *TaskTracker* and *JobTracker*

The third and fourth aspects developed focused on *TaskTracker* and *JobTracker* respectively. For *TaskTracker*, we focused on a new task initialization, where it needs to initialize a directory to the task (`localizeJob` method from `TaskTracker` class). The aspect was created following the same steps described above, but at this time producing an exception of type *IOException*, emulating an input failure during a directory creation. For *JobTracker*, we focused on the initialization of the process itself (`startTracker` method from `JobTracker` class), also producing an exception of type *IOException*. To test both aspects, the experiment chosen was ReliabilityTest.java. This experiment executes several reliability tests on the platform, causing the ending of the execution processes.

The tests with the third and fourth aspects were not performed because their source codes were not weaved with the original code as expected. Analyzing the original configuration of the compilation environment (`build.xml`) from version 0.21, we notice that it have no targets prevision to compiling fault injection on MapReduce, only on HDFS. As we analyze, FI Framework could be used in any of Hadoop components, but we notice that the weaving of aspects in MapReduce layer would require significant changes in `build.xml`, mainly to solve dependency problems.

Analyzing subsequent stable versions of Apache Hadoop platform, we notice that FI Framework continued to be distributed, but many of the examples of fault injection were removed. It was found that the examples of version 0.21 were related to classes that had been removed or changed in the stable version. In addition, we notice that the compilation environment was updated on the stable version, adding new compilation targets to fault injection, which have no information in FI Framework documentation. However, with this new compilation environment, it was not possible to inject the failures developed in this work, as none of the compilation targets had weaved the aspects with the source code. This problem occurs in all versions of Apache Hadoop

after 0.21. We tried to modify the file `build.xml` to solve this problem, but we didn't succed given the lack of documentation.

5 Conclusions

In this work, we analyze two alternatives to test the behavior of Apache Hadoop in face of failures. The first alternative causes failures accessing processes and objects involved in the implementation of MapReduce (*JobTracker* and *TaskTracker*). Analyzing the *MRReliabilityTest* source code, it is possible to understand how the failures are injected in the components. However, this solution is difficult to reuse to inject failures in other components, since the tests were only targeted to MapReduce.

The second alternative, the FI Framework, allows to inject failures in a systematic way. It allows to develop aspects that inject failures in any class of the platform. However, we notice that some files originally distributed with the framework (such as examples of aspects and some test programs using classes with injected failures) are missing in newer versions of Hadoop, apparently due to interface changes of some classes.

The experiments have shown advantages in the FI framework use, arising out of aspect-oriented programming. The fault injection source code is completely separated from the tool source code and can be combined with this in due course. In addition, we can reuse existing tests, changing its behavior through aspects, allowing easily create new tests not included into the platform.

However, unlike the documentation describes, FI Framework is not perfectly integrated into the platform. An example of this is that its documentation is not fully consistent with the stable version of Apache Hadoop. We believe that these problems may be overcome by the easiness introduced by the framework, so we intend to continue this work, first adjusting the compilation environment and then creating new fault injection tests.

The results of the experiments described here do not demonstrate the lack of reliability of the mechanisms, only indicate that the solution available to fault injection on Hadoop are not so easily applicable. Indeed, the documentation available for both tools is rather scarce and present some inconsistencies between the documentation itself and the source code distributed in the stable version of Hadoop. Further tests with log analysis and source code analysis are needed to explain the errors presented here.

Finally, it is worth noting that this work focus attention only on the features available by the platform itself. Another approach, which can be explored in future works, would be to use external tools to cause failures in Hadoop components.

6 References

[1] Apache Software Foundation. Fault injection framework and development guide. Available: https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/FaultInjectFramework.html

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, 2004.

[3] Apache Software Foundation. HDFS Architecture Guide. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[4] Apache Software Foundation. MapReduce Tutorial. Available: http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[5] A. N. Bessani, V. V. Cogo, M. Correia, P. Costa, M. Pasin, F. Silva, L. Arantes, O. Marin, P. Sens, and J. Sopena. Making Hadoop MapReduce byzantine fault-tolerant. In: *Proceedings of the International Conference on Dependable Systems and Networks*, 2010.

[6] A. S. Foundation. Welcome to Hadoop! Available: http://hadoop.apache.org

[7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwing. Aspect-Oriented Programming. In: *ECOOP – European Conference on Object-Oriented Programming*, Finland: Springer, pp. 220-242, 1997.

[8] J. A. Clark and D. K. Pradhan. Fault injection: a method for validating computer-system dependability. *Computer*, vol. 28, pp. 47–56, 1995.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI – USENIX Symposium on Operating Systems Design and Implementation*. San Francisco, California: ACM Press, 2004, pp. 137-149.

[10] K. Echtle and M. Leu. Test of fault tolerant distributed systems by fault injection. In *FTPDS – Workshop on Fault-Tolerant Parallel and Distributed Systems*. IEEE, 1994, pp. 244-251.

[11] A. Goldman, F. Kon, F. Pereira Junior, I. Polato, R. F. Pereira. Apache Hadoop: theoretical and practical concepts, developments and new possibilities. In: XXXI Jornadas de Atualização em Informática (in portuguese), 2012.

[12] Paul D. Marinescu and George Candea. 2011. Efficient Testing of Recovery Code Using Fault Injection. *ACM Trans. Comput. Syst.*, vol. 29, n. 4, pp. 1-38, 2011.

[13] A. Sangroya, D. Serrano and S. Bouchenak. MRBS: Towards Dependability Benchmarking for Hadoop MapReduce. In: Workshop on Big Data Management in Clouds (BDMC). Rhodes Island, Greece, Aug. 2012.

Architecture of a Java-Based Simulation System

Ray Kresman

Department of Computer Science
Bowling Green State University
Bowling Green, OH 43403
{kresman}@bgsu.edu

ABSTRACT

This paper presents the architecture of a system for distributed simulation where the user problem is modeled and implemented in Java. The native platform, however, is a C++ based distributed simulation engine, BGTW, that employs optimistic approach to the time management of the simulation events. An intermediate layer, Java Time Warp, provides access to the simulation engine for users programs written in Java. The hardware layer is a cluster of networked workstations. The MPI library is used as the communication layer between the compute nodes. This layered architecture facilitates reusability of the native engine and provides a portable platform for the user programs. Our API handles the transparency issue. This approach can be extended to provide a distributed web-based simulation platform.

1. Introduction

This paper suggests a layered architecture where each layer handles one or more of the three issues of portability, reusability and transparency. At the first layer BGTW, Bowling Green Time Warp [1, 2] furnishes a distributed simulation engine using Time Warp synchronization scheme [3]. This layer **is** implemented in C++ over a MPICH [4] communication layer and supports transparency through its Application Program Interference (API) which hides the parallel programming synchronizations and details.

User's application program can be modeled and implemented in one of two ways: directly on top of BGTW using C++ based API of the engine, or on top of a *new* Java Time Warp (JTW) middleware using Java based API. Using the former, the portability is limited by C++ and the MPICH. We chose the latter approach as it enhances portability through Java and its virtual machine. The JTW layer connects BGTW to the application layers and provides the functionalities needed for the Java programmer to model their problem in a distributed simulation platform. Reusability issues are handled by the object-oriented programming paradigm of the platform.

A number of distributed simulation technologies have been proposed in the literature. Discrete Event System Specification (DEVS) formalism [5] provides a solution to an ideal distributed simulation environment when implemented over middleware technologies. Distributed simulations based on DEVS include many systems, for example DEVS/CORBA [6]. Agent-Based system uses a distributed discrete event simulation model in the simulation of multi-agent systems [7]. Simulation-specific standard such as HLA and test-range specific middleware such as the Test and Training Enabling Architecture (TENA) [8] provide higher levels of dedicated support for distributed simulation. For reusability, several distributed simulation framework utilize CORBA [9], Source Content Object Resource Model (SCORM) [10]. Muralidhar et al [11] proposes a Java-based architecture for web-based interactive distributed simulation. They both use Java Proxies, sockets and HTML for interaction. Other examples of Java-based distributed engine include IDES [12], JiST [13], DSOL [14], Jane [15], DEVS/RMI [16], and a Java-based conservative distributed simulation [17].

2. A Layered Abstraction

This paper concerns a distributed simulation environment with an underlying Message Passing Communication (MPC) architecture. Exchange of messages between Processing Elements (PEs) requires an easy to use message passing system. The peer to peer communication between PEs is captured by the Message Passing Interface standard (MPI) [18]. MPI allows asynchronous and synchronous type of computation. MPI library is easy to use and is available for most processor architectures. Though the MPI standard specification is language independent, software support for MPI includes primarily Fortran, C and C++.

There have been some efforts at providing support for Java binding and to provide a standardized MP binding for Java, MPJ [19]. A pure java message passing system is attractive because of its portability, support for object oriented programming and modular architecture. MPJ when fully developed, will provide advanced features of the MPI specifications including derived-data types, virtual topologies, different communication models including group communications. mpiJava [20] provides JNI wrappers to native MPI software. mpiJava relies on the platform-dependent native MPI implementation for the PE. The java interface, through the JNI, invokes these underlying native methods that are implemented in another language (C or C++). The implementation generally works well though some run time issues have been reported due to the evolving nature of the Java language itself. An alternate approach is to implement the MPI functionality, ground up, in Java. Couple of approaches in this category include MPIJ. MPIJ is implemented within DOGMA system and communication uses native marshaling of primitive Java types. Communication speeds of MPIJ are comparable to native MPI implementations. Though Java interface for MPI involves issues beyond the plain description of language bindings, there is light at the end of the tunnel since the Java language and the original MPI specification are both object based. A pure java based MPI was not ready for prime time, a few years ago, and even now not all issues have been resolved with a pure java based MPI.

MPICH [4] is an implementation of the Message-Passing Interface (MPI) in C++. It provides an MPI implementation for platforms, including clusters, SMPs, and massively parallel processors. MPICH is robust and works nicely in a cluster computing configuration, such a Beowulf Cluster. Our simulation platform is implemented in a Beowulf Cluster with 16 compute nodes, a 1 Gbps Ethernet switch, and a server. We use MPICH as the communication paradigm in this environment.

On top of the MPICH communication layer, we use BGTW distributed simulation engine [1, 2]. The engine is implemented in C++. By having both the communication layer and the engine in the same language we avoid many of the language binding issues that one would encounter.

The central question is how Java users can access the simulation engine. Given the maturity of MPICH and perhaps some Java based hybrids one can envision two approaches, as illustrated in Figures 1 and 2.

In approach 1 (Figure 1), a middle layer, Java Time Warp, acts as a Java wrapper for BGTW. The real simulation work is done by BGTW which in turn uses MPICH for communication. JTW is responsible for object management and information exchange between end user Java application and BGTW. JTW uses Java Native interface (JNI) to interact with the native (BGTW) code, the same way mpiJava interacts with the native MPI library.

