SESSION

COMMUNICATION SYSTEMS: MOBILE COMPUTING, INTERCONNECTION NETWORKS AND TOPOLOGIES, WIRELESS SYSTEMS

Chair(s)

TBA

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

RM-circuits: Toward Feasible Use of Reconfigurable Mesh Algorithms

Yosi Ben-Asher
CS, University of Haifa

Esti Stein
CS, Tel Aviv-Yaffo Academic College

Vladislav Tartakovsky
CS, University of Haifa

Abstract— *The reconfigurable mesh (RM) is a powerful model for parallel computations that can outperform PRAM computation. Many basic algorithms has been shown to run in constant time on the RM. In spite of this power, the RM has not been realized mainly due to the theoretical assumption of constant time broadcasting, while practically its a function of the number of switches the broadcast has to pass through. We introduce the restricted-RM (RRM) model, wherein buses use mostly $d(n) = n^{1/k}$ switches. We show that counting the number of 1's in an n-bits input, can be done on the RRM in $2 \cdot k$ steps for $k = 2, 3, \ldots$. An almost matching lower bound is presented, showing that the RRM cannot compute counting of n variables in less than k steps. Finally, the algorithm was directly coded in Verilog outperforming a regular optimal parallel adders-circuit. This work thus present a practical version of the RM which is directly coded as a hardware-circuit showing not only that RM-algorithms are practical but also a simple way to program them.*

1. Introduction

One of the most interesting models in the field of parallel computations is the *reconfigurable mesh (RM)*. The RM consists of a mesh, augmented by the addition of a dynamic bus system, whose configuration changes in response to computation and communication needs. More precisely, a RM of size $N \times N$ consists of N^2 identical processing elements ($PE_{i,j}$) as described in figure 1. Each $PE_{i,j}$ is connected to its four neighbors $PE_{i-1,j}$, $PE_{i+1,j}$, $PE_{i,j-1}$ and $PE_{i,j+1}$ provided they exist, and has four ports denoted by $'N', 'S', 'E'$ and $'W'$. Local connections within the PE can be dynamically changed at each step of the RM using on-off switches (figure 1 depicts 15 possible reconfiguration states). This yields a variety of possible bus topologies for the mesh, where each connected component is viewed as a single bus. In every broadcast step, each PE receives incoming signals, executes local computations, chooses a new configuration and broadcasts on some of its S, N, E, W edges (ports).

Counting the number of bits set to $'1'$ in an input vector, is a fundamental operation of the RM. It is used in almost every RM algorithm, and is performed in many variations. Consequently, when introducing a new RM model, it is important to study its complexity in this model. Figure 2 depicts how counting is applied to a 4-bits input using a 4×4 RM. Based

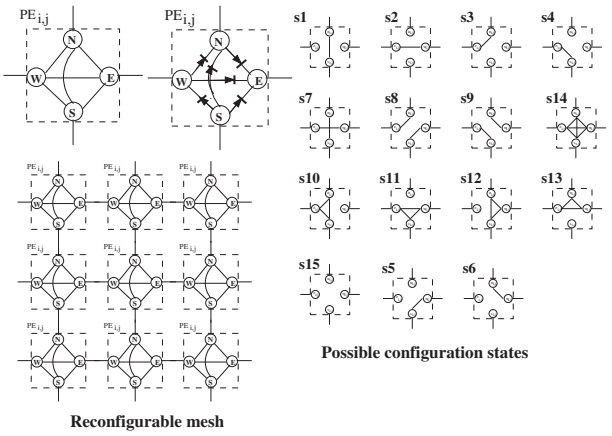


Fig. 1: *Reconfigurable mesh and all* 15 *possible states*

on the i'th input bit, each switch in column-i of the RM choose either to perform a "band" $< W \Longrightarrow N,\ S \Longrightarrow E >$ or a "horizontal-pass" $< W \Longrightarrow E,\ S,\ N >$. As a result, an incoming signal ($>$ in figure 2, bottom-left) is banded where $INP[i] == 1$, and the output signal comes out through the k'th output iff $\sum_{i=0}^{N} INP[i] == k$. Note that since we are targeting a circuit, we can freely use wires connecting different RM switches, and also input bits. Thus, as shown in figure 2, the task of converting the position $(0, 1, 2, 3, 4)$ of the output signal to a binary number is done via connecting each possible position to its binary representation using $\log 4 + 1$ vertical wires. Counting of N input bits takes one step on $N \times N$ RM, using broadcasts that traverse N switches. It is thus not immediate to find a counting algorithm for the restricted RM using buses of length less than or equal to $d(N)$, maintaining minimal possible number of steps. Moreover, a lower bound for counting on the restricted RM should be devised, so that optimal values of $d(N)$ can be determined.

As such it has been shown that the RM can perform parallel computations faster than boolean circuits or by the PRAM (Parallel Random Access Machine) model [15]. This include $O(1)$ summing [14], [5], $O(1)$ multiplication [12], sorting [13], convex hall [19], graph algorithms [6], [23] and image processing [16]. However, this potential power of the RM could not practically be used, since the RM model assumes that broadcasting a signal along a bus/connected

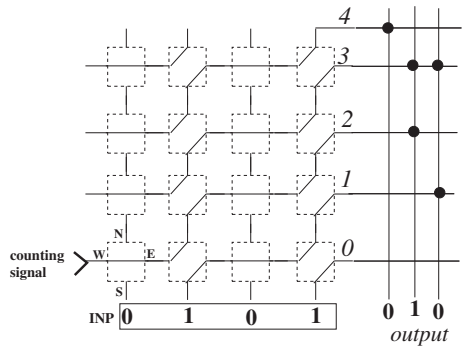


Fig. 2: *Counting on the reconfigurable mesh*

component can be done in one step regardless of the number of switches/ports it passes. This assumption is not feasible using current CMOS transistors since the dynamic reconfigurations inside each PE must be realized by switches connecting/disconnecting its S, N, E, W edges. As switches are made from CMOS transistors, each with a $Resistance \cdot Capacity$ delay for ON/connected-state, broadcasting along a bus composed of n switches, is at least $n \cdot R \cdot C$ which is in contrast to the $O(1)$ assumption. In fact, using the Elmore model [9] to estimate the delay along n switches, yields a quadratic delay of $(n^2)/2 \cdot R \cdot C$. This is due the time required to fill C_i (the capacitance of the i'th switch of the bus), which is proportional to $(i \cdot R)$.

Several types of restricted models such as the RMBM (Reconfigurable Multiple Bus Machine) [22] have been proposed. Since all of them are using n switches on an n length bus, they are not good candidates to solve the above problem. One restricted model that is closer to the proposed restricted RM is the SRGA of [21], [8], where each row/column of the mesh has a complete binary tree of reconfigurable switches, allowing to route messages between the leaves of this tree. However, the SRGA still allow broadcasts that pass through $O(N)$ switches. A more related work is [4] proposing k-constrained RM model that allows buses of wire-size at most k to be formed per cycle. [4] shows that a version of column-sort [18] can sort N items on the k-constrained $k \times N$ RM in $O(N/k)$ steps (and a similar result for convex hull). This implies that sorting on the k-constrained RM has optimal VLSI complexity of $A \cdot T^2$, mainly due to the linear $O(N)$ bus configurations of the sorting algorithm, which can be optimally simulated in N/k steps by the k-constrained RM. We remark that optimal self simulations for the RM with linear bus configurations has been shown in [2], hence any RM algorithm that uses only linear buses can be efficiently simulated by k-constrained RM. This work differs from [4] as we go beyond simulation of larger buses over small buses and ask a different question: what is the minimum number of switches (length) on a bus, forming a restricted RM, that can be a platform for solving a problem running on a non-restricted RM, without increasing time complexity. For example, in order to execute counting of N bits by an $N \times N$ RM in a constant number of steps, one should use buses of length $N^{\frac{1}{k}}$. In comparison, had we just simulated the N-bus of the regular $N \times N$ counting by buses of length $N^{\frac{1}{k}}$ we would have end with $N^{\frac{k-1}{k}}$ steps using [4], rather than the k steps we obtain. In addition, our model focuses on the number of switches a broadcast goes through compared to the length of bus, since the main delay on the RM bus is quadratic in the number of switches.

Thus in this work we study a new model of the RM called the "restricted-RM" (RRM) wherein broadcasting along buses is restricted not to use or pass through more than $d(n)$ switches where n is the input size. For example if $d(n) = n^{\frac{1}{4}}$ then for $n = 10^6$ we get that $d(n) \approx 30$, a number for which the RM $O(1)$ assumption may be feasible, using current transistors technology. Thus, we believe that by restricting broadcast to pass through no more than $d(n)$ switches we can obtain RM circuits that are feasible and outperform their regular gate-based circuits. We study the fundamental problem of counting the number of 1 bits in an input sequence of n variables and show that:

- Counting can be done using the RRM with $d(n) = n^{1/k}$ in $2k$ steps for any constant value of $k = 2, 3, \ldots$.
- An almost matching lower bound showing that a RRM with $d(n) = n^{1/k}$ can not compute counting of n variables in less than $k+1$ steps. Proving lower bounds for reconfigurable algorithm is more difficult than regular lower-bounds, due to the need to bound information that can be obtained by reconfiguration. The technique presented in this work adds to the few existing lower-bound techniques proposed so far (e.g. [1]).
- We show that RM-algorithm can be directly coded in Verilog. This way of programming RM-algorithms overcome most of the drawbacks of the C-like programing style proposed so far for RM-algorithms (e.g.,ARMlang [10]). We thus demonstrate that RM algorithms can be directly synthesized to circuits using hardware description language even for large size of RMs. Previous realizations of the RM were mainly to a small-size grid of Soft-CPUs on the FPGA using MUX-gates for reconfiguration. Clearly this method [11] can not scale well.

2. Counting with the Restricted RM

We first indicate that the restricted RM (RRM) model we consider is a form of a circuit. Therefore, wires and logic/arithmetic sub-circuits can be used freely. We can thus use a set of x subRMs each of size $z \cdot w$ where fix wires can arbitrarily connect between the processing units of these subRMs. We refer to such a circuit of subRMs as a RRM of size $z \times x \cdot w$.

The ability to perform counting on a RRM requires an ability to sum numbers. Therefore, we use a simple RM

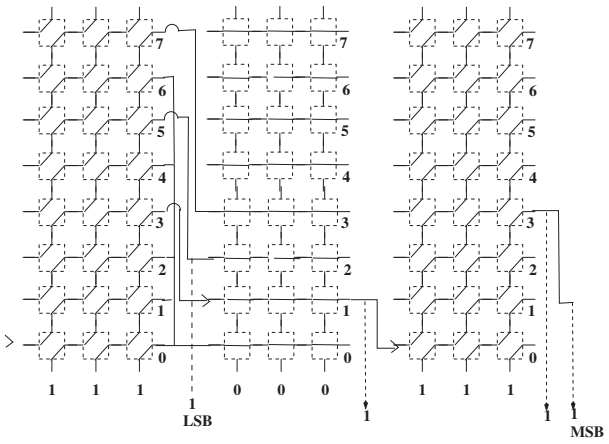


Fig. 3: One step summing

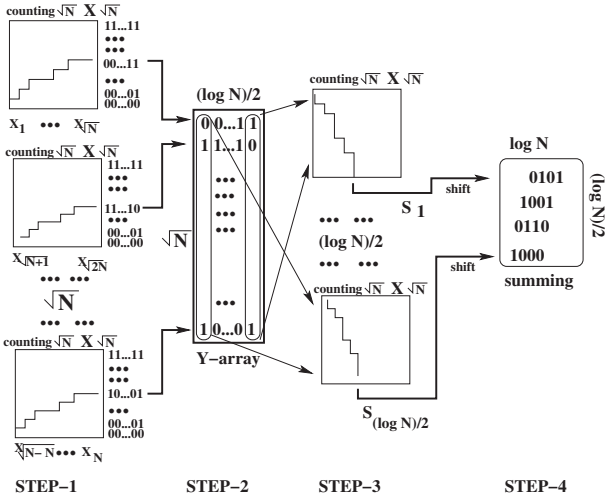


Fig. 4: Restricted counting for $k = 2$, schematic layout of the algorithm/circuit

technique for summing:

Theorem 2.1: Summing of L binary numbers $x_1, \ldots, x_L$ of L input bits each, can be done in one step by a $2 \cdot L \times L^2$ RRM using bus lengths of at most L^2 switches.

This can be done by concatenating L RMs, each of size $2 \cdot L \times L$ $(RM_1, \ldots, RM_L)$ such that each RM_i computes counting of the i'th bits of each number, $y_i = counting_{input\_signal, RM_i}(x_1[i], \ldots, x_L[i])$. The position of the output signal of RM_i is converted to an output bit $y_i = i \ mod \ 2$ such that $y_L, \ldots, y_1$ form the final sum. The input_signal of RM_{i+1} is either:

- fed to $RM_{i+1}[0][0]$ if the output position of RM_i is zero or
- fed to $RM_{i+1}[0][k]$ if the output position of RM_i is $2 \cdot k$ (by "banding" the output bus).

The validity of this construction follows from the fact that each 1-value at column-i represents 2^i in the final sum, thus the carry from column $i-1$ should be the number of 1s divided by 2. Figure 3 depicts the summing for the case of $L = 3$ where $101 + 101 + 101 = 1111$. This simple construction (plus some other known techniques [3], [17], [12]) yields that multiplications, parallel sums, divisions and shift operations can be performed in one parallel step using RMs.

Next, we consider counting where the RM is restricted to use buses of length $\leq N^{\frac{1}{2}}$ where N is the input size. The algorithm is depicted in figure 4 and contains the following four steps:

$restricted\_counting_2(N, x_1, \ldots, x_N)$:

STEP-1 A set $RM_1, \ldots, RM_{N^{\frac{1}{2}}}$ each of size $N^{\frac{1}{2}} \times N^{\frac{1}{2}}$, computes counting of $N^{\frac{1}{2}}$ input bits each. This yields $N^{\frac{1}{2}}$ partial sums of $(\log N)/2$ bits.

STEP-2 The $N^{\frac{1}{2}}$ partial sums are packed into $N^{\frac{1}{2}} \times (\log N)/2$ array (denoted as Y-array in figure 4). This is done by broadcasting along $(\log N)/2$ wires for each RM_i .

STEP-3 Counting of the $N^{\frac{1}{2}}$ bits in each column of the Y-array is applied using $RM_1, \ldots, RM_{(\log N)/2}$. This yields $(\log N)/2$ partial sums $S_1, \ldots, S_{(\log N)/2}$ of $(\log N)/2$ bits each.

STEP-4 Summing of $S_1, \ldots, S_{(\log N)/2}$ is performed using the RM described in theorem 2.1. Each S_i must be first shifted i positions since each 1 in column i of the Y array is 2^i. Due to this shifting, we sum $\frac{1}{2}\log N$ numbers of $\log N$ bits. Thus we need a RM of $\log N \times \frac{1}{2}\log^2 N$ RM to be used as described in theorem 2.1. Since all reconfiguration paths must be less than $N^{\frac{1}{2}}$ we get that $N^{\frac{1}{2}} > \frac{1}{2}\log^2 N$.

Thus the following claim hold:

Theorem 2.2: For $\frac{1}{2} \cdot (logN)^2 \leq N^{\frac{1}{2}}$, counting of N input bits can be done in four steps by a $N^{\frac{1}{2}} \times N$ RM restricted to $N^{\frac{1}{2}}$ bus lengths.

This counting can be extended to $d(N) = N^{1/k} \quad k = 2, 3, 4, \ldots$ as follows. For the case $k = 3$ we can use the following steps:

$restricted\_counting_3(N, x_1, \ldots, x_N)$:

STEP-1 A set $RM_1, \ldots, RM_{N^{\frac{2}{3}}}$ each of size $N^{\frac{1}{3}} \times N^{\frac{1}{3}}$ RMs computes counting of $N^{\frac{1}{3}}$ input bits each. This yields $N^{\frac{2}{3}}$ partial sums of $(\log N)/3$ bits.

STEP-2 The $N^{\frac{2}{3}}$ partial sums are packed into $N^{\frac{2}{3}} \times (\log N)/3$ Y-array (similar to the way it was done in figure 4).

STEP-3 Counting of the $N^{\frac{2}{3}}$ bits in each column of this

Y-array is performed using

$$S_1 = restricted\_counting_2(N^{\frac{2}{3}}, Y[1][1], \ldots, Y[1][N^{\frac{2}{3}}])$$
$$S_2 = restricted\_counting_2(N^{\frac{2}{3}}, Y[2][1], \ldots, Y[2][N^{\frac{2}{3}}])$$
$$\ldots\ldots\ldots$$
$$\ldots\ldots\ldots$$
$$S_{(\log N)/3} = restricted\_counting_2(N^{\frac{2}{3}}, Y[(\log N)/3][1], \ldots, Y[(\log N)/3][N^{\frac{2}{3}}])$$

Each S_i should be shifted according to its column position, i.e., i positions.

STEP-4 Summing of the shifted $S_1, \ldots, S_{(\log N)/3}$ is performed using the algorithm of theorem 2.1. Hence, for the summing we need a RM of $\frac{1}{3}\log N \times \frac{1}{3}\log^2 N$. Consequently it follows that $\frac{1}{3}\log^2 \leq N^{\frac{1}{3}}$.

Clearly this recursive scheme can be used for any $d(N) = N^{1/k} \quad k = 2, 3, 4, \ldots$ using

$$S_i = restricted\_counting_{k-1}(N^{\frac{k-1}{k}}, Y[i][1], \ldots, Y[i][N^{\frac{k-1}{k}}])$$

The following claim holds:

Theorem 2.3: For $\frac{1}{k}\log^2 N \leq N^{\frac{1}{k}}$, counting of N input bits can be done in $2 \cdot k$ steps by a RM restricted to $N^{\frac{1}{k}}$ bus lengths.

Finally, we can solve the condition $\frac{1}{k}\log^2 N \leq N^{\frac{1}{k}}$ as follows:

- Using the fact that $N^{\frac{\log\log N}{\log N}} = \log N$ we get that

$$\log^2 N \leq N^{\frac{1}{k}} \iff k \leq \frac{\log N}{2 \cdot \log\log N} \quad (1)$$

- Applying some calculations (omitted due to space limitations) to this equation we can improve this bound to:

$$\log^2 N \leq N^{\frac{1}{k}} \iff k \leq \frac{\log N}{\log\log N + 2 \cdot \log\log\log N} \quad (2)$$

For example, if $N = 2^{32}$ then the first bound implies $k = \frac{32}{2 \cdot 5} = 3.3$ while the second bound yields that $k = 4$ which is the correct result.

We remark that by adding another step (STEP-3.5) to the $restricted\_counting_2$ algorithm, repeating the partial sums calculations, we can end with $(\log N)/2$ partial sums each of a size of $(\log\log N)$. By carefully wire the results, we can end using a smaller RM for the final summing of a length of $\frac{1}{2}\log N \cdot \log\log N$. This gives us a higher k for the same N, since $\frac{1}{k}\log N \cdot \log\log N \leq N^{\frac{1}{k}}$ holds.

3. Lower bound

An almost matching lower bound for counting can be shown for the RRM. However, due to space limitation its proof has been omitted. The lower bound works by replacing a T-steps computation of the RM by a stronger model of T non-deterministic levels of Branching programs $B^1, \ldots B^T$ such that:

- B^1 is any polynomial (in N) DAG whose some of its edges are labeled by input variables $x_i, \neg x_i$. Based on the input values edges whose label are false, are disconnected.
- $B^{t>1}$ is any polynomial (in N) DAG whose some of its edges are labeled by $e_i, \neg e_i$ where e_i are edges of B^{t-1}. An edge labeled e_i in B^{t-1} is valued true iff there is a path from a leaf node in B^{t-1} to e_i (vice versa for edges labeled $\neg e^i$).

We show that

Lemma 3.1: A d-restricted (#labeled edges in any path) LBP^T, $d = N^{\frac{1}{k}}/k$, cannot compute the function $f(x_1, x_2, \ldots, x_n) = \sum x_i = n/2$ (and hence counting) in less than $k + 1$ steps.

In addition we show that LBP^T can simulate any T steps of the RM. Consequently, we get matching lower bound for counting showing the optimality of the above algorithm.

4. Feasibility and FPGA results

Two realizations of the RM switch for the RM-counting are presented, followed by comparisons between the delay for a $d(n) \times d(n)$ RM-counting in our algorithm, and equivalent realizations of an adder based circuit. Finally, we show a comparison of the two mentioned methods synthesized on an FPGA using Xilinx Vivado. The RM switch for the RM-counting is realized using three basic on-off switches as described in figure 1 (upper left part). There are two possible states $< W \longrightarrow N, S \longrightarrow E >$ and $< W \longrightarrow E, S, N >$ that are determined by a control bit x directly connected to an input bit. The state of each basic switch can be either connect or disconnect generating the required configuration. Figure 5 describes the realization of the RM-switch using three NMS transistors to implement the basic on-off switches we need. These three transistors are controlled by the input value X and one inverter. The following holds:

- Only the nmos transistors along the signal-path are in on-state, the rest are in-off state and all basically disconnected from the transistors of the signal path.
- The voltage after each transistor along such a chain of nmos-transistors is $vdd - v_{threshold}$, thus there is no need to insert buffers to restore the signal strength.
- An nmos-transistor at on-state can be regarded as a source-drain resistor of resistance R + capacitor with capacitance C to the ground. Thus, the delay along a chain of $N^{\frac{1}{k}}$ nmos transistors, is based on Elmore model, $N^{\frac{2}{k}} \cdot \frac{R \cdot C}{2}$. The capacitors of the rest of the transistors and the inverter can be ignored while computing the delay of $N^{\frac{1}{k}} \times N^{\frac{1}{k}}$ counting RM.

The implementation for RRM counting of N inputs and k levels contains:

1) k levels of $N^{\frac{1}{k}} \times N^{\frac{1}{k}}$ sub-RMs to perform the different counting operations of STEP-1 (see figure 4). Once all switches have been configured, the counting signal (see figure 2) needs to traverse $N^{\frac{1}{k}}$ switches to the output of this step. This is referred as the signal critical path.

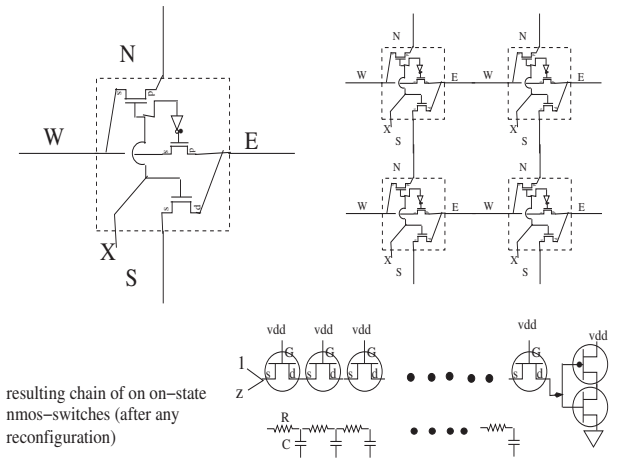


Fig. 5: *NMOS realization of the RM switch and equivalent RC chain of the signal path*

2) k levels of summing where each summing is applied to at most $\frac{\log N}{k}$ numbers in $\frac{k+1}{k}$ bits of STEP-4 (see figure 4). The resulting sum-RM needed for this summing is of size $\frac{\log N}{k} \times \frac{\log N}{k} \cdot \log N$, which is still in the restriction of $d(n) \leq N^{\frac{1}{k}}$. This is because we have selected the suitable value for k from eq. 1,2.

Thus, all signal paths are of the same length and the delay for restricted counting is $2k \cdot N^{\frac{2}{k}} \cdot \frac{R \cdot C}{2}$

This delay should be compared to the delay of a CMOS circuit that computes counting of N inputs using a complete binary tree of adders:

- $\log N$ adders in each path of this summing tree.
- Each adder has depth (critical path) of $4 + 2 \cdot \log \log N$ gates (see [20]).
- Each gate has a critical path of two-three CMOS transistors.
- The switching delay of each CMOS transistor can be estimated by $2 \cdot R \cdot C$, counting for the delay to switch each transistor to conducting state + source-drain propagation delay of the current.

$$tree\_of\_adders\_delay = \log N \cdot (4+2 \cdot \log \log N) \cdot 3 \cdot 2 \cdot R \cdot C$$

For a given N, the restricted RM-counting will outperform the tree of adders circuit if

$$\log N \cdot (4 + 2 \cdot \log \log N) \cdot 3 \cdot 2 \cdot R \cdot C \geq 2k \cdot N^{\frac{2}{k}} \cdot \frac{R \cdot C}{2}$$

We get that

$$24 \cdot \log N + 12 \cdot \log N \cdot \log \log N \geq k \cdot N^{\frac{2}{k}}$$

This hold for most cases of $N \geq 2^{16}$ and $k \geq 4$.

Another possibility to implement the RRM is to use Optical Ring Resonator (ORR) [7] which consists of silicon micro-ring resonator coupled to two straight wave-guide. Basically, ORR allows light to "jump" from one wave-guide ($'L'$) to the other wave-guide $'R'$, when an electrical field is applied. Unlike CMOS transistors, this is done without any latency apart from the speed of light inside $|L| + |R|$. Note that ORR are directional and thus (ideally) light beams cannot propagate "backwards" in the switching network that we are building. The RM-switch work with one ORR and one Y-junction. Therefore, light coming from $'W'$ will continue either to $'N'$ or to $'E'$, depending on $'X'$. However, light coming from $'S'$ will always continue to $'E'$. For this optical realization, the delay calculation should be:

$$\log N \cdot (4+2 \cdot \log \log N) \cdot 3 \cdot 2 \cdot R \cdot C \geq 2k \cdot N^{\frac{1}{k}} \cdot ORR\_dellay$$

For current CMOS technology $R \cdot C = 5ps$, and

$$ORR\_delay = ORR\_length/speed\_of\_light\_silicon = 82 \cdot 10^{-6} \cdot 208 \cdot 10^{6} = 0.4ps$$

which is significantly faster for any value of N and k.

Finally, the design was synthesized on the Kintex FPGA using Xilinx Vivado. The proposed RRM-counting algorithm was compared with a regular summing circuit which is a binary tree of adders (BTA). From the above analysis, it is clear that the RRM-counting will outperform an equivalent circuit of BTA only for large input size. Both circuits (RRM-counting and BTA) are basically combinatorial, and the clock period is measured for one round of computation. The BTA circuit was optimized to use the minimal amount of wires possible to be embedded in the FPGA. Figure 7 depicts the synthesis results for both the restricted RM-counting and the BTA circuits. For the RRM-counting, we used two restrictions on the number of switches a broadcast can traverse $N^{\frac{1}{k=4}}$, $N^{\frac{1}{k=5}}$. These results suggest that:

- Only at $k = 5$ and for $N = 2^{15}$, the clock period of the RRM-counting outperforms that of the BTA circuit. This is close to what was predicted by the above analysis, showing that the RRM-counting for sufficiently large values of N, will outperform BTA circuits.
- Observe that the BTA clock period is constantly increasing by 1ns for an increase of 1 in $\log N$, while the RRM-counting improves its run-time (for $k = 5$) by 0.36ns per increase of 1 in $\log N$.
- Even when the RRM-counting clock period is longer than that of the corresponding BTA, it is significantly more efficient in terms of power and LUTs, as the RM uses far less logic than the BTA circuit. In particular, note that the BTA for N^{16} can not fit into the FPGA while the RRM-counting can.

5. Verilog coding

The programming style to express RM algorithms is known to be a hard problem. Therefore, describing the Verilog coding of RRM is essential, since it demonstrates a new approach of direct and simple way of coding. Previous work that has addressed this issue, tried to extend C code to include RM processing elements (PEs) and low-level notion of the

log N		Rmcounting K=4	Rmcounting K=5	BTA
8	Period(ns)	9.16		6
	LUTs	573		665
	Power(W)	0.051		0.093
10	Period(ns)		9.4	8
	LUTs		2026	2685
	Power(W)		0.171	0.258
12	Period(ns)	17.6		10
	LUTs	10434		10831
	Power(W)	0.263		0.839
15	Period(ns)		11.2	12
	LUTs		18803	87545
	Power(W)		0.899	1.636
16	Period(ns)	17		
	LUTs	139059		Too large to handle
	Power(W)	1.42		

Fig. 6: *FPGA synthesis results*

RM-steps. In a RM-step, a PE would: 1-Reconfigure its switch; 2-Broadcast values on its un-connected ports; 3- Read values from its connected ports; 4- Computes new values. Integrating RM-steps into C, created complex programming problems, most of them coordination issues, such as:

1) Conditional execution of RM-steps may cause some PEs to execute more RM-steps than the others.
2) Recursion and arbitrary function calls involving RM-steps, creates non-equal length RM-steps.
3) Inner computations between RM-steps can vary in the amount of instructions that are executed.
4) Different RM algorithms use different size of RMs. Should a programmer work with variable or fixed size RMs?

All these problems are hard to solve for a C like programming style due to its serial mode of execution. However, in Verilog (or any other Hardware Description Language) this can be naturally done since it employs a parallel mode of execution wherein clock synchronization is imposed.

The following code show how simple counting can be programmed in Verilog. First, we define the set of switches needed for the algorithm. For simple counting, we need only one type of switch that have two states: a band $< W \Longrightarrow N\ ,\ S \Longrightarrow E >$ or horizontal-pass $< W \Longrightarrow E\ ,\ S,\ N >$. The control of this switch is done by the input bit x which is directly fed to the switch. We use bufif1 tristate devices to implement this switch.

```
module SW (x, w, s, e , n);
    input   w,s,x;
    output  e,n;
    tri     e,n;
    bufif1  bwn(n, w, x);
    bufif1  bse(e, s, x);
    bufif1  bwe(e, w, ~x);
endmodule
```

Next, we define the module that creates a complete 8×8 RM. We mainly use the $generate-for$ construct to create a two dimensional array of switches. The input bits are passed to the module via an array of $N = 8$ wires $inp[\ldots]$. The x parameter of each switch will be assigned the suitable input bit when this switch is instantiated by the $generate - for$ construct. Thus, the following code (for one dimension of the RM) will create/instantiate six switches $s_1, \ldots, s_6$ with the suitable input bit $inp[X]$.

```
genvar X,Y;
generate
for (X=1; X < N-1; X=X+1)
    SW   s (inp[X],...);
```

Next, when a switch s_X is instantiated, its W-port needs to be connected to s_{X-1}'s E-port. This is done by declaring an array $WLR[\ldots]$ of wires and connecting to the E-port of s_{X-1} to the W-port of s_X via a wire $WLR[X-1]$ as follows:

```
wire  WLR [0:N-1];
genvar X,Y;
generate
for (X=1; X < N-1; X=X+1)
  SW s(inp[X],WLR[X-1],...,WLR[X][Y],...);
```

Since the RM is two dimensional, we need to connect $s_{X,Y}$'s E-port to $s_{X-1,Y}$'s W-port and $s_{X,Y}$'s S-port to $s_{X,Y-1}$'s N-port. Thus, full connections are obtained using 2D arrays of wires as follows:

```
wire  WLR [0:N-1][0:N];
wire  WUD [0:N-1][0:N];
genvar X,Y;
generate
for (X=1; X < N-1; X=X+1)
 for (Y=1; Y < N; Y=Y+1)
  SW s(inp[X],WLR[X-1][Y],
       WUD[X][Y-1],WLR[X][Y],WUD[X][Y]);
```

Creating The overall RM is more complicated since the external edges of the RM form the ends and should be instantiated differently than the internal switches of the RM. The following module instantiate a full RM using separate $generate-for$ statements for the edges and separate instantiations for the corners of the RM.

```
module RMcounting #(parameter N=8)(inp,out);
input inp;
output out;
wire [0:N-1] inp;
wire [0:N]   out;
wire  WLR [0:N-1][0:N];
wire  WUD [0:N-1][0:N];
genvar X,Y;
generate
for (X=1; X < N-1; X=X+1) begin
 for (Y=1; Y < N; Y=Y+1) begin
   SW s(inp[X],WLR[X-1][Y],
        WUD[X][Y-1],WLR[X][Y],WUD[X][Y]);
  end end endgenerate
generate for (Y=1; Y < N; Y=Y+1) begin
 SW sl(inp[0],1'b0,WUD[0][Y-1],
       WLR[0][Y],WUD[0][Y]);
end endgenerate
generate for (Y=1; Y < N; Y=Y+1) begin
 SW sr(inp[N-1],WLR[N-2][Y],
       WUD[N-1][Y-1],out[Y],WUD[N-1][Y]);
end endgenerate
```

```
generate for (X=1; X < N-1; X=X+1) begin
 SW sd(inp[X],WLR[X-1][0],
       1'b0,WLR[X][0],WUD[X][0]);
end endgenerate
generate for (X=1; X < N-1; X=X+1) begin
 SW su(inp[X],WLR[X-1][N],
       WUD[X][N-1],WLR[X][N],WUD[X][N]);
end endgenerate
SW s00(inp[0],1'b1,1'b0,WLR[0][0],WUD[0][0]);
SW s0N(inp[0],1'b0,WUD[0][N-1],
                  WLR[0][N],WUD[0][N]);
SW sN0(inp[N-1],WLR[N-2][0],1'b0,
                  out[0],WUD[N-1][0]);
SW sNN(inp[N-1],WLR[N-2][N],
       WUD[N-1][N-1],out[N],WUD[N-1][N]);
endmodule
```

6. Conclusion

In this paper we presented the RRM wherein broadcasting along buses is restricted not to pass through more than $d(n) = n^{\frac{1}{k}}$ switches, where $k = 2, 3, \ldots$ and n is the input size. We argued that using the RRM, the theoretical broadcast assumption of $O(1)$ is feasible, using current transistors technology. We have shown that the fundamental problem of counting can be done on the RRM in $2k$ steps. An almost matching lower bound was presented, showing that the RRM can not compute counting in less than $k + 1$ steps. Finally, we presented a realization of the counting algorithm, coded directly in Verilog. We demonstrated that RM algorithms can be directly synthesized to circuits using hardware description language even for large size of RMs, while previous realizations of the RM were mainly to a small-size grid of Soft-CPUs using MUX-gates for reconfiguration, which cannot scale well. The Verilog realization of the RRM is shown to outperform a regular optimal parallel adders-circuit both for current CMOS technology and Optical ring resonators. This work thus presents a significant step toward a realization of the RM algorithms.

References

[1] Y. Ben-Asher and A. Schuster. The bus-usage method for the analysis of reconfiguring networks algorithms. In *Proc. of the Intl. Parallel Processing Symp.*, Beverly Hills, March 1992.

[2] Yosi Ben-Asher, Dan Gordon, and Assaf Schuster. Efficient self simulation algorithms for reconfigurable arrays. In *AlgorithmsâĂŤESA'93*, pages 25–36. Springer, 1993.

[3] Yosi Ben-Asher, David Peleg, and Assaf Schuster. The complexity of reconfiguring network models. In *Israel Symposium on Theory of Computing Systems*, pages 79–90, 1992.

[4] Bryan Beresford-Smith, Oliver Diessel, and Hossam ElGindy. Optimal algorithms for constrained reconfigurable meshes. *Journal of Parallel and Distributed Computing*, 39(1):74–78, 1996.

[5] G. Chen, B. Wang, and H. Li. Deriving algorithms on reconfigurable networks based on function decomposition. *Theoretical Computer Science, 120(2):215–27, November*, 1993.

[6] J. L. Trahan C.P.Subbaraman and R. Vaidyanathan. List ranking and graph algorithms on the reconfigurable multiple machine. In *Proceedings of International Conference on Parallel Processing, pages III–224–247. CRC Press, August*, 1993.

[7] D. Ding and D. Z. Pan. Oil: a nano-photonics optical interconnect library for a new photonic networks-on-chip architecture. In *Proceedings of the 11th international workshop on System level interconnect prediction*, 2009.

[8] Hatem M El-Boghdadi, Ramachandran Vaidyanathan, Jerry L Trahan, and Suresh Rai. On the communication capability of the self-reconfigurable gate array architecture. In *ipdps*, page 0152b. IEEE, 2002.

[9] W. C. Elmore. The transient response of damped linear networks with particular regard to wideband. *Journal of Applied Physics*, 19:55–63, 1948.

[10] Heiner Giefers and Marco Platzner. Armlang: a language and compiler for programming reconfigurable mesh many-cores. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

[11] Heiner Giefers and Marco Platzner. An fpga-based reconfigurable mesh many-core. 2013.

[12] J. Jang, H. Park, and V.K. Prasanna. An optimal multiplication algorithm on reconfigurable mesh. In *Proc. Symp. on Parallel and Distributed Processing*, pages 381–391, 1992.

[13] J. Jang and V.K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. In *Proc. Inter. Parallel Processing Symp.*, pages 130–137, March 1992.

[14] Ju-wook Jang and Viktor K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. In *Proceedings of 6th International Parallel Processing Symposium, pages 130–137. IEEE,*, 1992.

[15] Y. Matias and A. Schuster. On the power of a 2-band reconfigurable network. Unpublished Manuscript, 1992.