In approach 2 (Figure 2), Java Time Warp sits below mpiJava; mpiJava together with MPICH act as the communication layer. However, one needs to build a distributed simulation engine as part of JTW before JTW can provide a Java based simulation interface to end users.

Approach 1 is preferred for many reasons:

1. Cleaner Interface. By separating the simulation engine from the mechanics of providing a Java interface to end users, one can concentrate on the service being provided to end users rather than simulation issues.

2. Consistency. JTW provides a set of consistent and transparent interface to Java users. As BGTW continues to evolve, JTW can immediately benefit from the gained improvements.
3. Less complexity. Redesigning a parallel simulation engine from the scratch is a complex and expensive undertaking. Complicated algorithms should be considered for. Building a Java wrapper for an existing simulation engine using JNI is much easier and controllable.
4. Similar performance. As we can see, both the approaches have JNI that helps in the interaction between native code and Java program. The only difference is JNI appears in different places. Thus, one would expect the two approaches to have a similar performance.

In summary, our design of JTW uses BGTW simulation engine and makes it available to end user programs written in Java. In the next section, we provide an overview of BGTW.

3. BGTW Simulation Engine

This section provides an overview of the BGTW. More complete details can be found in [1, 2]. The design of the simulation engine, BGTW, is guided by some basic principles. The platform should handle transparency of the underlying parallel complexities. The user of this system may not be familiar with parallel programming or distributed simulation. Further, the engine should have a small, well defined, and easily understandable user interface. Additionally, the engine should be amenable to future extension. For example, it should support implement a new synchronization algorithm or other PDES algorithms. This requirement was a key design issue and concerned modularity of the engine's components.

Thus we wanted to have a modular design with a small set of core functions and a set of APIs. There is a small kernel running on each PE, called *Process*. To ensure that only one process is running on a compute node we use the Singleton design pattern. The process knows all other processes in the simulation and additionally knows where the user defined Logical Processes (LPs) are running. Each kernel process is responsible for a distinct subset of the user LPs; the mapping is statically for now, but easily can be modified to a more sophisticated mapping algorithm advised by a partitioning scheme.

BGTW uses an optimistic approach to time management. Time management is based on the time warp mechanism developed by Jefferson [3]. The Time Warp protocol is transparent from the application program. The rollback mechanism, error correction of Time Warp, and GVT (Global Virtual Time) calculations are completely invisible from the users. To maintain correctness of the simulation, the users are required to register state-variables of any user defined simulation entities (i.e. LPs).

The kernel process uses basically four other modules to delegate the work to: a GVT module, a Message Handler, a Memory Manager, and a Queue. Additionally, a Thread class is defined to ease pthread.

The messages are represented by the Msg Class. This is the base class for the user defined message types. The template Message derives from Msg and the user defines its Messages by using this template with his/her payload class.

To define an LP the user derives their class from an abstract LP class, e.g. ***twlp***, and implements the virtual functions ***run(), init(),*** and ***finish()***. The first one is called when a message for that LP is processed and the other two are to initialize and finalize the LP, respectively.

The user has to write the main method where the process is initialized and the LPs are registered to the engine. The simulation starts by calling ***simulate()*** method of the kernel process. Finally, ***finalize()*** is called before exiting the program. The output is written to files, one for each process. Optionally a debug output is produced to trace the internals of the simulation engine.

4. Design Overview of JTW

We note that BGTW provides two major functionalities to the end user: *process* and *twlp*. Class process is responsible for the steps necessary to configure and execute a simulation task. Class twlp is used to declare and implement the behavior of each logical process in the underlying simulation. Distributed simulation partitions the simulation task into several logical processes. Each computing node normally house more than one logical process. Note also from Section 3 that the main behavior of one logical process "twlp" consists of initialization, run and finish phase which are provided in BGTW as three abstract functions: init(), run() and finish() for the user to define and implement the behavior of each logical process. By implementing these functions the user specifies the mechanics of the actual simulation. Our goal is to provide the same interface to Java users, through JTW.

JTW communicates with BGTW through BGTW's Applications Programming Interface (API). It's implemented in C++ and communicates with the Java interface through Java Native Interface. As a wrapper, JTW provides similar functionalities as BGTW: jtwlp and jprocess for Java user. In addition, JTW provides the same abstract interface in Java side for users to implement their task. The main functionality of these three BGTW abstract functions was implemented inside JTW to exchange simulation data between Java and C++ program through JNI.

Like BGTW, JTW has four major components: message processing, object management, logical process implementation and data exchange (Figure 3).

The workflow between user space and the backend simulation engine is captured in JTW, as illustrated in Figure 4**.** First, a "process" instance is created and initialized through the corresponding JNI function call. Several "twlp" instances are created and registered to this "process" instance. All the c++ object information is recorded in Java side for later use. Then the simulation task starts execution. The simulation process continues to loop through "run" phase until the end simulation time is reached. For each "run" iteration (the boxed item of Figure 4), any "twlp" object instance may receive simulation messages from other logical process and pass message data with other simulation data to "jtwlp" instance. Then "jtwlp" instance perform simulation actions which is defined by the user on simulation data, and finally passes newly computed simulation data back to "twlp" instance for update. Finally, the finalize() function is called from Java side to C++ side which ends the simulation.

User interface consists of two main Java classes: *Jprocess* and *Jtwlp*. Each class has a set of predefined native methods provided to the users for their simulation application. Native method means real implementation of this method is in another language; it only has a prototype defined on the Java side with no real implementation. Class Jprocess is designed for configuring and executing a simulation task, class Jtwlp is designed for declaring and implementing behavior of each LP; user's LP should inherit from this class.

5. Concluding Remarks

In this paper we have provided a design view of our layered architecture for a Java-based distributed simulation. Where applicable we have used existing technologies and applications. It is our hope that this modular design helps in accelerating the development process. Our approach permits Java users to access an existing distributed simulation engine. The interfaces are well designed and native methods in C++ are available to Java users using the mechanisms of Java Native Interface.

Reference

[1] Rajaei, H. Ehmmer, M., and Roeck, H., "BGTW: A Clustered-Based Distributed Simulation Platform" to appear In Proceedings of 2006 SCS International Conference and Simulation - Methodology, Tools, Software Applications, M&S-MTSA06, August, Calgary, Canada.

[2] Ehammer, M, and Roeck, H., "Bowling Green Time Warp", Department of Computer Science, Bowling Green State University, OH, Technical Document, August 2004.

[3] D. R. Jefferson. Virtual time. ACM Trans. Program. Lang. Syst., 7(3):404–425, 1985.

[4] MPICH--a portable implementation of MPI. http://www.mcs.anl.gov/mpi/mpich/

[5] Bernard P.Zeigler, Tag Gon Kim and Herbert Praehofer, “Theory of Modeling and Simulation”, Academic Press, 2000.

[6] Bernard P.Zeigler, Doohwan Kim, Stephen J. Buckley, “Distributed supply chain simulation in a DEVS/CORBA execution environment”, December 1999 Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future - Volume 2.

[7] Logan, B., Theodoropoulos, G. "The Distributed Simulation of Agent-Based Systems", IEEE Proceedings Journal, Special Issue on Agent-Oriented Software Approaches in Distributed Modeling and Simulation, February 2001.

[8] TENA-- Test and Training Enabling Architecture, https://www.tena-sda.org/

[9] Chan, A. and Spracklen, T,, "Web-Based Distributed Object Simulation Framework", In proceedings of 1999 SCS Summer Simulation Conference.

[10] SCORM-- Source Content Object Resource Model, http://www.adlnet.gov/scorm/index.cfm

[11] Muralidhar, R., Kaur, S., and Parashar, M., "An Archituecture for web-based Interaction and Steering of Adaptive Parallel/Distributed Application", in Proceedings of EuroPar 2000

[12] Nicol, D., Johnson, M., Yoshimura, A, and Goldsby, M, "A Java-based Distributed Simulation Engine", in Proceedings of Sixth IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications, Systems (MASCOTS'98)

[13] Barr, R., Haas, Z., and Renesse, R. "JiST: Embedding Simulation Time into a Virtual Machine", In Proceedings of EuroSim 04, September 2004.

[14] Lang, N., Jacobs, P., and Verbraeck, A., "Distributed, Open Simulation Model Development with DSOL Services", in Proceedings of the 15th European Simulation Symposium, ESS 2003

[15] Perumalla, K., and Fujimoto, R,. "Jane: An Architecture for Interactive Parallel Simulations", in Proceedings of Web-based Modeling and Simulation (WEBSIM) 1999 conference.

[16] Ming Zhang, Bernard P.Zeigler, and Phillip Hammonds "DEVS/RMI—An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies.

[17] Ferscha, A., and Richter, A., "Java Based Conservative Distributed Simulation", In Proceedings of 1997 Winter Simulation Conference, WSC97.

[18] MPI-- Message Passing Interface standard, http://www.mpi-forum.org/

[19] Mark Baker and Bryan Carpenter. MPJ: A proposed Java message-passing API and environment for high performance computing. International Workshop on Java for Parallel and Distributed Computing, Cancun, Mexico, May 2000

[20] Mark Baker, Bryan Carpenter, Geoffrey Fox, Sung Hoon Ko, and Xinying Li. mpiJava: A Java interface to MPI. First UK Workshop on Java for High Performance Network Computing, September 1998.