[16] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, and Q.F. Stout. Image computations on reconfigurable VLSI arrays. In *Proceedings of the Conference on Vision and Pattern Recognition*, pages 925–930, 1988.

[17] K. Nakano and K. Wada. Integer summing algorithms on reconfigurable meshes. *Theoretical Computer Science, Vol. 197, pp. 57–77, Jan*, 1998.

[18] Madhusudan Nigam and Sartaj Sahni. Sorting n numbers on n× n reconfigurable meshes with buses. *Journal of Parallel and Distributed Computing*, 23(1):37–48, 1994.

[19] Stephan Olariu, James L. Schwing, and Jingyuan Zhang. Fast component labeling and convex hull computation on reconfigurable meshes. *Image and Vision Compututing, 11(7):1993, September*, 1993.

[20] Oday Abdul Lateef Abdul Ridha. Performance estimation of n-bit classified adders. 2013.

[21] Reetinder Sidhu, Sameer Wadhwa, Alessandro Mei, and Viktor K Prasanna. A self-reconfigurable gate array architecture. In *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, pages 106–120. Springer, 2000.

[22] JL Trahan and R Vaidyanathan. Relative scalability of the reconfigurable multiple bus machine. In *Proc. Workshop Reconfigurable Arch. and Algs*, 1996.

[23] Biing-Feng Wang and Gen-Huey Chen. Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems. *IEEE Transactions on Parallel and Distributed Systems, 1(4):500–507, October*, 1990.

Hardware Implementation of Parallel Algorithm for Setting Up Benes Networks

Yikun Jiang and Mei Yang
Department of Electrical and Computer Engineering
University of Nevada, Las Vegas
Emails: jiangy3@unlv.nevada.edu, Mei.Yang@unlv.edu

***Abstract*— Benes/Clos networks have been used in many areas, such as interconnection network in parallel computers, multiprocessors system, and networks-on-chip. The parallel switch setting algorithm is the key to satisfy the requirements of high performance switching networks. The Lee's routing algorithm is by far the most efficient parallel routing algorithm for Benes networks. However, there is no hardware implementation for this algorithm. In this paper, the Lee's routing algorithm is fully implemented in RTL and synthesized. We have refined the algorithm in data structure and initialization/updating of relation values to make it suitable for hardware implementation. The simulation and synthesis results of the switching setting circuits for 8x8 to 32x32 Benes networks confirm that the timing, area, and power consumption of the circuit is consistent with the complexity of the Lee's algorithm. To the best of our knowledge, this is the first complete hardware implementation of the parallel switch setting algorithm which can handle all types of permutations including partial ones.**

***Keywords*— Benes, Parallel Algorithm, Hardware, RTL, Implementation, Synthesis**

I. INTRODUCTION

Both Benes and Clos networks are rearrangeably non-blocking multi-stage interconnection networks. Benes network is a special case of Clos network which has $N = 2^n$ inputs and outputs. The Benes network is constructed with 2×2 switching nodes recursively. Due to their non-blocking property and relative smaller number of crosspoints, Benes/Clos networks have received much attention in both academia and industry. Benes/Clos networks have been used in many areas, such as interconnection network in parallel computers, multiprocessors system [1], and networks-on-chip [2][3][4][12][13]. In packet switching systems, the switch fabric must be able to provide internally conflict-free paths for the requesting packets in each time slot [5]. This is implemented by setting the states of all switches in the network. It is clear that the routing assignment (i.e., switch setting) scheme in Benes/Clos networks has a strong impact to the efficiency of the Bene/Clos networks.

A number of switch setting algorithms have been developed in the past few decades, including sequential algorithms and parallel algorithms. Sequential algorithms such as looping algorithms [7] are designed for circuit switching systems where the switching configuration can be rearranged at relatively low speed. In [7], a switch setting algorithm with time complexity $O(NlogN)$ is proposed based on Waksman's proof. As a matter of fact, using sequential algorithm, the $N \times N$ Benes network cannot be set up in less than $O(NlogN)$ time, because there are $O(NlogN)$ switches. The set-up time is much longer than the latency in Benes networks, which is $O(logN)$ for $N \times N$ network. In order to obtain a switch setting algorithm of complexity comparable to the network latency, parallel algorithms are needed.

In [9], Nassimi and Sahni developed a parallel set-up algorithm which runs significantly faster than the sequential algorithm based on Waksman's proof. The complexity of this algorithm depends on the parallel computer model and the number of processing elements available. Four SIMD models with different topologies are studied: Completely Interconnected Computer (CIC) with time complexity of $O(log^2 N)$, Mesh-Connected Computer (MCC) with time complexity of $O(\sqrt{N}\, log^2 N)$, Cube Connected Computers (CCC) with time complexity of $O(log^4 N)$, and Perfect Shuffle Computer (PSC) with time complexity of $O(log^4 N)$. The time complexity of topologies other than CIC is fairly high. However, CIC is simply too complex to be realized. In addition, this parallel algorithm [9] cannot handle the partial permutations. The authors also proposed a self-routing algorithm for Benes network [9] to route through the network using destination tags. However, this algorithm cannot route all permutations.

In [5][11], Lee and Liew present a parallel routing algorithm for Benes Networks. It has time complexity $O(log^2 N)$ which is same as CIC but using only $N/2$ processing elements [9]. This algorithm was developed based on the previous work in [8] and [9], but can handle the partial permutation problem. In addition, the algorithm can be extended and applied to Clos networks with two's power number of central modules. In the literature, there is nearly no hardware implementation of this parallel algorithm. In [3], a simple hardware design based on Lee's algorithm for 16×16 Benes network in FPGA is presented. However, no detailed design and simulation results are shown in that paper. Another problem about [3] is that, the work is only limited to the switch setting unit for the first stage of 16×16 Benes network. Without the design of the switch setting circuit for different size networks, there is no way to tell the trend of how the hardware cost would increase correspondingly when the network size grows.

In this paper, we present the hardware design of Lee's parallel routing algorithm for Benes networks in different

sizes ranging from 8×8 to 32×32. The algorithm is refined to make it more suitable for hardware design. The Register-Transfer-Level (RTL) design of the algorithm is coded in Verilog, simulated, and synthesized using Cadence tools under 65nm technology. The timing delay trend is consistent with the time complexity trend of Lee's algorithm. The switch setting hardware design can be integrated with Benes network circuit to be used in high-performance network-on-chip systems.

The rest of the paper is organized as follows. Section II presents the parallel routing algorithm. Section III presents the RTL design and improvement of Lee's parallel routing algorithm. Section IV presents the synthesis results and analysis of the results.

II. LEE'S PRALLEL ROUTING ALGORITHM

A. Lee's Algorithm

Lee's parallel algorithm can be decomposed into four major steps: initialization, searching, merging and calculating the permutation for subnetworks. Denote the set of input and output ports as I and O, respectively, i.e., $I = O = \{0, 1, \ldots, N-1\}$, and $\pi: I -> O$ be an input-output permutation indicating connection requests. We use $(i,\ j)$ to indicate the ith input port is going to connect to the jth output port in the permutation. In this part, we will use an example permutation to elaborate the main concept of this algorithm. In the below permutation, $'X'$ means this input port has no output request.

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 3 & 2 & 6 & 4 & 7 & 5 & X \end{pmatrix}$$

Because of the symmetric routing constraint, the algorithm only need to find out the routing bits of the stages in one Omega subnetwork, then the routing bits of the counterpart stages in the other Omega network will be determined. In Lee's algorithm, the output side switch setting is determined first, then the input side switch setting is derived.

B. Initialization

The first step of Lee's algorithm is to build the connections between output switching nodes using relation values. The connection between output switching nodes are built on the internally conflict-free constraint, to avoid this internal conflict, the algorithm need to group switching nodes with the same relation together, and assign the switch state values to them consistently.

Here, we adopt the same notations as in (Lee et al., 1996). We use a_i and b_i to denote the switch state value of input/output switching node a_i and b_i, respectively. Let $\alpha: I \to \{0,1\}$ and $\beta: O \to \{0,1\}$, where $\alpha(k)$ is the routing bit of kth input, and $\beta(k)$ is the routing bit from kth output.

From [5], the symmetric self-routing constraint requires that

$$\alpha(k) = \beta(\pi(k)) \qquad k=0, 1, \ldots, N-1 \qquad (1)$$

The internal conflict-free constraint requires that

$$\alpha(k) = \overline{\alpha}(k+1),\ \beta(k) = \overline{\beta}(k+1) \quad k=0, 1, \ldots, N-2 \qquad (2)$$

The combination of (1) and (2) gives

$$\beta(\pi(k)) = \alpha(k) = \overline{\alpha}(k+1) = \overline{\beta}(\pi(k+1)) \qquad k=0, 1, \ldots, N-2 \qquad (3)$$

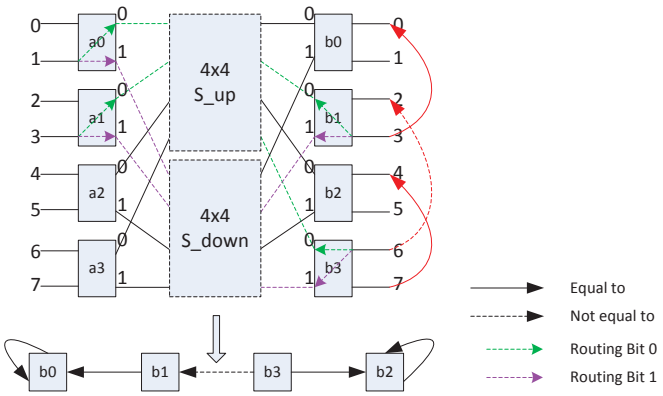


Fig. 1 Initialization

Then we have

$$\alpha_i = \begin{cases} a_{\frac{k}{2}}, & k \text{ is even} \quad k = 2i \\ \overline{a_{\frac{k-1}{2}}}, & k \text{ is odd} \quad k = 2i+1 \end{cases} \qquad (4)$$

$$\beta_i = \begin{cases} b_{\frac{k}{2}}, & k \text{ is even} \quad k = 2i \\ \overline{b_{\frac{k-1}{2}}}, & k \text{ is odd} \quad k = 2i+1 \end{cases} \qquad (5)$$

For the given permutation, we have:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 3 & 2 & 6 & 4 & 7 & 5 & X \end{pmatrix} =>$$

$$\begin{pmatrix} a_0 & \overline{a_0} & a_1 & \overline{a_1} & a_2 & \overline{a_2} & a_3 & \overline{a_3} \\ b_0 & \overline{b_1} & b_1 & b_3 & b_2 & \overline{b_3} & \overline{b_2} & X \end{pmatrix}$$

For the ith input switching node, we refer to the output port pair (k, l) corresponding to the input port pair $(2i, 2i+1)$ as a *connection pair*. Then we obtain:

$$\begin{pmatrix} 2i & 2i+1 \\ k & l \end{pmatrix} => \begin{pmatrix} a_i & \overline{a_i} \\ \beta(k) & \beta(l) \end{pmatrix}$$

Based on Eqn. (3), we have:

$$\beta(k) = \overline{\beta(l)} \qquad (6)$$

Consider the given permutation, taking $(1, 3)$ as example. As shown in Fig. 1, in order to have the same routing bits ($'0'$ or $'1'$) for input port 1 and output port 3, the corresponding input switching node must set the state value base on the corresponding output switching node, i.e., for input/output permutation $\binom{1}{3}$, we have $\alpha(1) = \overline{a_0}$, $\beta(3) = \overline{b_1}$, since $\alpha(1) = \beta(3)$, then we can get $a_0 = b_1$. Similarly, for $\binom{3}{6}$, we derive, $\overline{a_1} = b_3$. Together, we obtain

$$\begin{aligned} a_0 &= b_0, & \overline{a_0} &= \overline{b_1} \\ a_1 &= b_1, & \overline{a_1} &= b_3 \\ a_2 &= b_2, & \overline{a_2} &= \overline{b_3} \\ a_3 &= \overline{b_2}, & \overline{a_3} &= X \end{aligned}$$

After eliminating all a from above equations, we can obtain a set of $N/2$ initializing equations as follows:

$$b_1 = b_0,\quad b_3 = \overline{b_1},\quad b_3 = b_2,\quad b_2 = X$$

These equations about b can help us to build the relation connections between output switching nodes as shown in Fig. 1. All output switching nodes are connected like a linked list, where the index of the state variable is taken as the node

address. Each initializing equation is used to establish a pointer, in which the state variable with larger index points to the other with smaller index.

After initialization step, all output switching nodes can be grouped into equivalent classes. For switching nodes in the same class, the state value of any switching node is relevant to the state value of others. The *representative node* of each class is the switching node with the smallest index number. For the above example, as shown in Fig. 1, all output switching nodes are in the same class. The representative node of the group is b_0. Regardless of the Benes network radix, the initialization step is processed at all PEs at the same time with time complexity $O(1)$.

C. Searching

As shown in Fig. 1, there are two pointer types, Type 0 Pointer indicating the two state variables are equal, and Type 1 Pointer indicating the two state variables are not equal. All switching nodes except the representative node in the group will go through the searching step to point to the representative node. The time complexity of searching step is $O(logN)$.

Fig. 2 shows the searching result for Fig. 1.

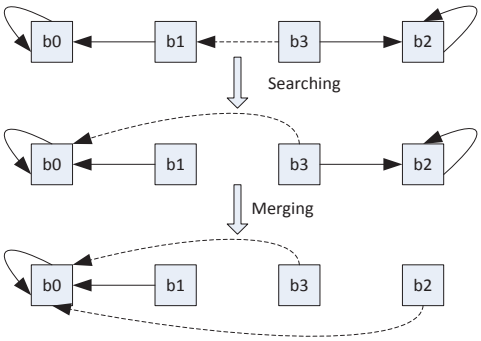


Fig. 2 Searching and Merging

D. Merging

Usually, among the nodes belonging to the same class, there should be only one endpoint which is the representative node of the class. If there are two endpoints in one class, then the merging step is needed to eliminate one of them. The time complexity of this merging step is $O(1)$. Fig. 2 shows that the two endpoints b_0 and b_2 are pointed by b_3, which means the value of b_3 will be determined by the values of b_0 and b_2, causing confliction. As shown in Fig. 2, after the merging step, the direct connection between two endpoints b_0 and b_2 is found.

After all switching nodes point to the representative of the class, the state values of all switching nodes can be determined by assigning the state value of the representative as 0 or 1. One of the assignments of the above example is derived as by letting $b_0 = 0$:

$$State\ (b_0, b_1, b_2, b_3) = (0, 0, 1, 1)$$

By applying the symmetric routing constraint, the state values of input switching nodes should be setup as:

$$State\ (a_0, a_1, a_2, a_3) = (0, 0, 1, 0)$$

Fig. 3 shows the settings of input/output switching nodes for the given permutation π.

E. Permutation for Subnetworks

After the state values of input/output switching nodes are determined, the switch settings of two inner $\frac{N}{2} \times \frac{N}{2}$ subnetworks can be determined recursively. The permutations of the two inner subnetworks can be derived by tracing the routing paths from both input and output sides. Then Lee's algorithm is applied to derive the state values of the input/output switching nodes of the two subnetworks. The time complexity to calculate those permutations for subnetworks is $O(1)$. In a recursive manner, the state values of all stages will be computed by the Lee's parallel routing algorithm.

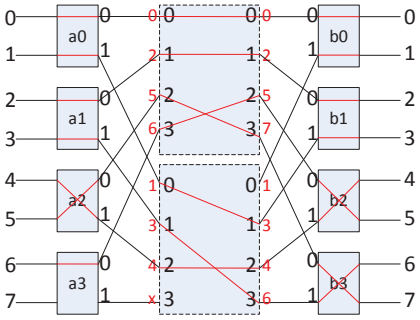


Fig. 3 Permutation for Subnetwork

Fig. 3 shows the connections of the two inner subnetworks and the derived two permutations π_0 for S_{up} and π_1 for S_{down} for two inner subnetworks, respectively.

$$\pi_0 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 3 & 2 \end{pmatrix}, \pi_1 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 3 & 2 & x \end{pmatrix}.$$

Continue this process until the state values of the middle stage switching nodes are determined.

As we can see from the description in above section, the searching step is the only procedure which is relevant to the radix of Benes network. All the other procedures could be finished in $O(1)$. The time complexity for each round is determined by the searching procedure which is $O(logN)$.

III. Hardware Design of Lee's Algorithm

A. Design Flow

The hardware design of Lee's algorithm follows the common RTL design flow which consists of four steps: 1) specification, 2) RTL design, 3) simulation of the RTL code, 4) synthesis of the RTL design. In the second step, we use Verilog HDL to implement the RTL design of Lee's parallel algorithm.

As shown in Fig. 4, the switch setting circuit of $N \times N$ Benes network takes the input of N output port indexes representing the permutation and generates the switch setting of every two stages as well as the permutation of two inner $\frac{N}{2} \times \frac{N}{2}$ subnetworks. There are $N/2$ processing elements (PE), each representing an output switching node, are connected by the main frame. Each P_i holds several variables. In the main frame, two major parts are the control logic and shared memory. TABLE I. lists the variables used in our design. For $N \times N$ Benes network, each variable storing port index has $n = log_2^N$ bits. The global variables are shared among all processing elements.

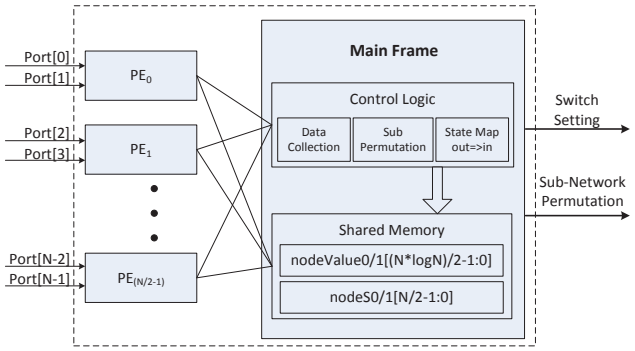


Fig. 4 Circuit Architecture

As each output switching node (represented by one processing element) has two ports, 0 and 1, we adopt a two-register structure for each output switching node to store the pointers associated with port 0/1. In the searching step of Lee's algorithm, each PE may need search in two directions. The two-register structure allows each PE keeps searching in two directions until they reach the representative nodes. Here four variables are used for storing the index of the node ($nodeValue0/1$) pointed by the port 0/1 pointer and corresponding relation value ($nodeS0/1$), respectively. The size of these shared registers is determined by the radix of Benes network. For $N \times N$ Benes network, the size of $nodeValue0/1$ is $(N/2) * logN$ bits as there are $N/2$ output switching nodes and $logN$ bits are needed to represent the index of each port. The size of $nodeS0/1$ is $N/2$ as one bit is needed to represent the relation value between two connected switching nodes, '0' represents not equal, '1' represents equal.