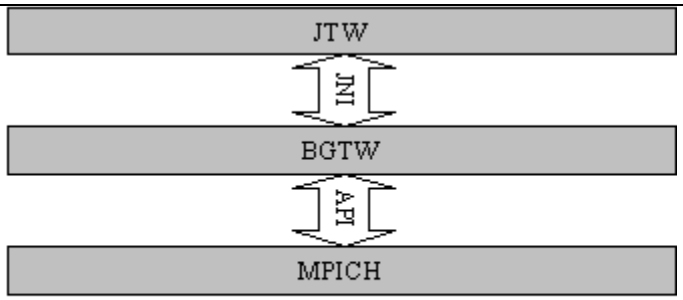


Figure 1: Approach 1 - Java Time Warp is a Wrapper to the Simulation Engine

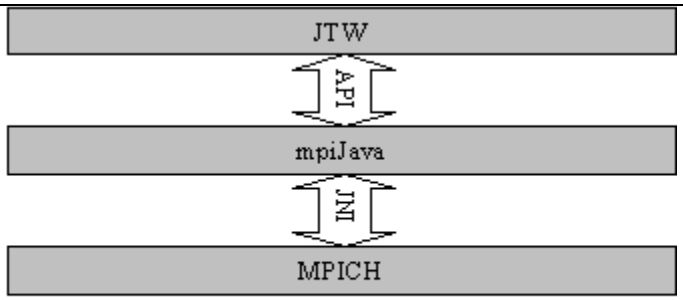


Figure 2: Approach 2 - Java Time Implements A Distributed Simulation

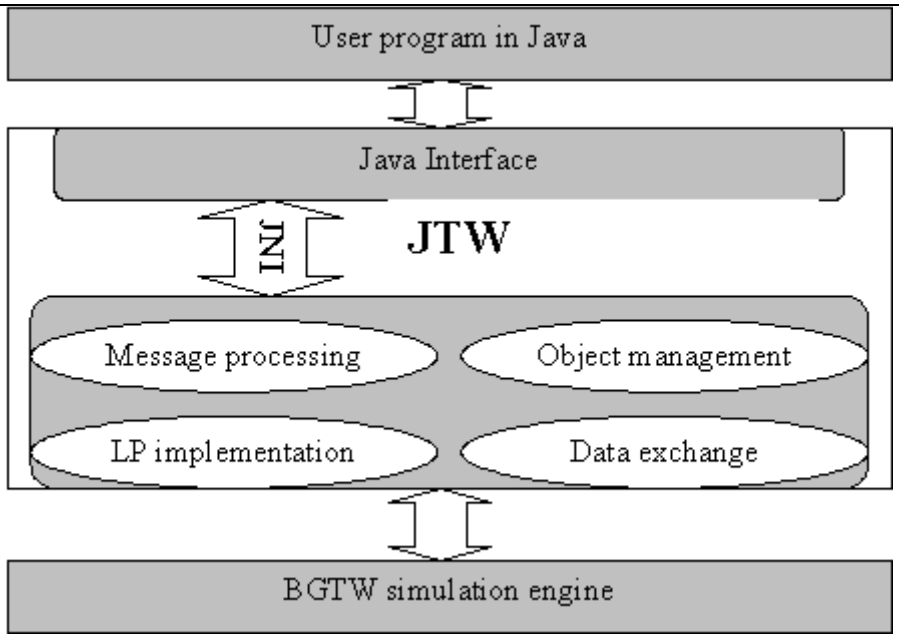


Figure 3: Design Layout of JTW Components

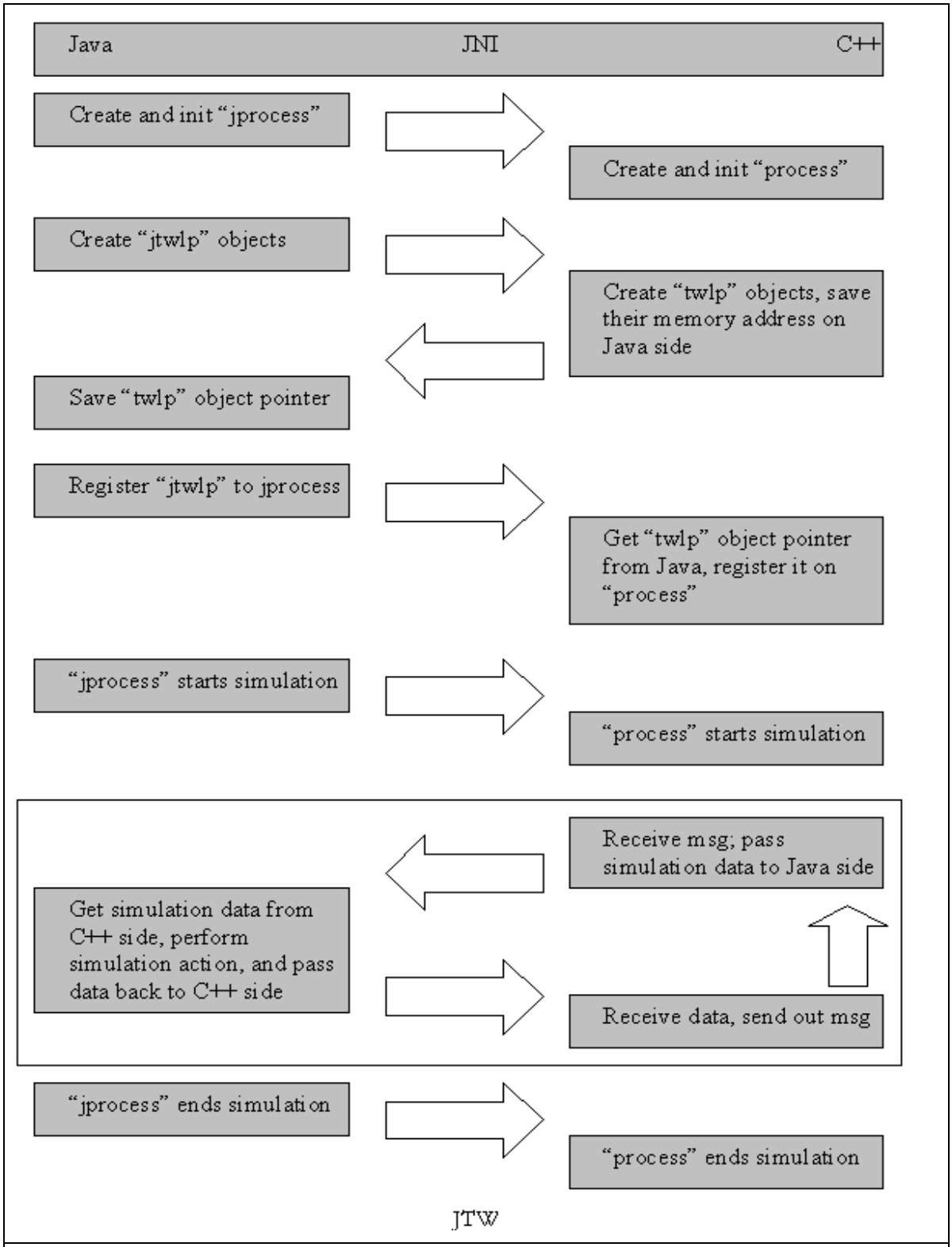


Figure 4: Communication Between the Simulation Engine and User Space

MPI Communications Management in Cloud

Laura Espínola[1], **Daniel Franco**[1], **and Emilio Luque**[1]
[1]Computer Architecture and Operating System Department
Universidad Autónoma de Barcelona (UAB)
Barcelona - Spain

Abstract— *Cloud computing is an important branch in computer science and HPC, currently many distributed computing tasks can be performed on virtual machines that are parts of clouds public or private, and all of this without needing physical cluster. For this reason several distributed computing tasks such as scientific applications are being moved from clusters to clouds. MPI (Message Passing Interface) is a key component in common and distributed computing tasks. The virtualized environment hides the network topology information of the users, and existing optimizations based on network topology for MPI applications are no longer viable in the cloud environment, so that improvements are needed regarding these factors. Our proposal is studying the feasibility of improved latency and network traffic for parallel applications running on a cloud environment. In addition we present a study of the behavior of MPI communications in Amazon EC2 public cloud. After analyzing the current situation, we design a method called MPI Communications Management (MCM).*

Keywords: Cloud Computing, MPI communications, HPC, parallel applications.

1. Introduction

Cloud Computing has become a popular paradigm for many distributed and parallel applications. Especially considering basic concepts such as the facility to acquire computing resources on demand, scalability and a business model in which the criterion of *Pay-Per-Use* is used [1] [2].

This paradigm is currently applied in scientific applications and data analysis. It is used especially in the world of High Performance Computing (HPC) for the scalability offered. Many HPC Applications are deployed in public clouds like Amazon's Elastic Compute Cloud (EC2)[1] with good results, but the efficiency of networks communications in virtualized environments is still affected the quality of its services.

Message Passing Interface (MPI) is a common standard software component used in distributed and parallel applications in HPC. A critical factor for MPI applications on a cloud environment are the communications between the nodes [3]. In this research we design a method to improve MPI communications, providing a dynamic management in cloud computing environments.

In early days, algorithms developed to optimize the performance of parallel applications, including MPI applications are those with knowledge of network topology and communication patterns, specifically when it comes to a cluster environment. In clusters the execution of parallel applications reveal a repetitive behavior of their communication patterns, this is useful to develop adaptive routing algorithms that reduce and maintain low latency values enhancing the communications [4].

In a cloud environment, network topology is not a relevant fact, moreover, normally is not available. Virtualization technique is used on clouds and it handles the management of computing resources. Virtualization typically hides the network topology from users, providing an uniform interface, without exposing the underlying hardware and software configurations. Another key factor by which the topology is hidden lays on the fact that when virtual machines are created, its network flows are dynamically configured [5].

Our research presents a methodology that provide dynamic management of MPI communications in cloud environments. *MPI Communications Management* (MCM). MCM is based on Predictive and Distributed Routing Balance(PR-DRB) method, it proposes an alternative path distribution considering the latency and possible congestion but keeping in mind the characteristics of the cloud.

This paper is organized as follows: in section 2, we realize a brief introduction above the state of the art; in section 3, we detail our design method with its parts and phases; in section 4, we present the experimental environment; in section 5, we present experiment results and our analysis; in section 6, we conclude this paper with some ideas remarks.

2. Related Work

Today the scalability of HPC in a Cloud environment depends largely on the effective support of network communications in virtualized environments.

A series of studies are analysed in the research area, in this section we introduce some of the main topics related to our study.

2.1 Cloud Computing and HPC

Cloud Computing is the long-held dream of computing as a utility, it has the potential to transform a large part of the

[1]Successful use cases are found in `http://aws.amazon.com/solutions/case-studies/#hpc`

IT industry, changing the way IT hardware is designed and purchased [6]. The elasticity that it provide makes it even more attractive for HPC users.

The insertion of HPC in cloud environments has been growing over the past few years. Cloud Computing have attractive features for HPC, like: availability, computational power, performances improve and elasticity. Many HPC applications are deployed in public clouds, and provides to users facilities in the installation and maintenance of physical resources, users only pay for the services used on demand.

HPC in cloud also have some key challenges, the lack efficient communication support in virtualized network is one. This problem inhibit parallel applications to take advantage of high performance networks [7]. One experiment in MPI applications that verified this as showed in Fig. 1, the end-to-end delay of many clouds are worse in magnitude than NCSA cluster [8] [9].

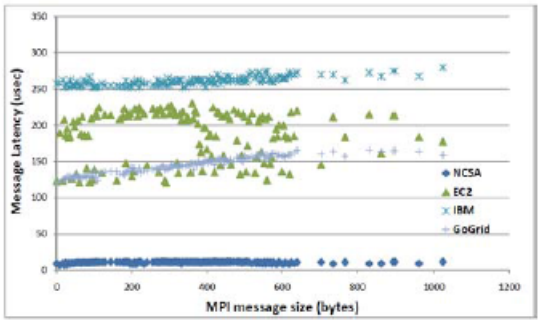


Fig. 1: MPI message latency in clouds

Inefficiency in virtualized networks can not be afford with traditional method of optimization like topology aware algorithms because this information is in underlying layers, and is hide from users.

There are many techniques of topology aware algorithms, in this section we want to introduce one of them that is significant for our approach. An effective method to control network efficient in clusters called Predictive and Distributed Routing Balance, it controls network congestion based on paths expansion, traffic distribution, applications patterns and speculative adaptive routing [10].

PR-DRB looks better response time using cached communications and alternatives path. Its model has four basic procedures:

- **Monitoring**: It includes the tasks of latency values accumulation and contending flows identification, performed at intermediate routers.
- **Notification**: It is initiated at destination end nodes. An Acknowledge (ACK) message with path information is created and sent back to the source.
- **Path Configuration**: This part involves the configuration of new alternative paths (Metapath Configuration) according to latency values, also performed at source nodes. If there are saved solutions for a congestion situation, the paths are taken from the saved solution database. Otherwise, new alternative paths are created.
- **Path Selection**: when new messages are injected into the network, selection procedures distribute messages among the paths configured in previous task.

Our work extract the main idea of PR-DRB and design a methodology that are specifically for MPI Communications Management in cloud, with the problems that virtualization impose. .

2.2 Amazon EC2 Cloud platform

For this research we select one of the most popular cloud computing platform, Amazon EC2. Because it has been the target platform for numerous academic and commercial applications [9].

In Amazon EC2, the leased virtual machine instances provide to the user a highly customizable operating system environment. The users have complete control of their instances, they have root access to each instance, and they can interact with them as they would.

This characteristics allowing users run applications of distributed data analysis, scientific simulations and HPC. Recently, some large physics experiments such as STAR [8] have also experimented with building virtual-machine-based clusters using Amazon EC2 for scientific computation.

Amazon provides to the users the choice of multiple instance types, operating systems, and software packages. Amazon EC2 allows to select a configuration of memory, CPU, instance storage, and the boot partition size optimal for their choice of operating system and application. The relative processing performance is given in Elastic Compute Units (ECU) where one ECU corresponds roughly to the equivalent CPU capacity of a 1.0 - 1.2 GHz 2007 Opteron or 2007 Xeon processor. The smallest machine can be instantiated with 1.7 GB RAM and Gbit Ethernet and has a single virtual core with a compute power of 1 ECU; the largest machine offers 60 GB RAM, 10 Gbit Ethernet and 88 ECUs, based on the actual dual eight-core Intel Xeon CPU systems [11].

These features took us to select Amazon EC2 as case of study to understand the network performance in the public cloud environment. We use the instance t2.micro, because it is a lowest-cost for general purpose instance. It has balance of compute, memory, and network resources that can be useful for our research.

2.3 MPI Communications

MPI is a widely used standard in HPC for message passing applications. It have been used for around two decades. Point to point and collective communications are provided to processes by this interface. Although, the network communications are abstracted from the processes,

hence applications have not control or information about the underlying network.

When the MPI processes of an application are launched, they communicate through a *Communicator*, which is part of the MPI implementation. The standard MPI message transmission model is illustrated in Fig. 2.

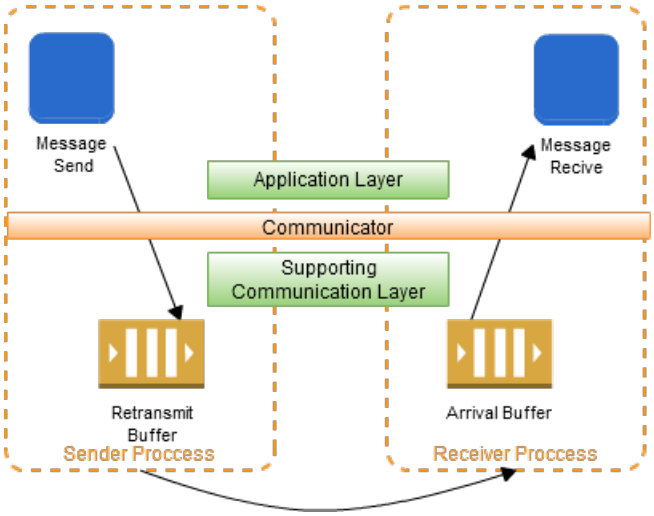


Fig. 2: MPI message transmission

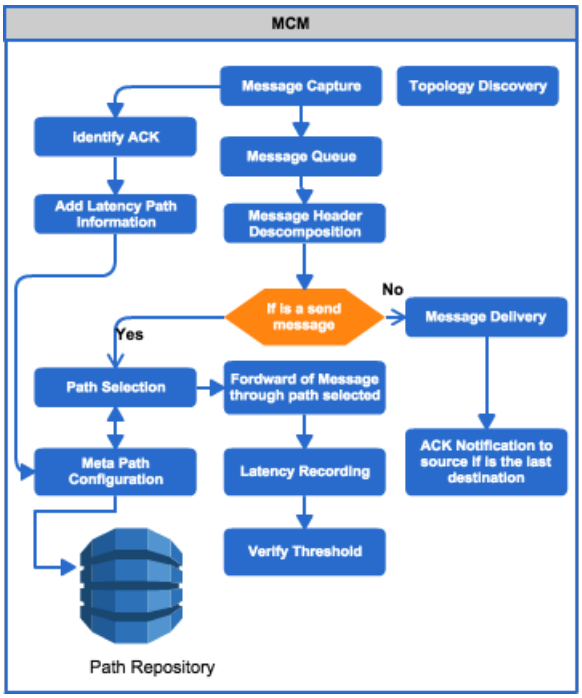


Fig. 3: MPI communications Management Design

Generally these processes communicate in a not uniform way, allowing unbalance of link usage. This problem decreases available bandwidth between nodes and generate performance degradation of MPI applications [12].

Most of the optimized algorithms for MPI operations implemented for a cloud environment are specialized either for latency or throughput, considering groups of virtual machines and the distance between them, but do not work for the manage of the message [13].

For this reason we propose a method that capture MPI communications and forward them, trying to balance the communications and achieve a better performance in this kind of applications.

3. MPI Communications Management

We design a communications management method for MPI based in the study of communication latencies between processes. The core of MCM is a daemon which captures sent and received messages between processes in order to compute alternative paths avoiding congested routes. After the selection of the better path is made, the message is forwarded through the calculated path. In this section we introduce MCM showed in Fig. 3.

Our approach initiates with the topology discovery, because it is based in PR-DRB, an algorithm in which the topology data is one of the essential inputs. This information is not available in clouds, so we need to find out similar data input. We have to discover information relevant to the topology. Thereby, we analyse the MPI communication between all pairs of virtual machines forming a cluster mounted in a Cloud. The network performance of processes in MPI applications is related to the network performance of their corresponding virtual machines. This allow us to know how the communications are carried out. Then this information have to saved in a repository for its posterior usage, because it is relevant for the selection of paths in MCM.

Messages routing decision take place when a message is sent or is received in a process of an MPI application. We propose to capture the message and alter the normal course of its communication. This message first is stored in a message queue. Then we analyse and decompose the message header to evaluate its source and destination.

When the capture is from a send instruction, we perform a calculation of path using the meta path configuration which uses the path repository to select the best route. After the best route is found, message is forwarded through this path. The latency is recorded in every hop in the path of the message, also, the verification of threshold is made. In another hand, when the capture is from a receive instruction, we send an ACK notification to the original source and deliver the message to the receiver process.

The capture of an ACK message, generates the update of the latency information for the corresponding route into the path repository.

During the initialization process, early messages are send through they normal path, but information about latency is

stored in the path repository. In further communications the evaluation of the path is made.

Repetitive communication behavior allow us to compute better routes. When a repetitive pattern appear we verify if the latency of the link is in the range accepted by a threshold. When the latency is acceptable we send the message using the standard route. If the latency in the current path is not acceptable, we compute a better path and store it in the path repository. Our approach creates a database with source-destination latency pairs, thereby, future messages can use it. Figure 4, show how MPI applications interacts with MCM in a cluster environment in Amazon EC2.

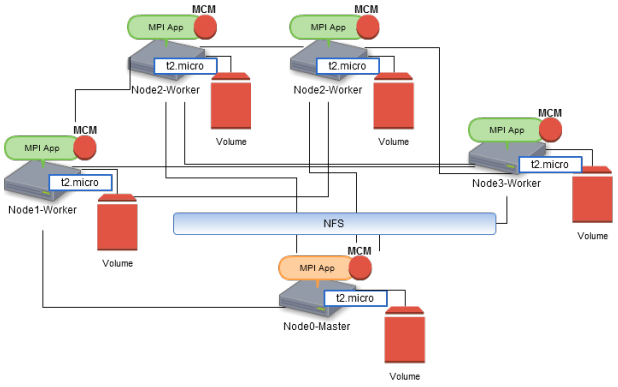


Fig. 4: Cluster Diagram in Amazon EC2

3.1 Fundamental components

- **Thresholds**: identifies the moment when an action must be taken. We select three areas, one with low latency, another with medium latency (where congestion is raising but network can handle), and finally one with high latency (with extreme congestion). Like PR-DRB in low and medium latency areas we do not open new paths, but when a transition to medium and high latency occurs, our approach searches a new path.
- **Path Selection**: This component handles the selection of path from the repository, also detects new paths solutions and stores them in the repository. The selection of path from the repository will be taken probabilistically.

4. Experimental Environment

4.1 Hardware Configuration

To analyze the functionality of our proposal, we conducted a series of experiments evaluating the latency of MPI communication in cloud Computing, Table 1 describes the main components and configurations of the system.

4.2 Software Configuration

To perform the experiments different types of software were used. Since from the lowest layer to the highest. We launch our instance from StarCluster, this is an open source cluster-computing toolkit for Amazon EC2.

Table 1: Experimental Setup

Component	Cluster Amazon EC2
Instance	t2.micro
Memory	1 GiB
Storage(GB)	EBS
Network Performance	Low to Moderate
Physical Processor	High Frequency Intel Xeon Processors operating at 2.5GHz with Turbo up to 3.3GHz
Clock Speed (GHz)	2.5 GHz
vCPU	1 vCPU

Each instance are Hardware-assisted virtual machine (HVM), this virtualization type provides the ability to run an operating system directly on top of a virtual machine without any modification, as if it were run on the bare-metal hardware. All the instances are acquired from US East (N. Virginia) data center of Amazon and their operative system is Ubuntu 10.11.

The measurement of point to point MPI communications latencies and bandwidth are obtained using OpenMPI, which is a High Performance Message Passing Library [**?**]. We measure the latency and the bandwidth as two key network performance metrics.

The program that realize the measures is BWLAT, this tool is open source and is part of the enovance project on clouds. They build and delivery solutions with open cloud.

4.3 Applications

We also have experiments running the NAS parallel benchmark. We select Conjugate Gradient (CG) with class B, it has many communications between nodes. Class B is a problem with:

- Numbers of rows: 75000.
- Numbers of iterations: 75.
- Eigenvalue shift: 60.
- Numbers of nonzeros: 13.