TABLE I. DEFINITION OF VARIABLES

Global Variable	**Meaning**	**Size (bit)**
port[N]	Store the output port index of the permutation.	$NlogN$
nodeValue0/1[$\frac{N}{2}$]	Store the index of the port which is pointed by the port 0/1 pointer of each output switching node. For example, nodeValue0/1[i] = j, 0 <= j < i <= N/2, means node i points to node j, i.e., there is a relation connection between node i and node j.	$\frac{NlogN}{2}$
nodeS0/1	Store the relation value for the connection from the port 0/1 pointer of each output switching node.	$N/2$
inNodeStateValue[$\frac{N}{2}$]	Store the state value of input switching nodes.	$N/2$
outNodeStateValue[$\frac{N}{2}$]	Store the state value of output switching nodes.	$N/2$
sub0/1_port[$\frac{N}{2}$]	Store the permutations for two inner subnetworks.	$logN/2$
Local Variable	**Meaning**	**Size (bit)**
port0/1	Stores the output port index of the connection pair corresponding to input port pair $(2i, 2i+1)$.	$logN$
preNodeValue0/1	Stores the nodeValueL0/1 before each searching procedure.	$logN$
nodeType	Two-bit value, '00' means the node doesn't point to any other node; '01' if the node points to only one other node, '11' if it points to two other nodes.	2

The control logic is responsible for the following functions:

1. Maintaining and updating the registers' data and status respectively, according to the newest information received from processing elements.
2. Calculating the setting value for switching nodes on the inputs/outputs stage.
3. Calculating the input/output permutation for the subnetworks.

B. Finite State Machine

In this part, the RTL design of Lee's parallel algorithm is presented. Following the process of Lee's parallel routing algorithm, we derive the finite state machine of each processing element as shown in Fig. 5 which encloses five steps: 1) IDLE, 2) INIT, 3) SEARCH, 4) MERGE, 5) DONE.

Each step could be divided into several states to complete the function that this step is supposed to do. Those states named with 'WAIT' as appendix are used to synchronize processing elements. All the processing elements need to wait one clock cycle so that the register values updated by other processing elements become valid in all processing elements. In the following part of this section, we will describe these five main steps.

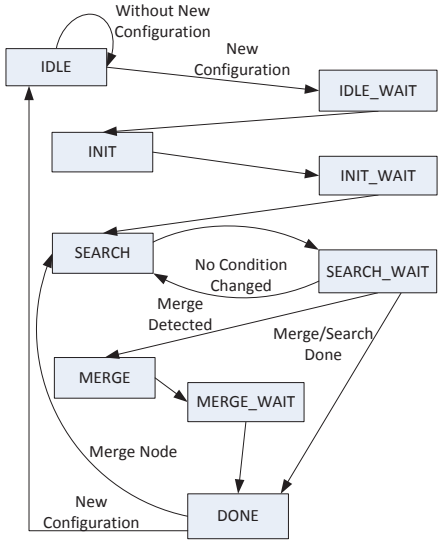


Fig. 5 State Diagram

IDLE

At the starting point, all processing elements are in the IDLE state to wait for the new permutation between input and output ports. When the new permutation arrives by setting input ports of all input switching nodes, all processing elements will enter the INIT state to conduct initialization functions. Before the processing element enters the INIT state, the control unit needs one clock cycle to synchronize with all other processing elements.

In the IDLE state, all register values are reset to default values, where nodeValue$0/1$ and preNodeValue$0/1$ are set to the current node index and nodeS$0/1$ are all reset to 0.

INIT

In Lee's parallel routing algorithm, the first step is to initialize the pointers and relation values between output switching nodes. This initialization process is determined by the permutation between inputs and outputs of Benes network. Consider the following permutation for a 16×16 Benes network:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 10 & 14 & 9 & 2 & 8 & 13 & 12 & 15 & 1 & \times & 7 & 11 & 5 & 0 & 4 & 6 \end{pmatrix}$$

There are two types of relation between two output switching nodes that have connection, equal or not equal, represented as $'0'$ or $'1'$ respectively. In Lee's parallel routing algorithm, in order to find out the relation between these two output switching nodes, the equations between routing bits of input/output switching nodes need be derived first. In our design, the relation between two output switching nodes can be derived directly from the parity of two output port indexes corresponding to the two input ports of each PE.

Given the connection pair (k, l) for an input port pair $(2i, 2i+1)$ (i.e., $port0$ and $port1$ in our design), according to Eqns. (4), (5) and (6), we derive the four possibilities of the above equation:

Case 1: k is even and l is even, we have

$$b_k = \overline{b_l};$$

Case 2: k is even and l is odd, we have

$$b_k = \overline{b_l} \rightarrow b_k = b_{\frac{l}{2}};$$

Case 3: k is odd and l is even, we have

$$\overline{b_{\frac{k}{2}}} = \overline{b_{\frac{l}{2}}} \rightarrow b_{\frac{k}{2}} = b_{\frac{l}{2}};$$

Case 4: k is odd and l is odd, we have

$$\overline{b_{\frac{k}{2}}} = \overline{\overline{b_{\frac{l}{2}}}} \rightarrow b_{\frac{k}{2}} = \overline{b_{\frac{l}{2}}}$$

As we can see from above options, when k and l have the opposite odd-even property, then their corresponding output switching nodes will have the same state value, otherwise, they have the opposite state value. The relation between two output switching nodes can be set according to odd-even property of k and l by checking $port0[0]$ and $port1[0]$ as shown in Eqn. (3).

$$NodeS0/1 = \sim(port0[0]\ XOR\ port1[0]) = \begin{cases} 0 & Equal \\ 1 & Not\ Equal \end{cases} \quad (7)$$

At each processing element P_i, the following code is used to set $nodeValue0/1$ and $NodeS0/1$, where $n = log_2^N$.

```
// pNode is the temporal variable to hold the larger node index
if (port0[n-1:1] < port1[n-1:1]) {
  pNode = port1[n-1:1];
  if (port1[0]) {
    nodeValue1[pNode] = port0[n-1:0];
    nodeS1[pNode] = (port0[0] == port1[0])? 1'b1 : 1'b0;
  }
  else {
    nodeValue0[pNode]] = port0[n-1:0];
    nodeS0[pNode] = (port0[0] == port1[0])? 1'b1 : 1'b0;
  }
}
else if (port1[n-1:1] < port0[n-1:1) {
  pNode = port0[n-1:1];
  if (port0[0]) {
    nodeValue1[pNode] = port1[n-1:0];
    nodeS1[pNode] = (port0[0] == port1[0])? 1'b1 : 1'b0;
  }
  else {
    nodeValue0[pNode]] = port1[n-1:0];
    nodeS0[pNode] = (port0[0] == port1[0])? 1'b1 : 1'b0;
  }
}
else
  null;
```

Note that each port register has width of log_2^N bits with the top $(log_2^N - 1)$ bits representing the output switching node number and the least significant bit representing the port number (0 or 1) of the output switching node as well as the parity of the output port index.

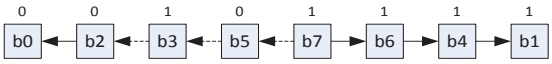


Fig. 6 Initialization

After the initialization step, all output switching nodes will be divided into one or more classes depending on the permutation of inputs/outputs as shown in Fig. 6. All nodes in the same class are bounded together such that once the state value of any node is determined, then the state values of all the other nodes will be determined. For the example shown above, if the switch setting value of $b0$ is 0, then the state values of the whole class are shown in Fig. 6.

SEARCH

In the searching step, all processing elements parallelly search and update the node pointer till reaching the representative node of the class, i.e., the switching node with the smallest index number in the class. The number of searching steps is bounded by $\frac{N}{2}$. As shown in Fig. 5, right after the state machine runs into the SEARCH state, each processing element P_i updates $nodeValue0/1[i]$ and relation values $nodeS0/1[i]$ stored locally till the pointer's values do not change in the current searching iteration. To detect the ending condition of searching step, before searching in SEARCH state, the node pointer's current value $nodeValue0/1$ will be stored in $preNodeValue0/1$.

Fig. 7 shows that after searching all processing elements point to one endpoint except the one representing $b7$, which reaches two endpoints $b1$ $(node[1])$ and $b0$ $(node[0])$. In each class, there is only one representative node. In order to solve this problem, we must merge these two end nodes pointed by the same processing element, as shown in Fig. 7, this process will be done in the MERGE state.

The following two conditions need be satisfied before transferring to the MERGE state.

- After one searching step, the value contained in register preNodeValue doesn't change.
- The switching node has type value "$nodeType == 2'b11$", which means the switching node points to two endpoints.

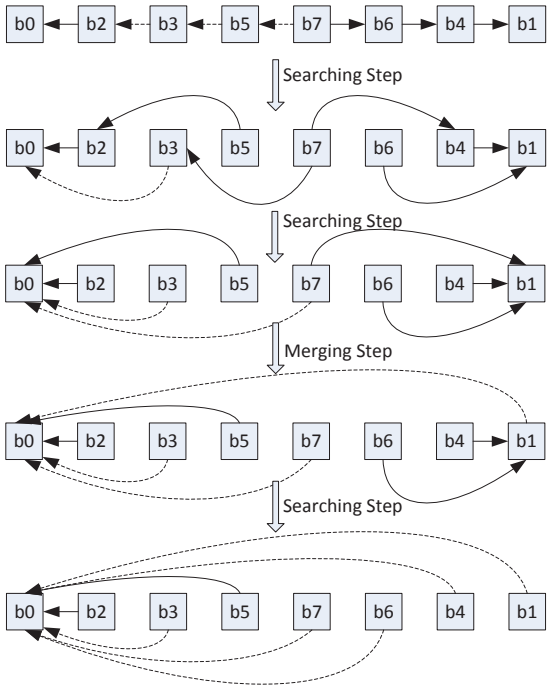


Fig. 7 Searching and Merging

At each processing element P_i, the following code is used to determine if transiting to the MERGE state.

if $((preNodeValue0 == nodeValue0)$ and
$(preNodeValue1 == nodeValue1))$
  if $(nodeType[1:0] == 2'b11)$
    if $(nodeValue0 == nodeValue1)$
      $state =$ MERGE_SN ;
    else $state =$ MERGE;
  else $state =$ DONE;
else $state =$ SEARCH

If the two pointers of the switching node point to the same endpoint, then FSM transits to MERGE_SN state, in which one of two pointers of the switching node will be reset to its initial value; otherwise, the FSM transits to the MERGE state.

MERGE

When the processing element reaches the endpoints in both directions and the two endpoints are different, the merging step will be conducted. As in the initialization step, the node pointer with larger node index is updated with smaller node index number. As shown in Fig. 7, the processing element merges the endpoints of $b7$ overwriting the nodeValue register storing $b1$ to $b0$. We can also see that, after the merging process, the switching nodes previously pointing to node $b1$ need be updated to pointing to $b0$. For the example in Fig. 7, after the merging step, nodes $b6$ and $b4$ need go through searching step again to update their pointers to the representative node $b0$.

For each processing element P_i, the following code is used to update pointers.

if $(nodeValue0[i] < nodeValue1[i])$
  if $(nodeValue1[i][0])$ {
    $nodeValue1[nodeValue1[i]/2] = nodeValue0[i];$
    $nodeS1[nodeValue1[i]/2] = nodeS0[i]\ XOR\ nodeS1[i];$ }
  else {
    $nodeValue0[nodeValue1[i]/2] = nodeValue0[i];$
    $nodeS0[nodeValue1[i]/2] = nodeS0[i]\ XOR\ nodeS1[i];$
}
else if $(nodeValue0[i] > nodeValue1[i])$ {
  if $(nodeValue0[i][0])$ {
    $nodeValue1[nodeValue0[i]/2] = nodeValue1[i];$
    $nodeS1[nodeValue0[i]/2] = nodeS0[i]\ XOR\ nodeS1[i]$ }
  else {
    $nodeValue0[nodeValue0[i]/2] = nodeValue1[i];$
    $nodeS0[nodeValue0[i]/2] = nodeS0[i]\ XOR\ nodeS1[i];$
  }
}
else
  null;

After the merging step, the processing element will notify the other processing elements so that all the other processing elements will transit to the SEARCH state. As shown in Fig. 7, after the searching step, all the switching nodes point to the representative node of this class. The initial state value for the representative node 'b0' of this class is '0', then the state value of all other switching nodes can be determined by the relation value nodeS in parallel. And the switch state values shown in Fig. 7 is exactly the same as those values shown in Fig. 6.

In our design, after all processing elements are in DONE state the mainframe will set the state values of output and input switching nodes.

C. Setting State Values of Output Switching Nodes and Input Switching Nodes

The state values for output switching nodes $outNodeStateValue\left[\frac{N}{2}-1:0\right]$ can be obtained directly from the relation value $nodeS0\left[\frac{N}{2}-1:0\right]$ or nodeS1$\left[\frac{N}{2}-1:0\right]$ as

$$outNodeStateValue[j] = NodeS0[j] \mid NodeS1[j];. \quad (8)$$

After the state values of output switching nodes are determined, the state values of input switching nodes are determined too. According to the symmetric routing constraint, the state value of an input switching node is equal to or opposite to the state value of its corresponding output switching node which depends on the relation of the input/output port index number.

Given permutation pair (k, l), where k is the input port number, and l is the output port number, due to the symmetric self-routing constraint, i.e., Eqn. (4) and (5), we have:

$$inNodeStateValue\left[\frac{k}{2}\right] = \begin{cases} outNodeStateValue\left[\frac{l}{2}\right] & \text{if } k, l \text{ are same parity} \\ \sim outNodeStateValue\left[\frac{l}{2}\right] & \text{if } k, l \text{ are opposite parity} \end{cases} \quad (9)$$

where $k/2$ and $l/2$ give the corresponding input/output switching node index. As we can see, either $port[2i]$ or $port[2i+1]$ can be used to determine the relation between state values of input switching node and its corresponding output switching node. Here we use $port[2i]$ to do the calculation. And $port[2i][0]$ gives the parity of the output port.

// For i=0, 1, …, N/2
if $(port[2i][0])$

$inNodeStateValue[i] = \sim outNodeStateValue\left[\frac{port[2i]}{2}\right];$

else

$inNodeStateValue[i] = outNodeStateValue\left[\frac{port[2i]}{2}\right];$

IV. Experiment Results

We have implemented the Lee's algorithm for finding the switch settings for input/output stages of 8x8 to 32x32 Benes networks in Verilog, simulated and synthesized the designs using Cadence tools. The RTL code is written in parameterized way so that it is easy to expand to larger sizes. In the simulation process, ModelSim is adopted as the simulation tool. For each design, five categories of permutations are used for validation including bit reversal, perfect shuttle, butterfly, matrix transpose, and random permutations. Under each category, one or more different permutations have been tested. In the synthesis process, Cadence Encounter RTL-Compiler is used with TSMC 65nm technology library. All size designs are synthesized under the same settings. The synthesized results are presented below

TABLE II. Timing result

Benes Size	8x8	16x16	32x32
Critical Timing Delay (ps)	8.34E+02	2.30E+03	3.68E+03
Time Complexity $O(log^2 N)$	9	16	25

The timing delay is mainly decided by the time complexity of the algorithm. While the size of the processing element will not affect the timing delay as much as that does to area and power consumption. The complexity of algorithm is determined by the number of searching steps. The simulation results in Table II show that the number of searching steps follows $O(logN)$.

TABLE III. Cell number and Area

Benes Size	8x8	16x16	32x32
Number of Cells	1.81E+03	8.11E+03	3.62E+04

TABLE III. shows the area result in terms of number of cells, the basic design unit used to measure the logic complexity. When the network size is doubled, the number of cells increases by about 4 times. It is clear that in Lee's algorithm, when the network size is doubled, the number of processing elements needed in each stage is doubled. For example, the 8×8 Benes has 4 processing elements and the 16×16 Benes network has 8 processing elements. Besides, the logic complexity of the processing element nearly doubles when the network size is doubled. Overall, the logic complexity of the processing element should be increased by four times when the network size is doubled. This explains the trend of number of cells in TABLE III.

TABLE IV. Power Consumption

Power Type \ Size	8x8	16x16	32x32
Leakage (nW)	9.24E+04	3.86E+05	1.76E+06
Internal (nW)	8.46E+04	3.85E+05	1.69E+06
Net (nW)	2.98E+04	1.42E+05	6.04E+05
Switching (nW)	1.14E+05	5.28E+05	2.29E+06

TABLE IV. shows the power consumption of the design in terms of static (internal) power, dynamic (mainly switching), net and leakage power. Each portion of power increases significantly as the radix of Benes network increases. The power consumption increasing trend is consistent with the increasing trend of number of cells. The switching power is the most significant portion, followed by internal (static) and leakage power which occupies 36%, 28% and 27% of total power, respectively. Together the three portions of power dominate the power consumption at more than 90%.

V. Conclusion

This paper presents the RTL design of a parallel switch setting algorithm in Benes Networks. We have refined the algorithm in data structure and initialization/updating of relation values to make it suitable for hardware implementation. The simulation and synthesis results confirm that the timing, area, and power consumption of the circuit is consistent with the complexity of the Lee's algorithm.

References

[1] K. N. Levitt, M. W. Green and J. Goldberg "A study of the data commutation problems in a self-repairable mutiprocessor." in *Proc. Spring Joint Computer Conf.*, 1968.

[2] Y. Kao, M. Yang, N. S. Artan, and H. J. Chao "CNoC: high-radix Clos network-on-chip," in *IEEE Trans. Computer-Aided Design of Intergrated Circuits and Systems*, vol. 30, no. 12, pp. 1897-1910, Dec. 2011.

[3] H. Liu, L. Xie, J. Liu, and L. Ding, "Application of butterfly Clos-network in network-on-chip," *The Scientific World Journal*, vol. 2014, pp. 1-11, 2014.

[4] A. Joshi, C. Batten, , Y. Kwon, S. Beamer, I. Shamim, K. Asanovic, V. Stojanovic, "Silicon-photonic Clos networks for global on-chip communication," in *Proc. 3rd ACM/IEEE Int'l Symp. Networks-on-Chip (NoCS)*, 2009, pp. 124-133.

[5] T. T. Lee and S. Y. Liew, "Parallel routing algorithms in Benes-Clos networks," in *Proc. 15th INFOCOM*, 1996 vol.1, pp.279-286.