5. Results

In order to prove the method designed we expose a series of experiments, in which the existing latency of sent messages between nodes is one of the most relevant data which confirms that MCM can obtain good results in the management of MPI communications.

In each experiments for N virtual machines, we need N iterations of evaluation in order to get all pair-to-pair

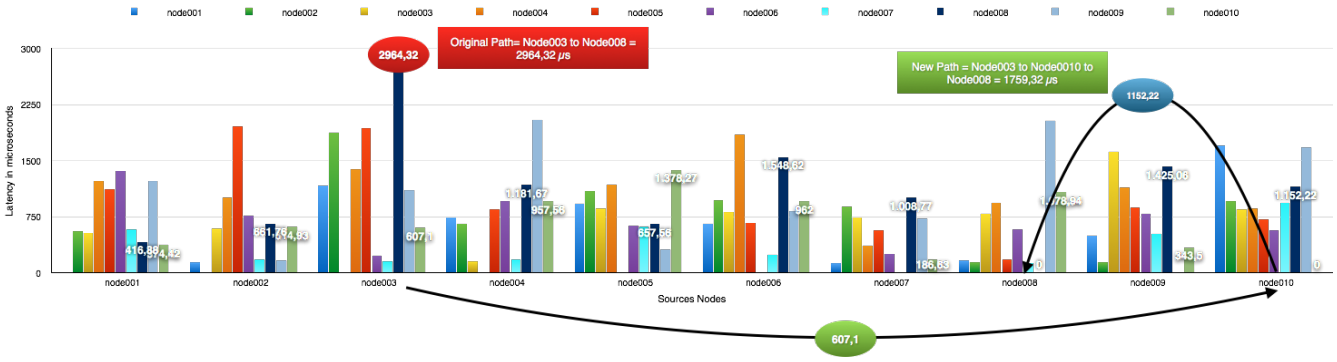


Fig. 5: Latency of Communication Message Size = 1MB in day 1

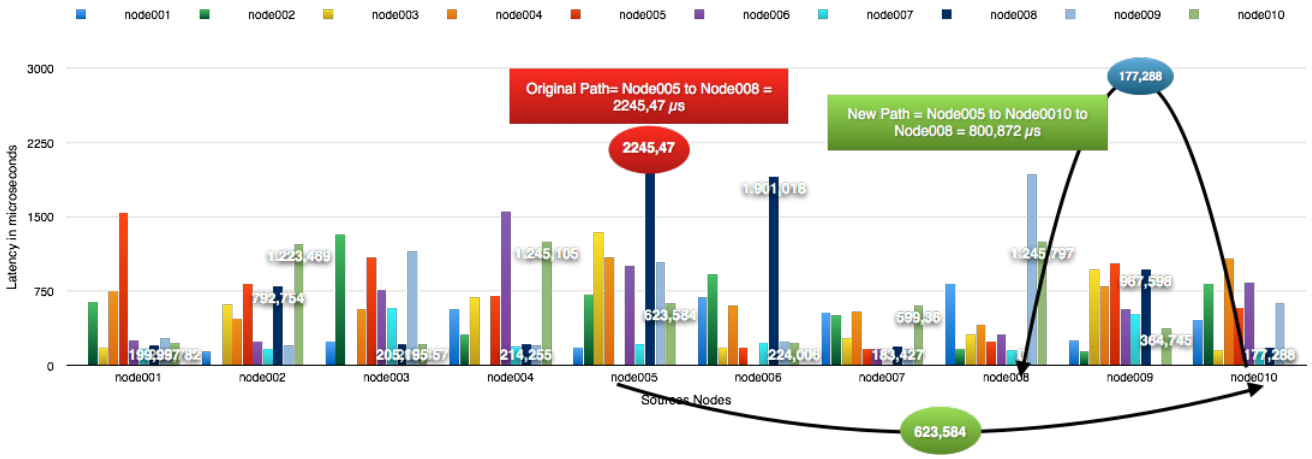


Fig. 6: Latency of Communication Message Size = 1MB day 2

performance. In each iteration, $\frac{N}{2}$ pairs are measured with MPI Send in both directions. When the number of instances is 20, the total evaluation overhead is usually smaller than 30 seconds.

For all of our experiments the hardware and software configuration is the same, we evaluate the impact of messages sizes for our technique and the selection of news path for communications between process of NAS-CG parallel benchmark.

In order to catch system variability, we run several experiments on different days and message sizes. Initially, we evaluate the latency with a message of 1 MegaByte. The result of the latency between nodes is showed in Fig. 5, in this graphic we can see that sending a message of 1 mega from node 3 to node 8 has a latency of more than 2500 microseconds, but this is not the only way to perform this communication because in this type of cluster we find that all nodes are interconnected. MCM proposes the selection of path with low latency, i.e.: instead of sending directly to the node 8, we can select the node 5 to send the message through it to node 8, and the sum of latency from node 3 to node 10 and then to node 8 will be approximate 50% lowest than through the first path. Due to space limitations, not all the communication between nodes are exposed.

Second we run the same experiment every 10 minutes for 1 hour and the results of one of it are in Fig 6. In this figure we can detect communications that can have lowest latency like the previous experiment, but also the variability of the network in cloud. We can observed with this two figures that according to an specific period the latency time of messages can be balanced or unbalanced.

Also we measure the latency with messages of 64KB to prove the behavior with smaller message. The results are showed in the Fig. 7. In this scenario, latency is maintained between 125 and 225 microseconds, for this reason, our technique have to avoid the selection of new path, because it will add more latency time.

Another type of experiments that we performed is with NAS-CG parallel benchmark Class B and the ping pong message calibration between every pair of virtual machine, running both in parallel, in sixteen process. The results are the following and are shown in Fig. 9 and Fig 8.

- For Message Communication in the NAS-CG Class B, between node 0 to node 1, we measure a total latency of

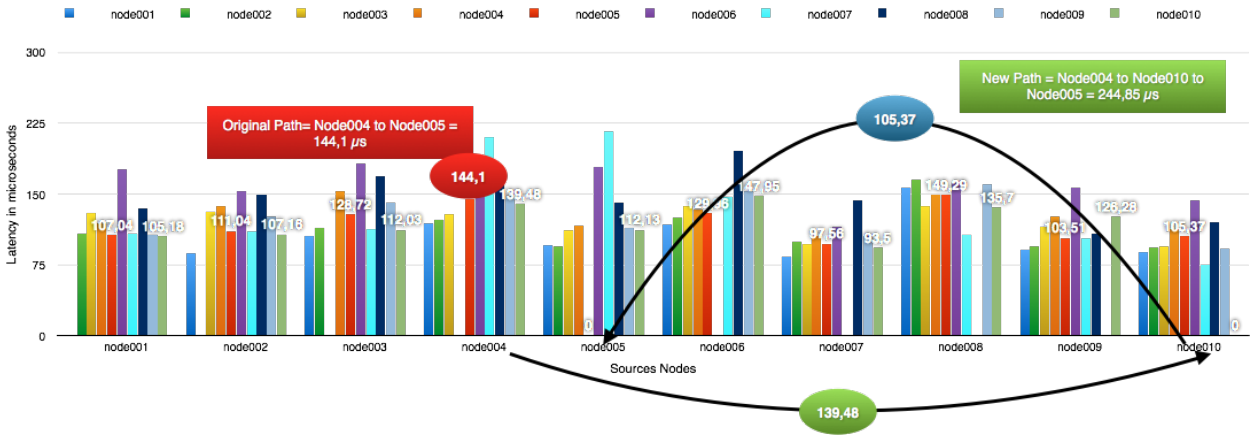


Fig. 7: Latency of Communication Message Size = 64KB

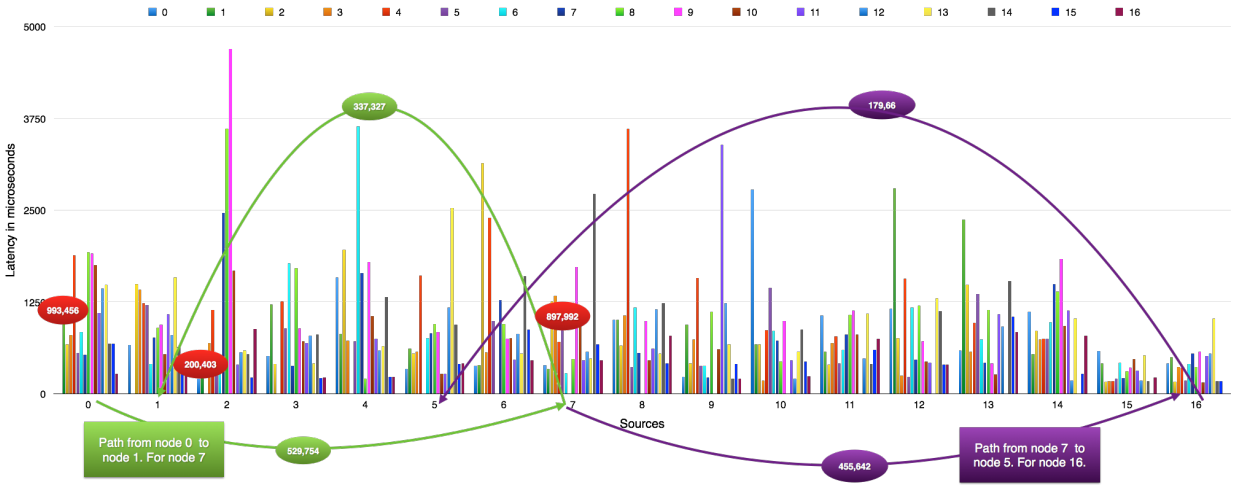


Fig. 8: Latency of Communication Message Size = 1 MegaByte in parallel with NAS-CG

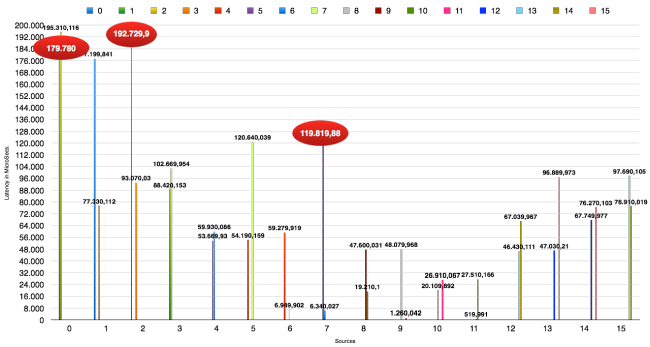


Fig. 9: Latency of NAS-CG Communication in parallel with Message Ping Pong between pairs of nodes

179.782 microseconds for the message size predefined for the application.

We also execute the ping pong calibration of communications between all the virtual machines at the same time, sending message of 1 MegaByte. In this scheme from node 0 to node 1 we measure a latency of 993,456 microseconds with the ping pong calibration. If we change the route, for example node 0 to node 7 to node 1 the latency will be 867,081 microseconds.

This means that if we use the same technique with the message sent by the NAS between the same nodes the total latency of its message will be decreased around 12,7%, because both applications are running at the same time and the virtual network conditions its equals for both of each.

- For Message Communication in the NAS-CG Class B, between node 7 to node 5 we measure a latency of 119.819,88. Like the first one we also have the ping pong calibrations measures, sending message of 1 MegaByte between all pairs of processes.
From node 7 to node 5 we measure a latency of 897,992 microseconds. If we change the route, for example node

7 to node 16 to node 5 the latency will be 635,302 microseconds. In this case our method also will be working with the NAS message between those nodes, the latency will be decreased around 29,26%.

- For Message Communication in the NAS-CG Class B, between node 2 to node 0 we measure a total latency of 192.729,95 microseconds. And then finally send message of 1 MegaByte from node 2 to node 0 have a latency of 200,403 microseconds with the ping pong calibration. In this case we do not have to change the path of the message sent, because there are not betters routes analyzing the behaviour of the network with message of 1 MegaByte.

With all the experiments we realize that our technique can work, decreasing the traffic congestion and the latency, but also can properly add overhead specially in the application runtime.

6. Conclusions

Cloud computing is a new paradigm that has been created as a dynamic model for distributed computing, with this type of environment the goal is to avoid the grid architecture problems. But over the years still have not been able to get the same results. For example in the world of HPC are many challenges.

The main areas of current study on HPC Cloud is based on improving the performance of the MPI communications offering better routing messages through packet routing in a virtual network or modifying MPI sentences [12].

We notice that one of the principals problems of running MPI application in cloud is that network communication does not have efficient support in virtualized environments.

For this reason we propose MCM, this technique search alternative path under hot spot situation, considering the variability of cloud networks and the dynamic traffic behavior. MCM try to manage the message communication between process and with this inquiry betters times of latencies.

The future work is to implement this technique and prove is with benchmark applications and scientific applications, to observe the real behaviour in publics cloud like Amazon EC2. Also we want to investigate how to obtained information about the actual state of the network, the system overhead and the determination of thresholds.

7. Acknowledgment

This research has been supported by the MINECO (MICINN) Spain.

Also, we would like to thank to Sebastian Badia, from Enovance project of Red Hat Technologies, for providing us BWLAT, a tool that measures the latency and message flow.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing recommendations of the national institute of standards and technology."

[2] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 54–62, 7 2013.

[3] K. Dashdavaa, S. Date, H. Yamanaka, and E. Kawai, "Architecture of a high-speed mpi bcast leveraging software-defined network," *2013 IEEE Symposium on Computers and Communications (ISCC)*, pp. 885–894, 2014.

[4] C. N. Castillo, D. Lugones, D. Franco, and E. Luque, "Predictive and distributed routing balancing on high-speed cluster networks," *2011 23rd International Symposium on Computer Architecture and High Performance Computing*, vol. 1, no. 1, pp. 72–79, 10 2011.

[5] Y. Gong, B. He, and J. Zhong, "An overview of cmpi : Network performance aware mpi in the cloud," *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming ACM SIGPLAN Notices - PPOPP '12*, vol. 47, no. 8, pp. 297–298, 2012.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "A view of cloud computing," *Magazine Communications of the ACM*, vol. 53, no. 4, 2010.

[7] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, "Performance analysis of hpc applications in the cloud," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 218–229, 1 2013.

[8] E. Walker, "benchmarking amazon ec2 for high-performance scientific computing," *Program*, pp. 18–23, 2008.

[9] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running hpc applications in public clouds - p395-he.pdf," 2010.

[10] D. Lugones, D. Franco, E. Luque, and N. Carlos, "Predictive and distributed routing balancing , an application-aware approach," *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, vol. 00, 2011.

[11] V. Mauch, M. Kunze, and M. Hillenbrand, "High performance cloud computing," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1408–1416, 2013.

[12] K. Takahashi, D. Khureltulga, Y. Watashiba, Y. Kido, S. Date, and S. Shimojo, "Performance evaluation of sdn-enhanced mpi allreduce on a cluster system with fat-tree interconnect," *2014 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 784–792, 7 2014.

[13] Y. Gong, B. He, and J. Zhong, "Network performance aware mpi collective communication operations in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2013.

Intra-cloud and inter-cloud Load balancing based on interaction between mobile agent and web service

Abir KHALDI[1], Kamel KAROUI[1], Henda BEN GHEZALA[1]
[1] RIADI Laboratory ENSI, University of Manouba, Manouba, Tunisia

Abstract- *Cloud computing is becoming the most important model to provide services to clients through internet. So to attract more customers, cloud providers should ensure a high quality service essentially highly available and efficient. Load balancing is one of the most relevant techniques used to increase service availability. This technique can be used in cloud environment to prove its added value. In this paper, a load balancing metric is defined to select a cloud service. So, we will propose a framework based on the triangulation of mobile agents, web service and load balancing.*

Keywords: cloud, availability, load balancing, mobile agent, web service, security

1. Introduction .

In clouds, the availability is one of the most critical requirements that cloud providers should ensure. As a technique, load balancing is used across different data centers to ensure network and service availability. Thus, computer hardware and software failures are kept to a minimum.

In this work, we will focus on cloud service high availability using load balancing. A dynamic load balancing algorithm based on interaction between mobile agents and web service will be proposed and applied within intra-cloud and inter-clouds.

This paper is organized as follows: section 2 introduces a literature review. Section 3 presents the related work. In section 4, we propose a load balancing framework to increase the cloud service high availability. Section 5, load balancing will be expanded to cover inter-clouds architecture. Section 6 is a case study. The proposed framework is evaluated in section 7. Finally, section 8 concludes and recommends future trends.

2. Literature view

2.1 Cloud computing

NIST [1] defines Cloud computing as a "model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and delivered with minimal managerial effort or service provider interaction" .

The essential characteristics of cloud computing are [1]: On-demand self-service , Broad network access, Resource Pooling, Rapid elasticity, Measured service.

The service models of cloud computing are [1]:

- **Software as a Service** (SaaS) : The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure.
- **Platform as a Service** (PaaS) : The capability provided to the consumer is to deploy onto the cloud infrastructure consumer - created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.
- **Infrastructure as a Service** (IaaS) :The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

The deployment models of cloud computing are [1]:

- **Public cloud** : The cloud infrastructure is provisioned for open use by the general public.
- **Private cloud**: The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers.
- **Hybrid cloud**: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (public, private, or community).
- **Community cloud** : The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns.

2.2 Mobile agent

Mobile Agent (MA) is a programming paradigm used in distributed applications [2]. It makes the implementation of applications dynamically adaptable easier and facilitates the development of distributed applications on large networks. This covers many domains such as e-commerce; telecommunications, workflow applications, remote maintenance and park administration [3].

Mobile agents are execution programs that can migrate from one host in a network to another in order to satisfy requests made by their clients. The state of the running

program is saved, transported to the new host and restored, allowing the program to continue where it left off.

2.3 Web service

A Web Service is a method of communication between two electronic devices over a network. It is a software function provided at a network address over the web with the service always on, as in the concept of utility computing. The W3C defines a Web service generally as a software system designed to support interoperable machine-to-machine interaction over a network.

The W3C Web Services Architecture Working Group defined a Web Services Architecture, requiring a specific implementation of a "web service." In this: "a web service has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP (Simple Object Access Protocol) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards"[4].

2.4 Load balancing

Load balancing is a relatively new technique that facilitates networks and resources by providing a maximum throughput with minimum response time [5]. Dividing the traffic between servers, data can be sent and received without major delay. Different kinds of algorithms are available to help traffic getting loaded between available servers. A basic example of load balancing in our daily life can be related to websites. Without load balancing, users could experience delays, timeouts and possible long system responses.

Load balancing solutions usually apply redundant servers which help a better distribution of the communication traffic so that the website availability is conclusively settled.

There are many different kinds of load balancing algorithms available, which can be categorized mainly into two groups. The following section will discuss these two main categories of load balancing algorithms:

- ***Static algorithms*** divide the traffic equivalently between servers. By this approach, the traffic on the servers will be disdained easily and consequently it will make the situation more imperfect. This algorithm, which divides the traffic equally, is announced as round robin algorithm. However, there were lots of problems reported within this algorithm. Therefore, weighted round robin was defined to improve the critical challenges associated with round robin. In this algorithm each server has been assigned a weight, accordingly, the highest weight will receive more connections. In the situation that all the weights are equal, servers will receive balanced traffic [5].
- ***Dynamic algorithms*** will attribute proper weights to servers and by searching through the whole network, the server with the minimum weight will be chosen in order to balance the traffic on. However, selecting the most adequate server will need real time communication with the network, which will lead to an extra added traffic on the system.

Comparing the two algorithms [5], dynamic algorithm could predict queries that can be made frequently on servers, but sometimes prevailed traffic will prevent these queries to be answered, and correspondingly more added overhead can be distinguished on network.

3. Related Work

Availability is a critical requirement in cloud computing services. Many studies attempt to balance high availability (HA), based on cloud system performance and cost.

Jung [6] studied a replication technique to guarantee HA while maximizing performance on a certain number of resources. Replication of software components is used to provide HA. In case of hardware failure, they used component redundancy and regenerated the software components into the remaining resources to achieve HA and optimize performance. It is based on a queuing model with different "mean time between failure" (MTBF) and "mean time to repair" (MTTR).

In [7], Thanakornworakij and al proposed a HA-OSCAR open source framework to increase availability.it aims to improve HA of any Linux-based cloud computing platform. they enhance the load balancing between just two servers.

In [8], Chaczko and al use Message oriented architecture to ensure load balancing in distributed networks. Based on messaging techniques XMPP allowed resources to be monitored and provide availability of cloud resources.

In [9], Hemant and al proposed a prototype system based on a governance body which will handle all the transactions from the user to the actual server from which the user is requesting. They introduced routing table at each end server and middle server (Governance server).

In the table bellow, we discuss the different works in load balancing in the cloud based on 5 criterion: cloud architecture, load balancing algorithm, using a middleware, service type, hypervisor metric and security metric.

TABLE 1. COMPARING PROPERTIES OF PREVIOUS RELATED WORK

Related work	Cloud Architecture	Load balancing algorithm	Using a middelware	Service type	Hypervisor's metric	Security's metric
[6]	Public/ Private	Dynamic	NO	Any	**No**	**No**
[7]	Public/ Private	Static	Yes	Any	**No**	**No**
[8]	Distributed	Dynamic	Yes	Data base	**No**	**No**
[9]	Public	Dynamic	Yes	Any	**No**	**No**

In our proposed framework, we will try to apply a dynamic, operating system independent and secure load balancing algorithm to cover all clouds models.