[6] Y. Kai, K. Hamada, Y. Miao and H. Obara., "Design of partially-asynchronous parallel processing elements for setting up Benes networks in O(log2N) time," in *Proc. Int'l Conf. Photonics in Switching*, 2009, pp. 1-2.

[7] Y. Yeh and T. Feng, "On a class of rearrangeable networks" in *IEEE Trans. Comput.*, vol. 41, no. 11, pp. 1361-1397, Nov. 1992.

[8] A. Waksman, "A permutation network," *J. Ass. Comput. Mach.*, vol. 15, pp. 159-163, Jan. 1968.

[9] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Benes permuation network", *IEEE Trans. Computer*, vol. c-31, no. 2, pp. 148-154, Feb. 1982.

[10] K. N. Levitt, M. W. Green, and J. Goldberg, "A study of the data commutation problems in a self-repairable multiprocessor", in *Proc. Spring Joint Computer Conf.*, 1968, vol. 32, pp. 515-527.

[11] T. T. Lee, S. Y. Liew, "Parallel routing algorithms in Benes-Clos networks", *IEEE Trans. Commun.*, vol. 50, no. 11, pp. 1841-1847, Nov. 2002.

[12] H. Richter, "Real-time interconnection network for single-chip many-core computers," in *East-West Design & Test Symp.*, 2013, pp.1-4.

[13] H. Moussa, O. Muller, A. Baghdadi, M. Jezequel, "Butterfly and Benes-based on-chip communication networks for multiprocessor turbo decoding," in *Proc. DATE*, 2007, pp.1-6.

[14] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153-161, Feb. 1971.

Traceability Acquisition Method for Network Security using Multiple Encryption and Decryption of the Tag in Packet

Kento Masukawa, Kenichi Takagiwa, Tadanori Matsui, Hiroaki Nishi
Graduate School of Science and Technology, Keio University, Japan
[+]{masukawa, takagiwa, matz, west}@west.sd.keio.ac.jp
Regular Research Paper

Abstract – *Multi-factor authentication is used to improves the existing vulnerability of traditional password authentication, such as biometric and one-time password token. However, these methods foist a burden on the user. We propose the traceability acquisition method in a network for the user authentication, which runs only on the routers; thereby it achieves secure and authenticated communication between a user and a server application without any burden on the users and applications. In this method, "tracer tags" is introduced for attaining packet traceability of a packet and the tags are encrypted with unique router's key. Recipient makes sure that packet has delivered through the expected routing path by decrypting a tracer tag. The Proposed method achieved secure and authorized communication using packet traceability without any additional effort to users The proposed method is evaluated in a virtual environment. By using the proposed method, routing throughput decreased only by 3% compared with normal routing.*

Keywords: network security, routing information, multi-factor authentication, multiple encryptions, tag information.

1 Introduction

The combination of username and text password is the most common authentic method as the user authentication of a Web service. The text password authentication should choose a strong password to resist brute-force or dictionary attack [1]. However, a strong password which combines multiple random alphabets, numeric and special characters are difficult to memorize for a human being. Many users are using the easy-to-remember weak passwords. Reusing the password across multiple Website also promotes the vulnerability of text password authentication [2]. Florencio and Herley indicated that user reuses the same password across 3.9 Web services on average [3]. Reuse of password could expand the damage to multiple Web service caused by password leak from one of Web services. This attack is called Password Reuse Attack.

In addition, Password Stealing Attacks (PSA) must be taken into consideration. The phishing site is the most common as a method of PSA. According to the APWG report [4], the total number of phishing attacks was observed in the 4th season of 2016 was a review 158,574.The email address has been used for user identification in many Websites. Therefore, password theft leads to crack user accounts.

Some researchers focus on multi-factor authentication to provide more reliable user authentication. NIST announced SP800-63-2 [5] for guidelines of online authentication; there are the following three factors.

- SYK(Something You Know)
- SYH(Something You Have)
- SYA(Something You are)

Text password authentication is categorized in the SYK. One-time password token is SYH. Biometric authentication is SYA. Multi-factor authentication utilizes the SYH or SYA beside the password. Although these multi-factor authentications improve security by convoluted user authentication, it incurs expensive to introduce compared with password authentication [6]. These methods also need new processes to the users such as preregistering biometric information and creating a one-time password, and burden on users is increased. Therefore, people who are not aware of security do not use multifactor authentication. For protecting private information, there is a need for a method to

improve security using multifactor authentication without increasing the user burden.

In this paper, we propose the traceability acquisition method in a network and apply it to the multifactor authentication, which requires no additional action to users. The tracer tag is stored in the packet, which is encrypted with each router during packet forwarding. The server acquires a traceability by analyzing the tag. This method requires special router which can analyze traffic and modify packet on the fly. Sevice-oriented Router(SoR) [7], a router is capable of analysis and processing of the packet, is used for this research. SoR is used for storing and encrypting tags in the packet.

2 Related Work

K.-P. Yee and K. Sitaker [8] propose Passpet, a password management tool that improves both the convenience and security of Website logins. By using password hashing, the tool manages multiple accounts by turning a single memorized password into a different password for each account. User-assigned site labels help users securely identify phishing sites. Password-strengthening measures defend against dictionary attacks. They propose new improvements to these techniques and integrate techniques into a single tool.

Some researchers focus on multi-factor authentication rather than text password authentication to provide more reliable. Multi-factor authentication depends on three factors [5], SYK e.g. password, SYH e.g. token, SYA e.g. biometric.

H-M. Sun, Y.-H. Chen and Y.H. Lin [9] propose a user authentication protocol named oPass, which leverages cell phones and short message service to protect password stealing, and password reuse attacks. Unlike generic web logins, oPass utilizes a user's cell phone as an authentication token and SMS as a secure channel. oPass requires each participating website possesses a unique phone number. oPass program on phone stores user's long-term password. The server on oPass requests for the user's account id and phone number, instead of a password. In login procedure, the user enters the long-term password in a phone; the oPass program generates a one-time password and sends a login SMS securely to the server. oPass does not require users to type the password into an untrusted web browser. Users only need to remember a long-term password for login on all Websites.

FIDO (Fast IDentity Online) Alliance [10] develop technical specifications that define an open, scalable, interoperable set of mechanism that reduce the reliance on passwords to authenticate users. In 2014, FIDO Alliance published FIDO1.0 [11] Specifications. UAF protocol has been described in FIDO1.0. UAF (Universal Authentication Framework) uses biometric of user and device with UAF stack installed. The user registers their device to the online service by selecting a local authentication mechanism such as swiping a finger, looking at the camera, speaking into the mic, etc. The user simply repeats the local authentication action whenever they need to authenticate to the service. The user no longer needs to enter their password when authenticating from that device.

As an above study, multi-factor authentication has been used in improving the security. However, it incurs expenses to introduce [6]. Also, multi-factor authentication requires new actions to the user for the operation such as scanning of the fingerprint, checking one-time password token. Moreover, the burden on users is increased. Furthermore, the user must manage the token and the scanning device. Therefore, the low security-conscious user hesitates to implementation.

The processing by infrastructure can solve these problems. Regarding the method without the burden of user, Rajitha et.al proposed [12] a hop-by-hop routing protocol that provides hop-by-hop data encryption using functions of SoRs, which offers security, privacy, and integrity. SoR [7] [13] is a router, which can inspect the data contents in a packet up to OSI layer 7. A SoR has a high-throughput database and can analyze all transactions on its interfaces.

However, routing path information has not been considered with this protocol. This information may vary according to the access points, and can be used for specifying a user. The contribution of this research is a novel user authentication based on routing path information obtained through the infrastructure such as routers using data encryption.

3 Traceability acquisition method

"Tracer tag" is introduced to store the route information in a packet. The tracer tag is encrypted

by the unique cryptographic key in the SoR. Since all SoRs in the route execute encryption process, the tracer tag is encrypted multiple times according to the order of the forwarding path of the packet. The packet acquires the traceability when the received packet is successfully decrypted according to the reversed order of the forwarding path.

In this method, SoR updates the tracer tag when the packet is forwarded because traceability of the packet is guaranteed by multiple encryptions on the router and decryption on the receiver. The proposed method allows enhancing security without requiring extra hassle for users in terms of introduction and operation compared to another method. This authentic method needs pre-registration of places, so accesses from home and workplaces are assumed.

We believe the difficulty of unauthorized access is raised by the increase of required information. In the password certification, two components, username, and password are needed. In the proposed method, adding to username and password, route information from the user to server and decryption keys on the router in the route are needed.

The proposed algorithm has three phases: addition of tracer tag, encryption on routers, and decryption to analyze route information.

3.1 Insertion of tracer tag

SoRs of the proposed method can inspect the data contents to OSI layer 7, and process data in packets. In this research, the tracer tag is defined as a data field in the packet to store route information. The first 16 bytes of the data field is set as tracer tag. When a packet is sent, 16-byte data is inserted into the data field. This research uses 16-byte password which is shared with sender and receiver for simplicity.

3.2 Encryption function introduced router

The SoRs do not only routing but also encryption process. An encryption module for SoR is implemented by C language.

Tracer tag is multi-encrypted with holding fixed field. Since all routers execute encryption process, the processing speed of encryption affects total delay of the system. AES (Advanced Encryption Standard) [14] is selected because public key encryption consumes more computation power. The length of AES key is 128 bits, 192 bits, and 256bits.

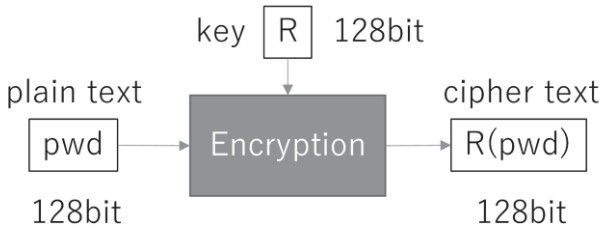


Fig. 1 The encryption flow

Bisclique attack [15] lessens the safety to 126.1 bits, 189.7 bits, and 254.4 bits. NIST recommended [16] that the length of AES key should be 112 bits or more to ensure code safety when the system used until 2030. This AES algorithm with 128 bits is in that scope. The proposed method uses the EVP (The Digital EnVeloPe Library) by The OpenSSL Project [17] for implementation of AES.

Fig. 1 shows the flow of encryption process in proposed method. 128-bit data in the tracer tag is encrypted by the unique 128-bit key on every router. There are different kinds of block cipher mode for AES since AES is one of the block encryption. ECB is chosen for this research since tracer tag is fixed length and the simplest one.

This algorithm uses destination port number to judge whether encryption process executed or not.

Every SoR has different 128-bit encryption key which is used for encrypting the data of tracer tag. The encryption process causes the change of UDP checksum because tracer tag is included in the UDP data field. UDP checksum is recalculated after encryption.

3.3 Analysis of routing information

The receiver validates the path information by preregistered user information and path information. The user information such as password and path information are linked with port number and registered to the receiver. This route information is acquired by traceroute function, and it is registered with a password when the first connection was established between sender and receiver. The receiver receives the decryption key of every router in the route. The receiver starts to analyze UDP data when the data arrived in the predefined port number. Firstly, it identifies the sender by user information in the packet. Secondly, The tracer tag is decrypted according to the forwarding order from receiver to sender. After decryption process, receiver validates

the password. If the validation is correct, it means the packet passed through correct route.

Fig. 2 shows the flow of proposed algorithm. "pwd" is the password in trace tag, "R1" and "R2" are encryption process on router 1 and router 2. "$R1^{-1}$" and "$R2^{-1}$" means decryption process. The data "pwd" from sender arrives as "R2· R1· pwd" on the receiver. Receiver assumes the packet pass through router #1 and router #2 after that receiver decrypts the packet according to the reversed order. Since it can get "pwd" correctly, the receiver can judge that packet passed through correct route.

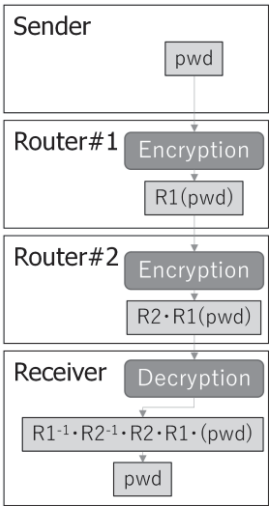


Fig. 2 flow of getting traceability

4 Evaluation

This section explains about the environment of evaluation. The virtual network includes the virtual bridges in the virtual machines. Virtual Box [18] is used as the virtual machine, which is an open source virtualization software. Table. 1 shows information about the host machine and virtual machine.

Constructed virtual network is shown in Fig. 3. Intel PRO/1000 MT Desktop is selected as a network adapter. Three virtual machines are used as routers, and another three is used as nodes. Every router routes packet by a static routing table in the router and contains 2 hops ahead node information from the node. The software routers of this environment implemented as a variety of the SoR.

Table. 1 Evaluation environment

CPU	Intel Core i7-4790 3.60GHz
Main Memory	16.0 GB
Host OS	Windows 8.1 Pro
Virtual Box	VirtualBox 5.06 r 103037
Virtual OS	CentOS 7.2. 1511
Virtual CPU	1 CPU
Virtual Memory	768 MB
VM Linux kernel	3.10.0-229

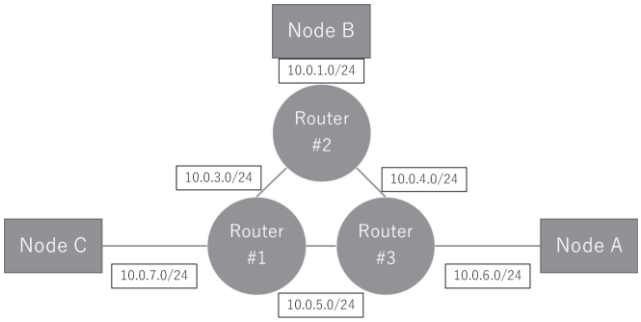


Fig. 3 Assumed virtual network model

4.1 Experimental environment

In this paper, traceability is definable as a function that the routed path of a packet can be confirmed after receiving the packet, and the traceability is achieved by using multiple encryption and decryption in order. Therefore, the multiple encryption processes decrease the performance of concerning routers. For evaluating the effect of the encryption process, a software router with encryption was compared to IP forwarding function of Linux kernel and software router.

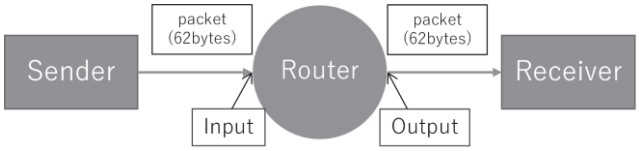


Fig. 4 Model of encryption process load test

For measuring the encryption throughput and latency, a simple model shown in Fig. 4 was constructed. The sender sends the packet with tracer

tag to the receiver via the router. Input and output interface are monitored for measuring the processing delay of the routing process. The size of the packet is fixed as 62 bytes. Tracer tag is handled as a part of UDP data when encryption process is not executed.

Sender stores the password in the 16 bytes tracer tag field and then send the packet to the destination to UDP port 12333 of the receiver. The number of sending packet is varied from 1 to 10,000,000. Data transfer rate is 2.3MB/s. The number of a passing packet of the router and the passing time were monitored at input and output interface by using tcpdump.

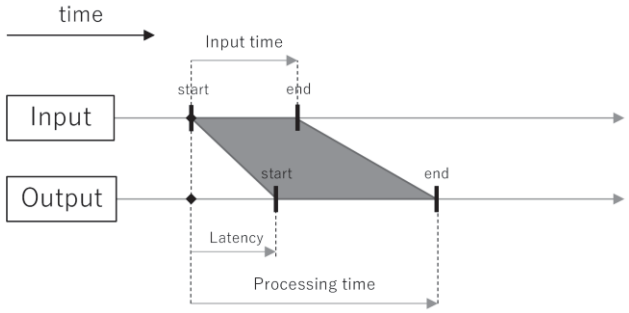


Fig. 5 Evaluation value got from input and output

The number of the passing packets, the start and end time of the evaluation, and starting and ending time of output are measured by monitoring interfaces on the router. The latency is calculated by using the measured values as shown in Fig. 5. The time when the packet was input, the latency of process and total processing time are used for this evaluation. The total data size was calculated by multiplying the number of output packet by 62 bytes, which are packet size. Then, throughput can be calculated by dividing the total amount of data by total processing time. We designed a middleware of a software router to implement the proposed algorithm on it. General Linux with kernel IP forwarding, the software router without the proposed algorithm, and the software router with the proposed algorithm were compared to evaluate routing performance.

Latency of these three models was shown in Table. 2. The average and standard deviation of the latency of Linux kernel are 1/6 and 1/50 of other software routers. Moreover, the throughput of Linux kernel is 3.0 times higher than others.

Table. 2 Latency comparison

	Linux kernel	Software router w/o encryption	Software router w/ encryption
Avarage	0.035ms	0.270ms	0.319ms
Standard deviation	1. 13μs	80. 8μs	77. 9μs

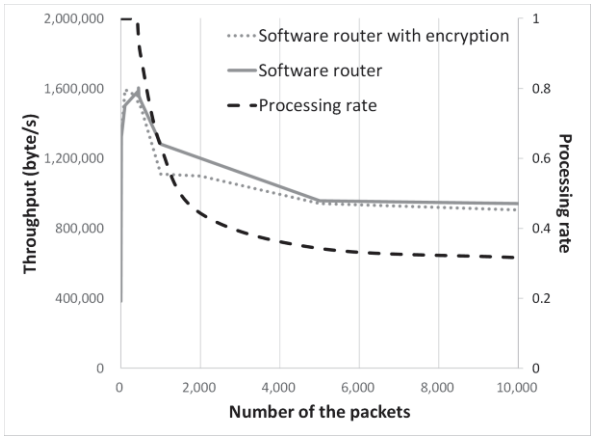


Fig. 6 Throughput and rate of packet processing

Designed software router routes packets by copying them into the buffer, which causes degradation of performance. From here, to evaluate the effect of additional load of the encryption process, proposed router is compared with existing router.

Fig. 6 shows the variation of throughput of the encryption process. The value increases rapidly until the number of the input packet becomes 500, and it decreases after that in any cases. The packet loss occurs after the number of the packet becomes 500, which means software router has a limitation on its packet buffers and the capacity of the buffer was exceeded after the number of the packet becomes 500. When the number of the input packet becomes more than 5,000, the throughput converges because a load of packet control becomes constant.

Table. 3 shows the average and maximum throughput after the number of the input packet becomes 5,000. Each value of existing router is higher than that of proposed router. The throughput of proposed router decreases by 3 % compared with the that of an existing router because of the encryption process delay. The delay is caused by the packet preprocessing, encryption, and rewriting UDP checksum.