4. Proposed framework for load balancing intra-cloud

In this section, we will describe our framework aiming to improve the cloud service performance. As follow, we will divide the framework description into 4 parts :

4.1 Global Framework architecture

In [10], we proposed a secure cloud architecture design based on 4 zones and we specified a DMZ zone to deploy different cloud servers. In fact, we didn't implemented in this previous work [10] a method to ensure the high availability. We will continue in this work to increase the availability of each server in the DMZ zone. So, we will propose a cloud DMZ zone managed by a master virtual machine named DMZManager. The DMZManager will be a middleware between cloud customer and cloud service.

So, the DMZManager as a cloud middleware will receive the customer requests and automatically forward them to the appropriate server in order to fulfill the request (figure 1).

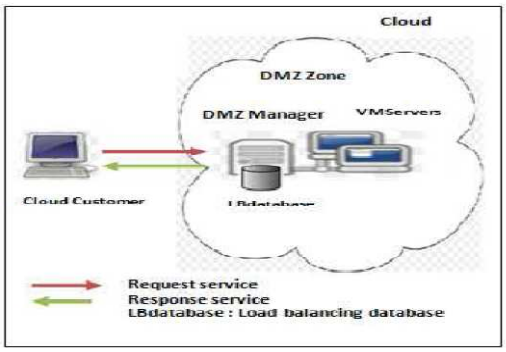


Figure 1. Global Load balancing Framework

4.2 Framework components

The proposed framework ensures the communication between 6 principal components which are responsible of load balancing through a dynamic algorithm. These components are as follows:

- ***DMZManager*** : It is a virtual machine (VM) located in the DMZ Zone as a middleware between cloud customer and cloud service. It selects the best efficient service using a proposed dynamic load balancing technique (section 4.3).
- ***VM servers*** : VM deploying public or private service such as web server, FTP server, Voip server, etc.
- ***Mobile agent platform*** : It is a platform based on intercommunication between 3 agents :
  - **Static Manager Agent (SMA)** : is located on the DMZManager to select the efficient service for the customer.
  - **Static Server Agent (SSA)** : is located on the VM server to receive mobile agent and to ensure secure communication.
  - **Mobile Agent (MA):** The SMA dispatches a MA to each server in order to apply the proposed load balancing technique (section 4.3). The SSA receives the MA to do its job if it is authenticated.
- ***Web service:*** when invocated by the MA, it responds by the requested information. It is the middleware between the MA and the local VM server resources (CPU, RAM, database,etc).
- ***Data base :*** it is located in the DMZManager containing those attributes :
  - **ServerType*:*** It is the service type offered by the server (web, ftp, voip, etc).
  - **Ip_address**: The ip address of the VM server.
  - **LB**(S_i).: the calculated load balancing metric of the service i.
- ***Intrusion detection system :*** It is a host intrusion detection system (HIDS) used to measure the number of intrusions detected in a second.

Those components will intercommunicate to ensure load balancing within the proposed technique following some specific metrics.

4.3 Load balancing technique proposed

We propose applying a dynamic load balancing technique based on an indicator for each service named LB(S_i). The LB(S_i) is calculated using those following metrics related to each service in a VM server:

- ***CPU***: The CPU usage for a specific service in a VM server.
- ***RAM*** : The memory usage for a specific service in a VM server.
- ***Number of incoming requests per second (NRIn)***: It is the number of requests for a specific service received by a VM server per second .
- ***Number of outgoing requests per second (NROut)*** : It is the number of request issued from a specific service in a VM server per second.
- ***Number of intrusions per second (NIntru)***: It is the number of intrusions aiming a specific service detected per second in a VM server. It is considered as a security metric to choose a secure service.

The proposed dynamic load balancing algorithm is based on 3 steps:

- ***Step1* : *Transforming metric value into a binary metric***

Metric values are so heterogeneous which complicates the calculation of the LB(Si). Our idea is to transform each metric value into a binary value composed of 2 bits.

This binary value will be reflecting the metric value importance. Each metric belongs to one of the 4 metric's classes.

- ***Low class***: it represents the class of the very high metric value. For example, when the value of one of those metrics (CPU percentage, RAM percentage, NRIn, NROut, NIntru) is very high so it belongs to the low class. The low class is represented by a word composed of 2 bits equals to 11.
- ***Medium class***: It represents the class of the medium metric value. For example, when the value of one of those metrics (CPU percentage, RAM percentage, NRIn, NROut, NIntru) is medium so it belongs to the medium class. The medium class is represented by a word of 2 bits equals to 10.
- ***Good class***: it represents the class of the medium metric value. For example, when the value of one of those metrics (CPU percentage, RAM percentage, NRIn, NROut, NIntru) is low so it belongs to the good class. The good class is represented by a word of 2 bits equals to 01.
- ***Excellent class***: it represents the class of the very low metric value. For example, when the value of one of those metrics (CPU percentage, RAM percentage, NRIn, NROut, Nintru) is very low so it belongs to the excellent class. The excellent class is represented by a binary word equal to 00.

We resume the transforming of a metric value in a binary word in table 2 using CPU metric as an example.

TABLE 1. CPU METRIC VALUE TRANSFORMATION AND CLASSIFICATION

CPU (%)	Low class	Medium Class	Good class	Excellent class
Metric value	[100%,50%]	[49%,30%]	[29%,10%]	[9%,1%]
Binary word	11	10	01	00

- ***Step2: Calculating LB(Si)***

The LB(Si) is calculated by using concatenation based on priority. This method consists to gather all metrics binary words in a unique sequence.

The sequence is composed on its first part of the metric of the strongest priority (figure 1). We propose that every metric has a priority P_i which helps to constitute the LB(Si). In our case, we have 5 metrics (CPU percentage, RAM percentage, NRIn, NROut, Nintru) so we define 5 priority level from 0 to 4 when 4 is the lowest priority value and 0 is the highest.

Metric's priority is set by the cloud provider based on cloud environment and its variables. Sometimes the most efficient VMserver for the cloud provider is the server using the minimum percentage of CPU so the lowest priority value (P=0) is for CPU metric. For that, we sort the metrics according to their priority to get the LB(Si) (figure 2).

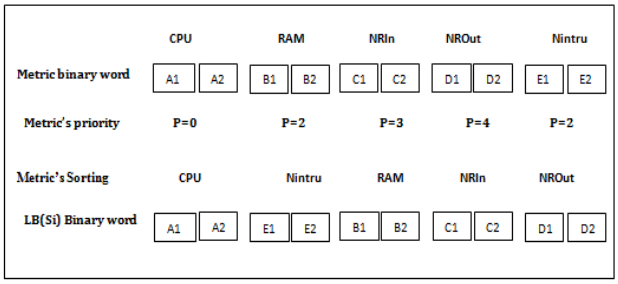


Figure 2. LB(Si) calculated using metric's priority

The LB(Si) is composed of 10 bits. It reveals the importance of all the metrics value due to their priority. So LB(Si) value is in [0,1024].

For example : In VM server, for a specific service (Si) , we have CPU=00, RAM=01, NRIn = 01 , NROut = 01 and NRIntru=10. So according to their priority (figure 3), we get LB(Si)=0010010101

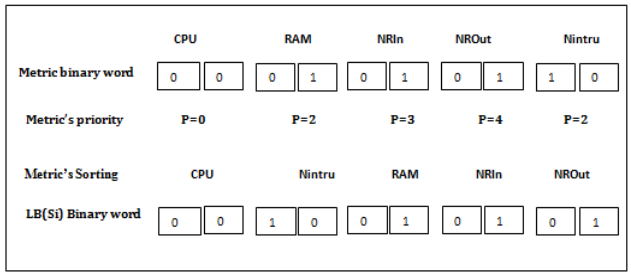


Figure 3. Example of LB(Si) calculation

The MA takes charge of step 1 and step 2 of the proposed dynamic load balancing technique.

- ***Step3: Classifying LB(Si)***

After calculating LB(Si), the SMA can classify LB(Si) into 4 class (see table 3):

- When $0 \leq LB(Si) \leq 255$, it belongs to the Excellent class due to its low value. In this case the VM service is highly available.
- When $256 \leq LB(Si) \leq 511$ it belongs to the good class and the VM service is well available.
- When $512 \leq LB(Si) \leq 767$, it belongs to the medium class and the VM service is moderately available.
- When $768 \leq LB(Si) \leq 1024$, it belongs to the low class and the VM service is not enough available.

TABLE 2. LB(SI) CLASSIFICATION OF METRIC'S VALUES

	Low class	Medium Class	Good class	Excellent class
LB(Si) value	[768,1024]	[512,767]	[256,511]	[0,255]

- ***Step4: Choosing LB(Si)***

The LB(Si) classification helps SMA to make the best decision for choosing the most available VM service. This method is very useful mainly when applying load balancing between more than two service. It is important to forward the customer request to the most available VMserver having the lowest LB(Si) value.

4.4 Framework function

We will describe the different steps followed to apply the cloud high availability load balancing algorithm (Figure 4):

- ***Step1 :*** The cloud customer request a service from the DMZManager. The SMA receives the request and searches in its data base the different ip addresses of the appropriate servers which can offer the service.
- ***Step2:*** The SMA dispatches a MA to each server using its ip address. The MA migrates to the server through the cloud network.
- ***Step3:*** The SSA receives the MA. The SSA asked the MA a password in order to accept it. If the MA is correctly authenticated it can continue its work.
- ***Step4:*** The MA invokes the web service to collect the following metrics : NRIn, NROut, NIntru, CPU.
- ***Step5:*** The web service analyses the HIDS log to determine NRIn, NROut, NIntru and demands to the system the CPU used.

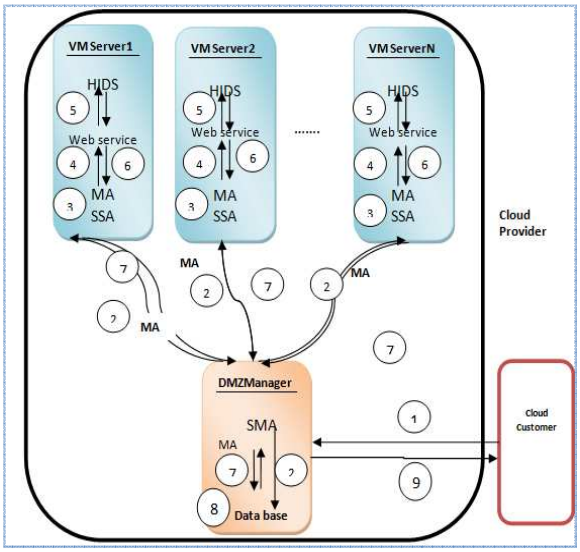


Figure 4. Load balancing Framework Function

- ***Step6:*** The web service responses the MA. The MA calculate the LBi.
- ***Step7:*** The MA calculates the LB(Si) as described in step1 and step 2 in section 4.3 and returns back to the DMZManager. The MA is received by the SMA to gives it the LB(Si) calculated. The SMA saves in the database the LB(Si) for the appropriate service using its ip address.
- ***Step8:*** The SMA compares the different appropriate LB(Si) in the data base and selects the ip address for the server having the best LB(Si).
- ***Step9:*** The SMA forward the customer request to the service selected and gives him the response.