Table. 3 Comparison of performance by throughput

	Software router with encryption	Software router	Reductionrate by encryption
Maximum throughput (Mbps)	1.56	1.60	2. 82%
Avarage throughput (Mbps)	0.88	0.90	3. 15%

4.2 User authentication using routing information

User authentication using routing information is implemented and evaluated according to the proposed method. Node B is a server (receiver), Node A and Node C are users (sender), and routers execute encryption process.

4.2.1 Traceability acquisition method

Node A and Node C send the same UDP data to Node B, and authentication process is confirmed. In this experiment, the route information only from Node C is registered and approved. Namely, accesses from the other nodes are refused.

Node B processes UDP data comes from port number 12333 and distinguish which the packet should be encrypted or not. After that, the packet is forwarded to Node C according to the registered route information. After that, the tracer tag is decrypted according to the order of registered routing information.

As a result, nonetheless, Node A and Node B send the same password from the same port, the multiple encryption and decryption process guaranteed the packets were delivered through the correct and hopeful route and were generated by different users despite the condition of using same passwords. This experiment confirmed that proposed method could acquire traceability by adding route information in a packet.

4.3 Authentication performance evaluation

Packet forwarding performance of proposed authentication using route information and existing password authentication was compared. Response time is the duration time from the time when 16 bytes password UDP data was sent to the time when a response from the server was received. Node B is a user, and Node C is a server, the date passed through router #1 and router #2 as shown in Fig.3.

Table. 4 Comparison of response

	Text password	Proposed authentication
Response time	1.12ms	2.03ms

In existing password authentication, the packet is transferred by the router and reaches the server, nad then password certification is executed. On the contrary, in the proposed method, the password is repeatedly encrypted on every router, and password certification is executed after the decryption process on the server. Soon after the password certification is succeeded, the response packet is sent to the user.

Response time is measured by monitoring network interface of the user by using tcpdump. Table 4 shows the comparison of the response time of user certification. The response time of user authentication using router information increases 0.909 milliseconds compared with the existing password certification. This difference means the processing delay of encryption on router and decryption on the server. In this experiment, two routers are passed, and it means 2 set of encryption and decryption were executed. The processing time per 1 hop is 0.454 milliseconds. Since the Time To Live, which is the limit of a number of hops, is set to 64 on the CentOS7, the maximum delay could become 29 milliseconds by adopting proposed method.

In terms of user load, the user only inserts 16 bytes password in a packet of a stream in any cases. This means the user load is the same even using the proposed user certification. Compared with the one-time password, token control, and biological certification, it is easy to implement proposed method that adds the tracer tag to the packet because this can introduce by software.

5 Conclusion

In this study, we proposed traceability acquisition method in a network and new user authentication using routing path information. The tracer tag is inserted in the packet, and the tracer tag is encrypted at each router on its forwarding path. The recipient decrypts the tracer tag to validate the

path of the packet. The router in the proposed method is evaluated by using VirtualBox. The throughput of a router with the proposed method decreased by 3% compared with a router without the proposed method. However, it achieves traceability of packet. User authentication leveraging packet traceability can enforce security without increasing the user's burden. The response time of the proposed user authentication was about 0.454ms increase in each through a router.

ACKNOWLEDGEMENT

This work was partially supported by the funds of SECOM Science and Technology Foundation.

6 References

[1] M.Bishop, "Password Management," Proceedings of Compoom Spring '91: Digest of Papers page 167-169, 1991.

[2] Tarwireyi, P., Flowerday, S., Bayaga, A. "Information Security Competence Test with Regards to Password Management," Information Security South Africa(ISSA), 2011.

[3] D. Florencio, C. Herley, "A large-scale study of web password habits," WWW '07:Proc. 16th Int. Conf. World Wide Web., 2007.

[4] APWG, "Phishing Activity Trends Report," APWG, 2016.

[5] NIST, "Special Publication 800-63-2," NIST, 2013.

[6] L. O'German, "Comparing passwords, tokens, and biometrics for user authentication," Proc.IEEE, vol.91, no. 12, pp.2021-2040, 2003.

[7] Koichi Inoue, Dai Akanishi, Michihiro Koibuchi, Hideyuki Kawashima, Hiroaki Nishi, "Semantic router using data stream to enrich services.," 3rd International Conference on Future Internet CFI 2008 Soul, June 2008.

[8] Ka-Ping Yee, Kragen Sitaker, "Passpet: convenient password management and phishing protection," SOUPS '06: Proc. 2nd Symp. Usable Privacy Security, New York, 2006.

[9] H.M. Sun, Y.H. Chen, Y.H. Lin, "oPass: A User Authentication Protocol Resistant to Password Stealing and Password Reuse Attack," IEEE Transaction on Information Forensics and Security, vol.7, No.2, 2012.

[10] FIDO Alliance, "FIDO Alliance," FIDO Alliance, 2016. [Online]. Available: https://fidoalliance.org.. [Accessed 04 04 2016].

[11] FIDO Alliance, "FIDO NFC Protocol Specification v1.0", https://fidoalliance.org/specs/fido-u2f-nfc-protocol-id-20150514.pdf. [Accessed 4 4 2016].

[12] R. Tennekoon, J. Wijekoon, E. Harahap, H. Nishi, E. Saito, S. Katsura, " Per Hop Data Encryption Protocol for Transmission of Motion Control Data Over Public Networks," Advanced Motion Control(AMC), 2014.

[13] K. Takagiwa, S. Ishida, H. Nishi, "SoR-based Programmable Network for Future Software-Defined Network," IEEE 37th Annual Computer Software and Applications Conference, 2013.

[14] NIST, "FIPS PUB 197," NIST, 2001.

[15] A. Bogdnaov, D. Khovatovich , C. Rechberger, "Bisclique Cryptanalysis of the Full AES," ASIACRYPT 2011, 2011.

[16] NIST, "NIST SP 800-57(Recon for Key Management)," NIST, 2007.

[17] OpenSSL Project, "OpenSSL," 2016. [Online]. Available: https://www.openssl.org. [Accessed 04 04 2016].

[18] Oracle, "Oracle VM VirtualBox," 2016. [Online]. Available: https://www.virtualbox.org. [Accessed 04 04 2016].

Node-Independent Spanning Trees in Gaussian Networks

Z. Hussain[1], **B. AlBdaiwi**[1], **and A. Cerny**[2]
[1]Computer Science Department, Kuwait University, Kuwait
[2]Department of Information Science, Kuwait University, Kuwait

Abstract— *Gaussian network is known to be an alternative to toroidal network since it has the same number of nodes with less diameter, which makes it perform better than toroidal network. Spanning trees are said to be independent if all trees are rooted at the same node r and for any other node u, the nodes of the paths from r to u in all trees are distinct except the nodes r and u. In this paper, we investigate the problem of finding node independent spanning trees in Gaussian networks.*

Keywords: Circulant Graphs, Gaussian Networks, Spanning Trees, Independent Spanning Trees, Fault-Tolerant Routing.

1. Introduction

The topology of an interconnection network plays a major role in achieving high performance computing in parallel systems. There are many varieties of these interconnection networks and some of them are popular such as hypercube, generalized hypercube, mesh, torus, De Bruijn, REFINE, and RMRN networks [2][3][4][5][6][7][9]. An efficient interconnection topology called Gaussian network has been studied in [8][15][16]. The studies show that the Gaussian network can be a better alternative to the torus network since it has the same number of nodes but with less diameter. This network is briefly reviewed in Section 2.

In parallel computing and distributed systems, a network can be represented as a graph $G(V,E)$ where V is the set of nodes and E is the set of edges. These nodes and edges represent processors and communication links between the processors in the network, respectively. A path from node s to node d in the graph is a sequence of edges, which connects a sequence of nodes from s to d. Two such paths are said to be independent if their nodes are different except the end nodes s and d, i.e. the intermediate nodes in the first path are distinct form the intermediate nodes in the second path. A spanning tree is a connected loop-free subgraph of graph G containing all the nodes of graph G. Spanning trees rooted at node r are said to be independent if the paths from r to any other node u in all trees are independent.

Node independent spanning trees used to resolve important issues in network applications such as fault-tolerant broadcasting [11][13] and secure message distribution [17][18]. These applications are briefly described below:

- Consider that there exist t node independent spanning trees rooted at node r in a network N. Assume that the network N contains at most $t-1$ faulty nodes. Then, r can broadcast a message to every non-faulty node u in the network N with the existence of $t-1$ faulty nodes. Since the number of faulty nodes is less than t, then at least one of those t node disjoint paths from r to u is fault free. Thus, every non-faulty node in the network N would receive the broadcasted message from r if all t node independent spanning trees are used to broadcast the message.
- Node independent spanning trees could be used to secure message distribution as follows. A message can be divided into t packets where each packet is sent by node r through a different spanning tree to its destination. Thus, each node in the network receives at most one of the t packets whereas the destination node receives all the t packets [14][18][19].

In [1], we have constructed two edge-disjoint node-independent spanning trees in dense Gaussian network, in which the network contains the maximum number of nodes for a given diameter k [16], where the depth of each tree is $2k$, $k \geq 1$. We also designed algorithms that can be used in fault-tolerant routing or secure message distribution where the source node in these algorithms is not restricted to a specific node; it could be any node in the network.

In this paper, we investigate the problem of finding node independent spanning trees in Gaussian networks where the trees are not necessarily edge-disjoint. This paper is organized as follows. The Gaussian network is reviewed in Section 2. Section 3 presents the algorithms that illustrate the construction of node independent spanning trees in Gaussian networks. The communication overhead and the amount of work to construct the trees are discussed in Section 4. The paper is concluded in Section 5.

2. Background

Gaussian networks are 4-regular symmetric networks where each node in the network has 4 neighbors. These networks are based on quotient rings of Gaussian integers $\mathbb{Z}[i] = \{x + yi \mid x, y \in \mathbb{Z}\}$ where $i = \sqrt{-1}$ [10]. $\mathbb{Z}[i]$ is a Euclidean domain. The nodes of the network are elements of the residue class modulo some $\alpha \in \mathbb{Z}[i]$. The total number of nodes in the network is known as norm, $N(\alpha)$, of the Gaussian integer $\alpha = a + bi$ and is equal to $a^2 + b^2$. Various representations of these residue classes are given in [12]. Each node in the network is labeled as $x + yi$. The nodes A and B are said to be adjacent, i.e. neighbors, if and only if

$(A-B)\ mod\ \alpha$ is equal to ± 1 or $\pm i$. As described in [15], if $N(\alpha)$ is odd, for each node in the grid, the number $D(s)$ of nodes at distance s such that $0 \leq s \leq k$ and $t = \frac{a+b-1}{2}$ is:

$$D(s) = \begin{cases} 1 & if\ s = 0 \\ 4s & if\ 0 < s \leq t \\ 4(b-s) & if\ t < s < b \end{cases} \quad (1)$$

Further, when $N(\alpha)$ is even, the number $D(s)$ of nodes at distance s such that $0 \leq s \leq k$ and $t = \frac{a+b}{2}$ is:

$$D(s) = \begin{cases} 1 & if\ s = 0 \\ 4s & if\ 0 < s < t \\ 2(b-1) & if\ s = t \\ 4(b-s) & if\ t < s < b \\ 1 & if\ s = b \end{cases} \quad (2)$$

when $0 < a = b$, the distance distribution of the graph G_{b+bi} is as follows:

$$D(s) = \begin{cases} 1 & if\ s = 0 \\ 4s & if\ 0 < s < b \\ 2b-1 & if\ s = b \end{cases} \quad (3)$$

Based on the above, the diameter of the network is k, which is equal to b when $N(\alpha)$ is even and to $b-1$ when $N(\alpha)$ is odd. Figure 1 shows an example of Gaussian network generated with $\alpha = 3 + 4i$.

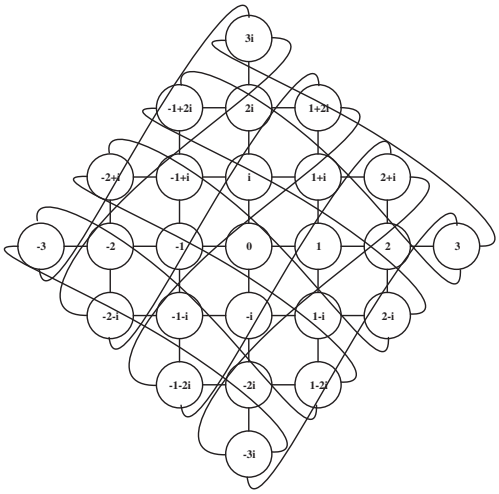


Fig. 1: Gaussian network generated with $\alpha = 3 + 4i$

The wrap-around links can be obtained by tiling the Gaussian network on an infinite grid. For example, Figure 2 shows the Gaussian network generated with $\alpha = 2 + 3i$ and the nodes' wrap-around links are illustrated based on tiling the network. Consider the node $1+i$ where its -1 edge and $-i$ edge are connected to the nodes i and 1, respectively, which are within the basic grid. However, the $+1$ edge should be connected to node $2 + i$, which is not within the basic grid. Thus, this edge is considered as a wrap around link and it is connected to node $-2i$, which is an equivalent to node $2 + i$ modulo $2 + 3i$. Similarly, $+i$ edge of node $1 + i$ is connected to node $1 + 2i$, whose corresponding node in the basic grid is $-1 - i$.

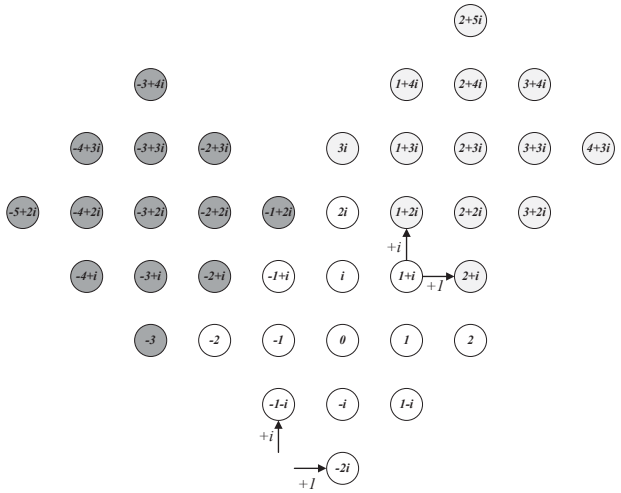


Fig. 2: Tiling Gaussian network generated with $\alpha = 2 + 3i$

In this paper, we deal with dense diameter-optimal graph, which is isomorphic to the Gaussian network $G(\alpha_k)$, where $\alpha_k = k + (k+1)i$, since GCD$(k, k+1) = 1$. We denote $G_k = (V_k, E_k) = G(\alpha_k)$, $k \geq 1$.

3. Node Independent Spanning Tree Construction

In this section, we describe how to construct four node independent spanning trees in Gaussian network G_k. We give algorithms that construct four different spanning trees and then, from the figures, we show that these trees are independent and the exact proof of the independence of these trees will be in an extended version of this paper.

Algorithm 1 constructs four paths of length 1 where each path connects the node 0 to one of its neighbors (children of *root*), which is considered as the first step for constructing the four node independent spanning trees. The variables used in the following algorithms are described as follows. *treeNo* = 1, 2, 3, and 4 determines the first, second, third, and fourth node independent spanning tree, respectively. *root* is the root node and *current* is the current node where both being of the form $x + yi$; *root*.x or *root*.y (*current*.x or *current*.y) describe the coordinates of the node in the network. The *root*.Child1, *root*.Child2, *root*.Child3, and *root*.Fourth are the links, in respective order, to the first, second, third, and fourth children of the node *root*. The *current*.Child1, *current*.Child2, and *current*.Child3 are the links to the first, second, and third children of node *current*, respectively, and *current*.Parent is the link to the parent node of the node *current*. The network generator is $\alpha = k + (k+1)i$; k is the network diameter in

dense Gaussian networks. In all algorithms, all operations of $+$ and $-$ are done $mod\ \alpha$.

Algorithm 1 allTrees: Construct four node independent spanning trees based on a network generator $\alpha = a + bi$

1: *root*.x $\leftarrow$ 0
2: *root*.y $\leftarrow$ 0
3: *root*.Child1 $\leftarrow$ *root*.x + 1
4: *root*.Child2 $\leftarrow$ *root*.y + i
5: *root*.Child3 $\leftarrow$ *root*.x $-$ 1
6: *root*.Child4 $\leftarrow$ *root*.y $-$ i
7: send through $+1$ packet (*root*, *root*.Child1, 1)
8: send through $+i$ packet (*root*, *root*.Child2, 2)
9: send through -1 packet (*root*, *root*.Child3, 3)
10: send through $-i$ packet (*root*, *root*.Child4, 4)

Algorithm 2 does the configuration for a node when it receives the packet (*parent*, *current*, *treeNo*). For example, it initializes the current node and it sets its parent node. After that, based on the value of *treeNo*, it calls the corresponding function to construct the targeted tree.

Algorithm 2 Node initialization based on the received packet (*parent*, *current*, *treeNo*)

1: $k \leftarrow b - 1$
2: *current*.Parent $\leftarrow$ *parent*
3: *current*.Child1 $\leftarrow$ Nil
4: *current*.Child2 $\leftarrow$ Nil
5: *current*.Child3 $\leftarrow$ Nil
6: **if** *treeNo* = 1 **then**
7: call Tree1(*parent*, *current*)
8: **else if** *treeNo* = 2 **then**
9: call Tree2(*parent*, *current*)
10: **else if** *treeNo* = 3 **then**
11: call Tree3(*parent*, *current*)
12: **else**
13: call Tree4(*parent*, *current*)
14: **end if**

Algorithm 3 sets the child nodes and forwards the packet to them based on the directions corresponding to the first spanning tree. The first tree uses the following edges. $2k$ edges are in $+1$ direction where k of them on the path from node 0 to node k and the other k are from node k to node ki and from node $(-1+ji)$ to node (ji) for $j = 1, 2, \ldots, k-1$. Further, k^2 edges are used in $-i$ direction from node $(m - ni)\ mod\ \alpha$ to node $(m-(n+1)i)\ mod\ \alpha$ for $m = 1, 2, \ldots, k$ and $n = 0, 1, \ldots, k-1$. Also, there are k^2 edges are used in $+i$ direction from node $(m+ni)\ mod\ \alpha$ to node $(m+(n+1)i)\ mod\ \alpha$ for $m = 1, 2, \ldots, k$ and $n = 0, 1, \ldots, k-1$. Thus, a total of $2k^2 + 2k$ edges are used in the first spanning tree, which are sufficient to form a spanning tree of all $2k^2+2k+1$ nodes of the network.

From figures 3, 4, 5, and 6, it is clear that the four spanning trees are symmetric. Also, it is obvious that the second tree in Figure 4 is a one rotation to the counter clockwise of the first tree in Figure 3. Similarly, the third and fourth trees as seen in figures 5 and 6, respectively, are two and three rotations to the counter clockwise of the first tree, respectively. Thus, it follows that the algorithms 4, 5, and 6 are similar to the Algorithm 3 except that the direction of sending the packet is set according to the rotation based on its corresponding tree.

Note that, the intersected edges between the first and third (also, second and fourth) spanning trees are used in opposite directions. That is, the path between any nodes u and v in the first spanning tree is independent from the third spanning tree. The similar argument applies to the second and fourth spanning trees. Thus, the first and third (second and fourth) spanning trees are independent.

Moreover, the first spanning tree uses mostly $\pm i$ (vertical) edges, which are not used in the second tree, while the second spanning tree uses mostly ± 1 (horizontal) edges, which are not used in the first tree. Also, note that the intersected edges between the first and second spanning trees are used in opposite directions. That is, the path between any nodes u and v in the first spanning tree is independent from the second spanning tree. The similar argument applies to the third and fourth spanning trees. Thus, the first and second (third and fourth) spanning trees are independent.

A similar to the above argument can be applied to show that the first and the fourth (the second and the third) spanning trees are independent. Thus, we get four independent spanning trees.

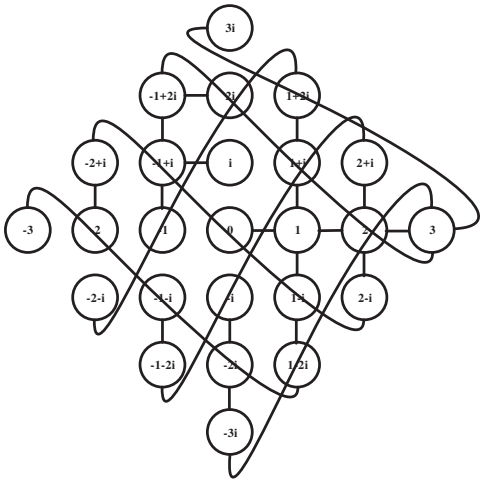


Fig. 3: First spanning tree

After constructing the trees, we can execute tree broadcasting to send a message from the root node to every other node in the network in $2k$ steps.

Algorithm 3 Tree1(*parent*, *current*): Invoked by a function call in Algorithm 2

```
1: if current.y = 0 and current.x > 0 and current.x ≤ k then
2:    current.Child1 ← current.y−i
3:    send through −i packet (current, current.Child1, 1)
4:    current.Child2 ← current.x+1
5:    send through +1 packet (current, current.Child2, 1)
6:    current.Child3 ← current.y+i
7:    send through +i packet (current, current.Child3, 1)
8: end if
9: if parent.Child1 = current then
10:   current.Child1 ← current.y−i
11:   if current.y ≥ i and current.y ≤ (k−1)i and current.x = −1 then
12:      current.Child2 ← current.x+1
13:   end if
14:   if current.Child1.y ≠ 0 then
15:      send through −i packet (current, current.Child1, 1)
16:   end if
17: end if
18: if parent.Child3 = current then
19:   current.Child3 ← current.y+i
20:   if current.Child3.y ≠ −i then
21:      send through +i packet (current, current.Child3, 1)
22:   end if
23: end if
```

Algorithm 4 Tree2(*parent*, *current*): Invoked by a function call in Algorithm 2

```
1: if current.x = 0 and current.y > 0 and current.y ≤ ki then
2:    current.Child1 ← current.x+1
3:    send through +1 packet (current, current.Child1, 2)
4:    current.Child2 ← current.y+i
5:    send through +i packet (current, current.Child2, 2)
6:    current.Child3 ← current.x−1
7:    send through −1 packet (current, current.Child3, 2)
8: end if
9: if parent.Child1 = current then
10:   current.Child1 ← current.x+1
11:   if current.x ≥ −k + 1 and current.x ≤ −1 and current.y = −i then
12:      current.Child2 ← current.y+i
13:   end if
14:   if current.Child1.x ≠ 0 then
15:      send through +1 packet (current, current.Child1, 2)
16:   end if
17: end if
18: if parent.Child3 = current then
19:   current.Child3 ← current.x−1
20:   if current.Child3.x ≠ 1 then
21:      send through −1 packet (current, current.Child3, 2)
22:   end if
23: end if
```

4. Construction Complexity

Each tree needs $2k$ parallel construction steps. In the following two subsections we will derive the communication overhead and the amount of work to construct a single tree. In the third subsection, we will derive the total complexity to construct all four trees.

4.1 Communication Complexity

We will enumerate the construction steps from 0 to $2k-1$. The number of messages generated in the i^{th} step of communication is:

$$Comm(i) = \begin{cases} 2i+1, \ 0 \le i \le k \\ 2(2k-i)+1, \ k+1 \le i \le 2k-1 \end{cases} \quad (4)$$

Thus, the total messages generated in each tree construction is:

$$= \sum_{i=0}^{2k-1} Comm(i) \quad (5)$$

$$= \sum_{i=0}^{k}(2i+1) + \sum_{i=k+1}^{2k-1}(2(2k-i)+1)) \quad (6)$$

$$= \sum_{i=0}^{k} 2i + \sum_{i=0}^{k} 1 + \sum_{i=k+1}^{2k-1} 4k - \sum_{i=k+1}^{2k-1} 2i + \sum_{i=k+1}^{2k-1} 1 \quad (7)$$

$$= 2(\frac{k(k+1)}{2}) + (k+1) + 4k(k-1) - 2(\sum_{i=1}^{2k-1} i - \sum_{i=1}^{k} i) + (k-1) \quad (8)$$

$$= 5k^2 - k - 2(\frac{((2k-1)(2k))}{2} - \frac{k(k+1)}{2}) \quad (9)$$

$$= 5k^2 - k - (3k^2 - 3k) \quad (10)$$

$$= 2k^2 + 2k \quad (11)$$

Note that, this one less than the number of nodes in G_k since each node receives a message except the root node.

4.2 Local Computation

Each node needs 26 local operations for assignments, comparisons, and sending the packets. 9 of these operations

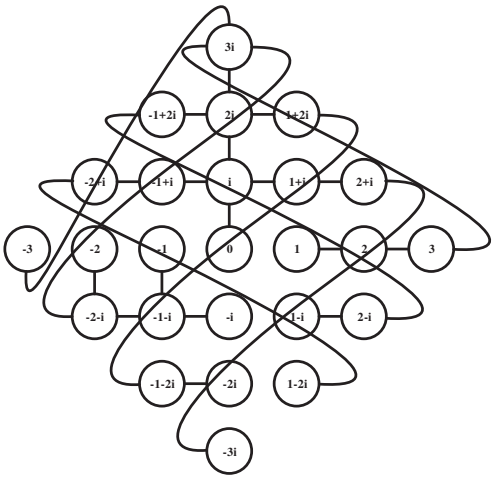


Fig. 4: Second spanning tree

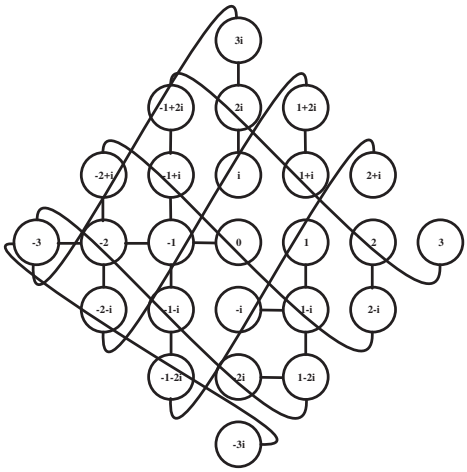


Fig. 5: Third spanning tree

are performed in Algorithm 2 and 17 of them are performed when calling one of the following algorithms 3, 4, 5, and 6. Hence, the number of local computations in step i is $26Comm(i)$. In details, it is:

$$\begin{cases} 26(2i+1), \ 0 \leq i \leq k \\ 26(2(2k-i)+1), \ k+1 \leq i \leq 2k-1 \end{cases} \quad (12)$$

Thus, the total amount of work over all phases is $52k^2+52k$.

4.3 Total Complexity

All four trees can be constructed in $2k$ parallel steps. The total communication overhead to construct all trees is:

$$= 4 \times Communication\ Complexity\ for\ One\ Tree \quad (13)$$
$$= 8k^2 + 8k \quad (14)$$

Algorithm 5 Tree3(*parent*, *current*): Invoked by a function call in Algorithm 2

1: **if** *current*.y = 0 **and** *current*.x $\geq -k$ **and** *current*.x ≤ -1 **then**
2: *current*.Child1 $\leftarrow$ *current*.y$+i$
3: send through $+i$ packet (*current*, *current*.Child1, 3)
4: *current*.Child2 $\leftarrow$ *current*.x-1
5: send through -1 packet (*current*, *current*.Child2, 3)
6: *current*.Child3 $\leftarrow$ *current*.y$-i$
7: send through $-i$ packet (*current*, *current*.Child3, 3)
8: **end if**
9: **if** *parent*.Child1 = *current* **then**
10: *current*.Child1 $\leftarrow$ *current*.y$+i$
11: **if** *current*.y $\geq (-k+1)i$ **and** *current*.y $\leq -i$ **and** *current*.x = 1 **then**
12: *current*.Child2 $\leftarrow$ *current*.x-1
13: **end if**
14: **if** *current*.Child1.y $\neq 0$ **then**
15: send through $+i$ packet (*current*, *current*.Child1, 3)
16: **end if**
17: **end if**
18: **if** *parent*.Child3 = *current* **then**
19: *current*.Child3 $\leftarrow$ *current*.y$-i$
20: **if** *current*.Child3.y $\neq i$ **then**
21: send through $-i$ packet (*current*, *current*.Child3, 3)
22: **end if**
23: **end if**

The total amount of computation work needed to construct all trees is:

$$= 4 \times Local\ Computations\ for\ One\ Tree \quad (15)$$
$$= 208k^2 + 208k \quad (16)$$

5. Conclusions

In this paper, we briefly described the definition of graphs, independent paths, node independent spanning trees, and Gaussian network. Then, we gave algorithms that construct four node independent spanning trees in Gaussian networks. The depth of each tree is $2k$. Further, we have computed the construction complexity for the given algorithms.

In our future work, we would like to extend this work and implement algorithms for parallel construction of node independent spanning trees and fault-tolerant routing from a given source node to a given destination node.

References

[1] B. AlBdaiwi, Z. Hussain, A. Cerny, and R. Aldred, "Edge-disjoint node-independent spanning trees in dense gaussian networks," *The Journal of Supercomputing*, pp. 1–19, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11227-016-1768-x

[2] H. R. Arabnia and J. W. Smith, "A reconfigurable interconnection network for imaging operations and its implementation using a multi-stage switching box," in *Proceedings of the 7th annual international high performance computing conference. The*, 1993, pp. 349–357.

Algorithm 6 Tree4(*parent*, *current*): Invoked by a function call in Algorithm 2

1: **if** *current*.x = 0 **and** *current*.y >= $-ki$ **and** *current*.y $<$ 0 **then**
2: *current*.Child1 $\leftarrow$ *current*.x-1
3: send through -1 packet (*current*, *current*.Child1, 4)
4: *current*.Child2 $\leftarrow$ *current*.y$-i$
5: send through $-i$ packet (*current*, *current*.Child2, 4)
6: *current*.Child3 $\leftarrow$ *current*.x$+1$
7: send through $+1$ packet (*current*, *current*.Child3, 4)
8: **end if**
9: **if** *parent*.Child1 = *current* **then**
10: *current*.Child1 $\leftarrow$ *current*.x-1
11: **if** *current*.x ≥ 1 **and** *current*.x $\leq k-1$ **and** *current*.y = i **then**
12: *current*.Child2 $\leftarrow$ *current*.y$-i$
13: **end if**
14: **if** *current*.Child1.x $\neq 0$ **then**
15: send through -1 packet (*current*, *current*.Child1, 4)
16: **end if**
17: **end if**
18: **if** *parent*.Child3 = *current* **then**
19: *current*.Child3 $\leftarrow$ *current*.x$+1$
20: **if** *current*.Child3.x $\neq -1$ **then**
21: send through $+1$ packet (*current*, *current*.Child3, 4)
22: **end if**
23: **end if**

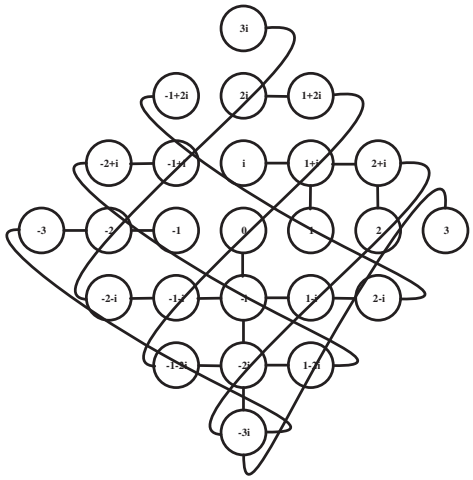


Fig. 6: Fourth spanning tree

[3] H. Arabnia and S. Bhandarkar, "Parallel stereocorrelation on a reconfigurable multi-ring network," *The Journal of Supercomputing*, vol. 10, no. 3, pp. 243–269, 1996.

[4] S. Bhandarkar and H. Arabnia, "The hough transform on a reconfigurable multi-ring network," *Journal of Parallel and Distributed Computing*, vol. 24, no. 1, pp. 107 – 114, 1995.

[5] S. M. Bhandarkar and H. R. Arabnia, "The REFINE multiprocessor – theoretical properties and algorithms," *Parallel Computing*, vol. 21, no. 11, pp. 1783 – 1805, 1995.

[6] S. M. Bhandarkar, H. R. Arabnia, and J. W. Smith, "A reconfigurable architecture for image processing and computer vision," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 09, no. 02, pp. 201–229, 1995.

[7] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, 1st ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1997.

[8] M. Flahive and B. Bose, "The topology of Gaussian and Eisenstein-Jacobi interconnection networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 8, pp. 1132–1142, August 2010.

[9] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[10] K. Huber, "Codes over Gaussian integers," *IEEE Transactions on Information Theory*, vol. 40, no. 1, pp. 207–216, Jan. 1994.

[11] A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," *Inf. Comput.*, vol. 79, no. 1, pp. 43–59, Oct. 1988. [Online]. Available: http://dx.doi.org/10.1016/0890-5401(88)90016-8

[12] C. J. P. J. H. Jordan, "Complete residue systems in the gaussian integers," *Mathematics Magazine*, vol. 38, no. 1, pp. 1–12, 1965. [Online]. Available: http://www.jstor.org/stable/2688007

[13] M. S. Krishnamoorthy and b. Krishnamurthy, "Fault diameter of interconnection networks," *Comput. Math. Appl.*, vol. 13, no. 5-6, pp. 577–582, Apr. 1987. [Online]. Available: http://dl.acm.org/citation.cfm?id=35064.36256

[14] J.-C. Lin, J.-S. Yang, C.-C. Hsu, and J.-M. Chang, "Independent spanning trees vs. edge-disjoint spanning trees in locally twisted cubes," *Information Processing Letters*, vol. 110, no. 10, pp. 414 – 419, 2010.

[15] C. Martinez, R. Beivide, E. Stafford, M. Moreto, and E. Gabidulin, "Modeling toroidal networks with the Gaussian integers," *IEEE Transactions on Computers*, vol. 57, no. 8, pp. 1046–1056, Aug. 2008.

[16] C. Martinez, E. Vallejo, R. Beivide, C. Izu, and M. Moreto, "Dense Gaussian networks: Suitable topologies for on-chip multiprocessors," *International Journal of Parallel Programming*, vol. 34, pp. 193–211, 2006.

[17] A. Rescigno, "Vertex-disjoint spanning trees of the star network with applications to fault-tolerance and security," *Information Sciences*, vol. 137, no. 1-4, pp. 259–276, 2001.

[18] J.-S. Yang, H.-C. Chan, and J.-M. Chang, "Broadcasting secure messages via optimal independent spanning trees in folded hypercubes," *Discrete Applied Mathematics*, vol. 159, no. 12, pp. 1254 – 1263, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X11001454

[19] J.-S. Yang, J.-M. Chang, and H.-C. Chan, "Independent spanning trees on folded hypercubes," in *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, ser. ISPAN '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 601–605.

A Load Service Structure with An Reputation System in Ad-hoc Networks

Ming-Chang Huang
Department of Business Information Systems / Operation Management
University of North Carolina at Charlotte
mhuang5@uncc.edu

Abstract - It is important how wireless hosts find other hosts securely and efficiently for load service purposes because hosts in an ad-hoc network moves dynamically. In this paper, I design a method for load services in computer networks with a new reputation system to check available host reputation. I use databases for directory agents to save information provided by load-server agents and build protocols that how a host can find available hosts for load service and load transfer purposes when it moves to a new region. This includes how a directory agent builds its database, how a load-server agent provides its services, and how a load-client agent gets the services it needs. I will also use fuzzy logic control method to transfer loads for load balancing, instead of fixed threshold level methods. The purpose of this new system structures is to provide efficient ways in building communication and accessing resources in ad-hoc computer network systems. This helps users to find data easily and securely.

Keywords: Load Service, Ad-Hoc Network, Directory Agent, Load-server Agent, Load-client Agent, Peer-to-Peer, Reputation System

1. Introduction

Computer networks can provide parallel computation and services. It is important that hosts send their loads to other hosts for certain function implementation through network transfer. With the increasing popularity of mobile communications and mobile computing, the demand for load services and load balancing grows. When a computer is overloaded or it needs special services from other computers, it may send requests to other computers for load transfer or load services. For example, a computer may need some jobs to be executed with higher quality of services or it needs some jobs to be done with a short period of time that its processor is too slow to perform the jobs; therefore, it may send part those jobs to other computers with higher speeds of processors. Since wireless networks have been wild used in recent years, how a host transfers its loads to other nodes has becomes a very important issue because not all wireless hosts have the ability to manipulate all their loads. For instance, a host with low battery power cannot finish all its jobs on time and should transfer some of them to other hosts. Currently, most of load balancing algorithms are based on wired network environments, it is important to find an efficient way for load service purposes.

Before a wireless host transfers its loads to other hosts or asks for load services from other hosts, it has to find available hosts using resource allocation algorithms. There are several resource allocation protocols been developed, for example, IEFT Service Location Protocol (SLP) [1] and Jini [2] software package from Microsystems. However, these protocols address how to find the resources in wired networks, not in wireless networks. Maab [3] develops a location information server for location-aware applications based on the X.500 directory service and the lightweight directory access protocol LDAP [4]; while it does not cover some important issues about the movements of mobile hosts, for example, how to generate a new directory service and how a host gets the new services, when a directory agent moves away its original region. In an Ad-Hoc network, system structure is dynamic and hosts can join or leave any time. Therefore, how to provide load services and how to find available hosts providing load services become importance issues in an Ad-Hoc network system.