In fact, the MA migration to calculate different LB(Si) isn't happening each time the customer requests a service because it will consume much more time to response. So, dispatch is done periodically to fulfill the data base in the DMZManager. Thus, when a customer requests a service, the SMA consults its data base to select the efficient service at that time.

5. Proposed framework for load balancing inter-cloud

In the previous section, we have proposed a framework for intra-cloud load balancing whether it is a public cloud or a private cloud.

In the following section we're going to explain how our load balancing solution get expanded to ensure inter-clouds load balancing.

The solution is to ensure an intercommunication between different SMA in DMZManager in each cloud. Every cloud has a metric related to the performance of its hypervisor. For that, it is important to introduce the metric of the cloud hypervisor (Cj) named H(Cj).

We propose an hypervisor's metric H(Cj) based on those metrics:

- *CPU*: The usage of CPU by the Cj hypervisor.
- *RAM* : The usage of RAM by the Cj hypervisor.
- *Number of intrusion/second(NIntru):* It is the number of intrusions aiming a specific hypervisor detected per second.

Those metrics are represented as a word composed of 2 bits like metric's representation in section 4.3 .

So the H(Cj) is calculated by the DMZAgent deployed in the DMZManager using the concatenation based on metric's priority defined by the cloud provider (figure 5).

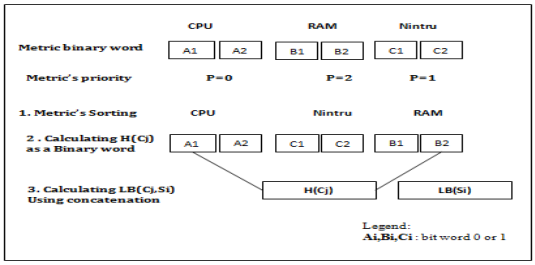


Figure 5. Calculating Load balancing metric inter-cloud

The LB(Cj,Si) is represented in 16 bits (See table 4) and can be classified into 4 class as we do for LB(Si) .

- When $0 \leq LB(Cj,Si) \leq 16383$, it belongs to the Excellent class due to its low value. In this case the VM service is highly available.
- When $16384 \leq LB(Cj,Si) \leq 32767$ it belongs to the good class and the VM service is good available.
- When $32768 \leq LB(Cj,Si) \leq 49151$, it belongs to the medium class and the VM service is moderately available.
- When $49152 \leq LB(Cj,Si) \leq 65536$, it belongs to the low class and the VM service is not enough available.

TABLE 3. LB(CJ,SI) CLASSIFICATION OF METRIC'S VALUES

	Low class	Medium class	Good class	Excellent class
LB(Cj,Si) value	[49152, 65536]	[32768, 49151]	[16384, 32767]	[0,16383]

The LB(Cj,Si) is stored in the appropriate LBdatabase's table in each DMZManager.

The DMZAgent communicates the lowest value of LB(Cj,Si) to another cloud DMZAgent for choosing the best performing VM service.

The different cloud's DMZManagers communicate through their LBDBAgents.

The new metric of the cloud load balancing LB(Cj,Si) for each server is the concatenation between the H(Cj) and the LB(Si). The H(Cj) will constitute the first and the most significant part of the LB(Cj,Si) (Figure 5).

6. Case study : Apache load balancing

In this study, we focus on load balancing intra-cloud. We've chosen proxmox [**11**] as a cloud hypervisor because it supports Graphical User Interface (GUI) so that the installation and configuration becomes easier than other platforms using the command Line Interface (CLI). Then, we deploy 3 virtual machines on proxmox hypervisor. In each one, we deploy an Apache web server.

By default the three apache web server had the same LB(Si) metrics.

Bee-Gent Mobile Agent has been used for implementation. Bee-Gent technology was first released in 1999 by Toshiba [12], as a new type of pure agent development framework for the advanced network society. Its communication framework is based on the multi-agents model. The Bee-gent framework is comprised of two types of agents: agent wrappers and mediation agents:

- Agent Wrappers are used to 'agentify' existing applications. The agent wrappers manage the states of the applications, which are wrapped around, and invoke the applications when necessary.
- Mediation Agents support inter-application co-ordination by handling all communications among applications. The mediation agents move from the site of an application to another where they interact with the remote agent wrappers.

For the IDS, we deployed SNORT[13] in each VM to monitor the system and the network intrusions.

We configured snort to save alerts in its mysql database to deal with analyzed phase by the web service.

We tested our proposed load balancing framework through two test cases (table 5):

TABLE 5. LOAD BALANCING INTRA-CLOUD TESTCASES

		Initial Condition	Testcase 1	Testcase 2
Apache1 192.168.233.152	**CPU**	10%	**12%**	**12%**
	RAM	3%	3%	3%
	NRint	1000	1000	1000
	NRout	1000	1000	1000
	NIntru	0	0	**0**
Apache2 192.168.233.153	**CPU**	10%	**10%**	**10%**
	RAM	3%	3%	3%
	NRint	1000	1000	1000
	NRout	1000	1000	1000
	NIntru	0	**0**	**1**
Apache3 192.168.233.154	**CPU**	10%	**15%**	**15%**
	RAM	3%	3%	3%
	NRint	10	1000	1000
	NRout	10	1000	1000
	NIntru	0	0	**0**
Selected Server		**Apache1**	**Apache 2**	**Apache 1**

- ***Testcase 1***: One of the three apache web server consumes less CPU than the others. So the traffic will be redirected to this server. (see figure 6).

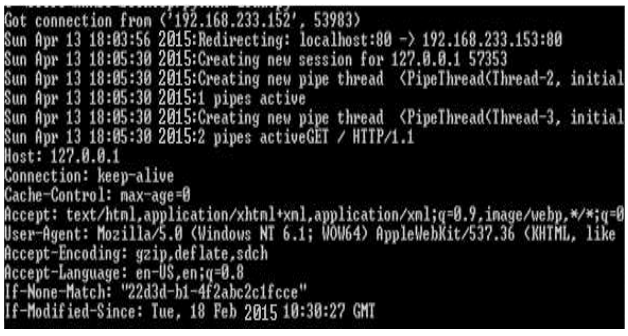


Figure 6. Traffic redirection for load balancing

- ***Testcase 2***: One of the three apache web server used less CPU but it is attacked by a DOS attack (see figure 6). So the traffic will be redirected to the server using less CPU.

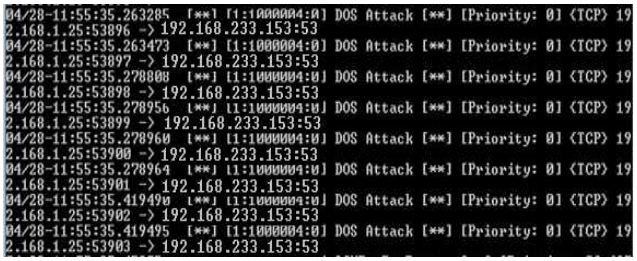


Figure 7. DOS attack to Apache server

7. Proposed Load balancing Framework Evaluation

The most important advantages of our proposed load balancing framework are :

- We benefit from the advantages of the mobile agent in a previous work to detect and repair intrusions in an hybrid cloud [14]. So in this work, the load balancing with Mobile Agent approach uses also less network load compared to the client/server approach, by shipping code to data instead of shipping data to code.
- The Bee-Gent mobile agent approach offers an important feature: it is an authenticated and encrypted agent intercommunication:
- Security is an integrated part of the load balancing indicator proposed which is not done in previous works. The proposed load balancing algorithm chooses the most secure service given the use of the Number of intrusion detected per second as a metric in the LB indicator.
- The hypervisor metric H(Cj) is a new metric proposed to select the most efficient hypervisor inter-cloud.

8. Conclusion

The high availability is an important factor for Cloud service providers to ensure service quality. Our proposed load balancing framework using mobile agents and web service interaction aims to improve the cloud service availability intra-cloud and inter-cloud.

The load balancing algorithm chooses the most efficient and secure service to fulfill customers request. This algorithm cuts down costs in terms of network load, enhances balancing execution and secures communication due to the mobile agent approach.

We plan to explore additional ways to expand the cloud service availability, robustness and reliability.

9. References

[1] Mell, P. &Grance, T., 2011, "The NIST Definition of Cloud Computing", NIST Special Publication 800-145 (Draft). Retrieved 2013-10-11)

[2] Lange, D. and Oshima, M.,1999. Seven Good Reasons for Mobile Agents - Dispatch your agents; shut off your machine. Communications of the ACM Issue.

[3] Guttman, R. et al., 1998. Agent-mediated electronic commerce: a survey. Knowledge Engineering Review. 13(2):143-147.

[4] "Web Services Glossary". W3C. February 11, 2004. Retrieved 2015-03-22.

[5] R. Shimonski. *Windows 2000 & Windows Server 2003 Clustering and Load Balancing. Emeryville.* McGraw-Hill Professional Publishing, CA, USA (2003), p 2, 2003.

[6] Jung, G., Joshi, K.R., Hiltunen, M.A.: Performance and Availability Aware Regeneration for Cloud Based Multitier Application. In: Dependable Systems and Networks (DSN), pp. 497–506 (2010)

[7] Thanakornworakij, T., Sharma, R., Scroggs, B., Greenwood, Z. D., Riteau, P., & Morin, C. (2012, January). High availability on cloud with HA-OSCAR. In *Euro-Par 2011: Parallel Processing Workshops* (pp. 292-301). Springer Berlin Heidelberg.

[8] Chaczko, Z., Mahadevan, V., Aslanzadeh, S., & Mcdermid, C. (2011, September). Availability and load balancing in cloud computing. In *International Conference on Computer and Software Modeling, Singapore* (Vol. 14).

[9] Hemant, P., Chawande, N. P., Sonule, A., & Wani, H. (2011, September). Development of servers in cloud computing to solve issues related to security and backup. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on* (pp. 158-163). IEEE.

[10] Khaldi Abir., Karoui, K., Tanabène, N., & Ghzala, H. B. (2014, April). A secure cloud computing architecture design. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on* (pp. 289-294). IEEE.

[11] Proxmox. (2015). VE. Retrieved 02 13, 2015, from Proxmox Virtual Environment: http://www.proxmox.com/products/proxmox-ve

[12] Bee-Gent,Online: http://flylib.com/books/en/4.4.1.92/1/ (January 2015)

[13] Snort, Online: http://www.snort.org/, (March 2015).

[14] KHALDI Abir, Kamel KAROUI, and Henda BEN GHEZALA. "Framework to detect and repair distributed intrusions based on mobile agent in hybrid cloud."