To find a host which can fulfill the load service purpose, the requesting host also has to make sure that the host it is looking for has good reputation in load services. For good reputation hosts, they will have to share their resources as well besides just requesting resources from other hosts. It is called the "free-riding" situation if a host only requests resources from other hosts without sharing it resources to others. Measurement study of free-riding on Gnutella was first reported by [10] in 2000 which indicated that approximately 70% of Gnutella users did

not share any files and nearly 50% queries were responded from top 1% peers. However, according to the most recently measurement study, the percentage of free riders rises to 85% [11]. It is very possible that a small number of peers who are willing to share information take most of the job loadings in P2P networks. As a result, the prevalence of free riders will eventually downgrade the performance of entire system and would make the system vulnerable [12].

In this paper, I am going build a system structure for load services with reputation checking in wireless Ad-Hoc network systems using peer-to-peer concept [8, 9]. In Ad-Hoc network systems, hosts move dynamically without base stations for communication. The load service architecture provides special services upon requests from hosts and these services, e.g., include resource location services and load balancing services. A host may send its special requests to other hosts for load services or send its loads for load balancing. The requests include service types the host needs or the amount of loads to be sent to other hosts. For those special services, the host should define the conditions that other hosts may accept the services. For example, the request includes the price of job execution, the limit requirement of execution time, etc. Besides looking for the desired resources, the requesting host also check the requested host's reputation to avoid "free-riding" cases [7].

In Section 2, I discuss the system structure. Section 3 expresses the details of the method. Section 4 and section 5 illustrate the information format for databases, and the scalability respectively. Section 6 presents the conclusion.

2. System structures

In this section, I am going to describe the structure used in the system. Basically, there are three components in my load service system – directory agent, load-server agent and load-client agent. A load-server agent provides load services that are queried by other hosts (load-client agents) which require load services. Load-server agents post the types of services periodically to their directory agents to update the services they can provide to load-client agents. A load-client agent is a host in the network, which may need some services performed by other hosts. It sends requests to its directory agents to ask for services from load-server agents when it is heavily loaded or it needs some special services, which it does not have the ability to perform. A directory agent forms groups for both load-server agents and load-client agents respectively and builds a database for service queries from load-client agents.

Figure 1 shows an example based on the architecture of my load service system. Figure 2 shows the structure of the reputation system (FuzRep) [7] which I am going to apply in the paper. Each directory agent has a query database, which stores all the query information from load-server agents. Load-server agents and load-client agents may join directory agents upon requests. In Figure 1, for example, Load-server Agent 1 and Load-client Agent 1 register with Directory Agent 1; Load-server Agent 2 registers with Directory Agent 1 and Directory Agent 2 at the same time. Load-client Agent 1 may send requests to Directory Agent 1 for querying load services and Directory Agent 1 checks its database and the reputation system to find fitted load-server agents and sends those available load-server agent addresses to Load-client Agent 1. The fitted load-server agents can be Load-server Agent 1, Load-server Agent 2, or both. Load-client Agent 1 can choose one of them based on its best convenience; or it can choose both of them for special purposes. Of course, it is possible that none of the load-server agents can be found.

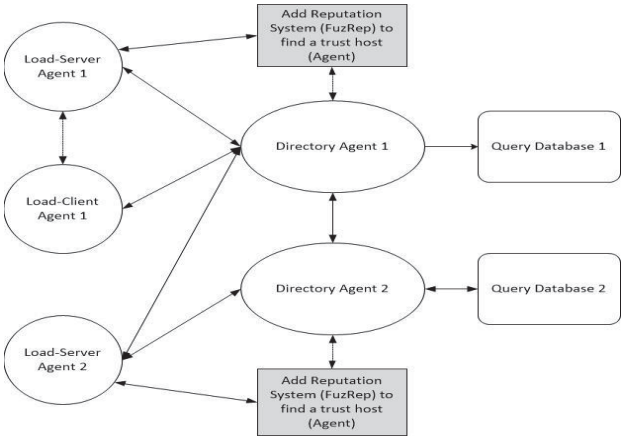


Figure 1: Load service system architecture with FuzRep Reputation System

FuzRep is a design of a fuzzy-based reputation system for P2P networks. It includes three techniques – reputation determination, selective polling, and service differentiation. I am going to describe how FuzRep works by revealing answers of the following questions.

- How to determine a peer's reputation level? What are the criteria? How to transfer a crisp score to a reputation level? How to maintain it?
- How and when to share the contribution information?
- How to encourage sharing and discourage free riding? How to differentiate the service level?

In FuzRep, a peer's reputation is determined by its contributions to the communities. A peer saves transaction information into local transaction repository, including requesters' or providers' IDs, and accumulated contribution scores. The transaction repository is updated after every successful transaction. The initial local contribution score is set to zero originally for pre-unknown peers at their first interactions. A global

accumulated contribution score is used to determine corresponding peer's reputation. It is built on two phase computes – personal reputation inference and global reputation deduction. Personal reputation inference simply fetches a peer's contribution score from its transaction repository. If a file provider chooses to determine a file request's reputation simply based on its experience, personal reputation inference fits that purpose. Otherwise, the file provider should run the global reputation deduction process using the selective polling reputation sharing process. [7]

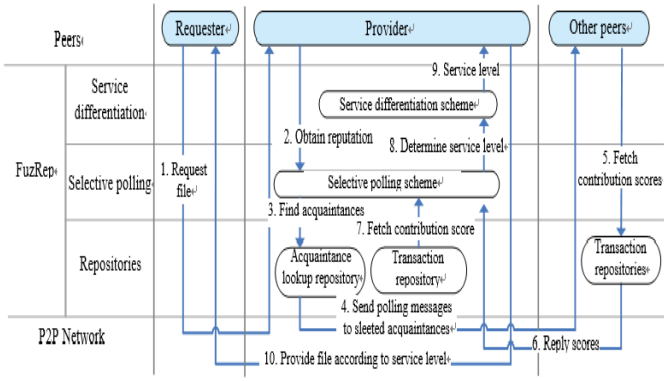


Figure 2. FuzRep architecture and operational processes

3. Algorithm for wireless ad-hoc load services

There are several issues that I consider when designing the system architecture, which includes, for example, how a directory agent asks a host to register with its database, the effects of the movement of mobile hosts to the join of load-server and load-client agents, and fault tolerance of the system. Below I explain how hosts join or leave directory agents and how directory agents form their databases when they move.

I also describe how a load-client agent should pay load-server agents that it asks the services from and how hosts in the system gain tokens in order to pay the services it need. How to transfer loads between load-server agents and load-client agents is also mentioned in this section.

3.1 A directory agent asks hosts for registration

In order to collect load service information from other hosts and provide results for queries, a directory agent builds a query database. The information in the database includes the addresses of load-server agents which provide information, the service types, or the loads that load-server agents can accept. The host can be a desk computer or a laptop once it has the ability; for example, it has high-speed processors, enough power for communication, etc. The method how a directory agent asks for registration is discussed below.

1. A directory agent broadcasts a message to the other hosts within the range that its power can reach.
2. A host, which receives the broadcast message from a directory agent and is willing to register with the directory agent's database as a load-server agent, sends an ACK message to the directory agent for registration. The ACK message includes information, such as the service types it can perform and/or the loads it can accept, etc., provided by a load-server agent.
3. The directory agent keeps the ACK information in its query database and therefore builds a link from itself to the load-server agent sending the ACK message.
4. To check if a load-server agent is still available in the database, a directory agent periodically sends multicast messages to all the load-server agents, which have query information in its database. This purpose for this is for database information update because load-server agents might move away anytime. When a load-server agent receives a query message from a directory agent, it should send back a response to the directory agent to indicate that it is still existed in the directory agent's power range. If the directory agent does not get the acknowledgement from a load-server agent that has query information in the database, it deletes the information provided by that load-server agent from its database and therefore deletes the link between them. The Figure 3 demonstrates the steps how a directory agent builds its query database.
   (1) A directory agent sends requests to hosts for registration.
   (2) Hosts, which are willing to register as load-server agents, send ACKs back to the directory agent.
   (3) The directory agent saves all the information in those ACKs to its database for future use.
   (4) The directory agent also builds links between itself and its load-server agents.

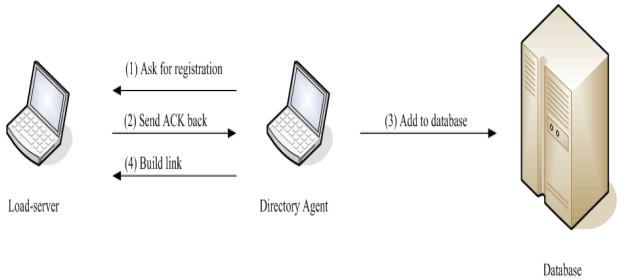


Figure 3: The procedures for a directory agent asks for registration

3.2 A host join directory agent's databases as a load-server agents

A mobile host may join directory agents' databases as a *load-server* agent when it has the ability to provide services, or it is lightly loaded and is willing to accept loads from other hosts. Not only a load-server agent may join a directory agent, but also it may join multiple directory agents. A load-server agent joins directory agent's databases in two ways.

Method 1: The first method is that it sends out messages to ask for registering with directory agents within its power range and waits for the replies from those directory agents. After receiving acknowledgements from directory agents, the mobile host registers with the databases of those directory agents by sending its address, the service types it can provide, and the amount of loads it can accept for load transfer. A mobile host can register with several directory agents at the same time; which means a mobile host can join several databases simultaneously.

Method 2: The second method, like the method in Section 3.1, is that a mobile host receives messages from some directory agents for requesting joining their databases. Thereafter, the mobile host may join those databases by replying acknowledgements (ACKs) back to those directory agents and the directory agents add the ACKs into their databases.

After the directory agents receive the ACKs from load-server agents, they build links between them. The following figure illustrates the procedures of Method 1 for a load-server agent to a directory agent database.

(1) A host sends request to directory agents for registering as a load-server agent.
(2) Directory agents send ACKs back to the host when they receive the request and allow it to join their databases.
(3) The host sends registration information to those directory agents once it receives the ACKs.
(4) Those directory agents add the information into their databases.
(5) The directory agents also build links between themselves and the load-server agent.

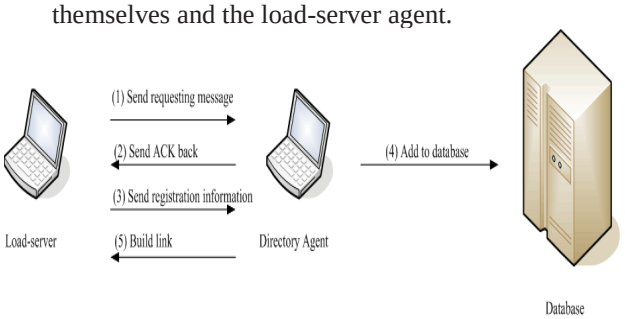


Figure 4: How a load-server joins a directory agent database for Method 1

3.3 Queries from load-client agents

A mobile host may join directory agents' databases as a *load-client* agent when it needs services from other hosts. Since directory agents broadcast their addresses periodically to ask for mobile hosts to register for services, a load-client agent can find the addresses of directory agents from those broadcasting messages. When a load-client agent needs load services, it sends queries to directory agents that it can contact and waits for the replies from them. The contents in these replies include the addresses of available load-server agents that can provide the services the load-client agent asks. The load-client agent may receive several replies from different load-server agents at the same time and it chooses the best-fit one. If it cannot find available load-server agents (without any reply from directory agents in a period of time), it waits for a certain period of time and sends queries again.

A load-client agent selects the best-fit load-server agent based on the service conditions it requests. For example, it may choose the one that satisfies the price the load-client agent asks. When a load-client agent selects the best-fit load-server agent, it directly sends service requirements or loads to the chosen load-server agent. Figure 5 shows the steps.

(1) A load-client agent sends query to directory agents to request services
(2) Directory agents apply the FuzRep reputation system to the requesting host for a free-rider check. Then the Directory agents search their database for the desired services requested by the load-client agent. Before the Directory agents send back searching information, they also apply the FuzRep reputation system to the load-server agents to avoid free-riding.
(3) Directory agents send replies back, which indicate the information they have in the databases.
(4) The load-client agent gets the services it needs from load-server agents.

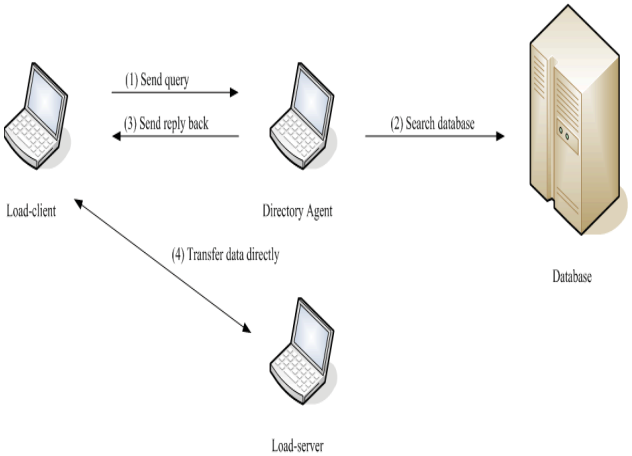


Figure 5: How a load-client agent sends queries

3.4 Movement of directory agents

When a directory agent moves to another region, it loses all the information in its database about load-server agents and its peer directory agents. How a directory agent notifies all the other agents about its movement becomes an important issue. There are two ways that other agents can detect the leave of a directory agent. The first is that the directory agent sends a message to notify other hosts about its movement. Hosts receiving the message will stop sending queries to this directory agent and remove the links between them.

The second method is to use the fact that hosts cannot detect the existence of a directory agent. Since load-server agents send update information to a directory agent periodically, load-server agents can notice that a directory agent does not exist in the region if hosts do not get the reply from that directory agent. For a load-client agent to detect the existence of a directory agent, if it does not receive any broadcast message during a period of time, then it deletes the link to that directory agent.

After moving to a new region, a directory agent sends messages to hosts in the power range it can reach to ask for hosts to join its database for load services as discussed in section 3.1. It may happen that some hosts do not have any directory agent to contact to once a directory agent moves away. Those hosts will keep sending messages to other hosts for finding new directory agents as described in section 3.2 and 3.3.

3.5 Movement of load-server agent

When a load-server agent moves to a new region, it may lose its original directory agents and it has to establish new links to its new directory agents as described in section 3.2. Once a directory agent does not receive update information from a load-server agent for a period of time, it deletes the information about that load-server agent from its database and therefore deletes the link between them.

3.6 An example

Figure 6 illustrates a flow how a directory agent, load-server agent, and load-client agent communicates each other. (1), (2), (3), (4), and (5) indicate the procedures for setting up the processes.

(1) A Directory Agent broadcasts join message to hosts.
(2) Load-Server Agent replies an acknowledgement to that Directory Agent to join the database.
(3) Directory Agent saves the information to its database.
(4) Load-Server Agent sends requests to Directory Agent for load services.
(5) Directory sends the address of Load-Server Agent if Load-Server Agent is suitable for load service.
(6) Load-Client Agent communicates with Load-Server Agent directly.

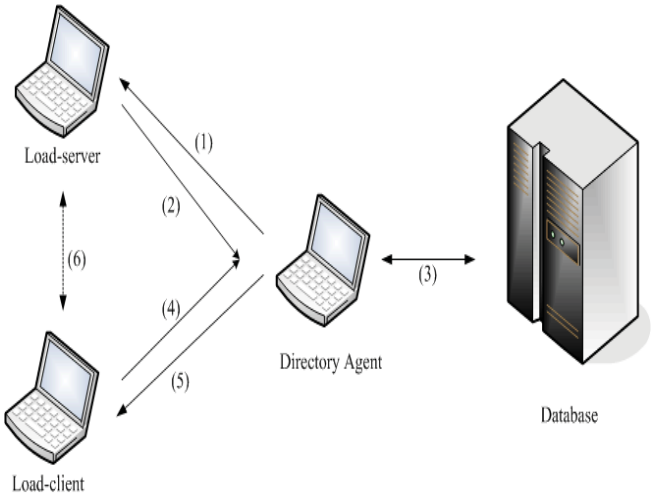


Figure 6: An Example for Communications between Agents

3.7 Load transfer

A host may transfer loads to other hosts when it is heavily loaded. Instead of using fixed threshold method to decide whether a host is heavily loaded, I use fuzzy logic control method to improve the performance. First, the host finds an available host by sending service request as I mentioned before. Once it finds a host that accepts its request for load transfer, it transferred its loads to the selected host. The amount of loads to be transferred is equal to half of the difference of loads between the load-client agent and the load-server agent. It is possible that there are several server load-server agents, which satisfy the request by a load-client agent. In order to reduce the distance and moving effect, a load-client chooses the load-server agent that is the closest one to it. Figure 7, for example, shows the power range that load-client agent C can reach and there are three load-server agents – S1, S2, and S3 – which satisfy the request from agent C. Since S1 is the closest one to C, it is chosen which C will transfer its loads to.

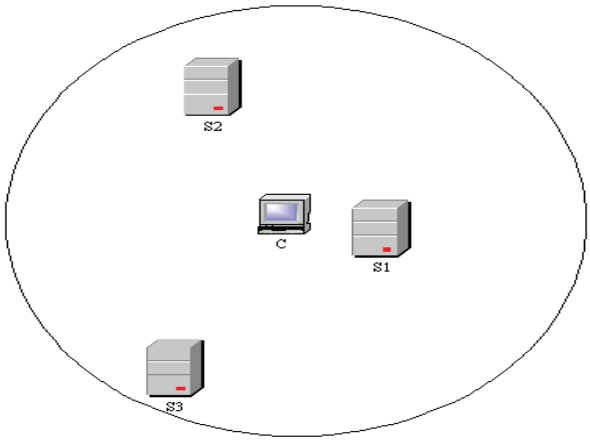


Figure 7: An example for a load-client agent to choose a best-fit load-server agent

The following steps show the details of load transfer.

(1) When a host detects that it is heavily loaded, it broadcasts a request message to hosts in its power range to ask for load transfer service. Instead of using fixed threshold levels to check if it is lightly loaded or heavily loaded, I use fuzzy logic control [5] to check its queue status to improve the performance. This method is mentioned in [6, 7].
(2) Hosts, which receive the request, check their queue status using fuzzy logic control method, and returns ACKs, if they are lightly loaded, to the load-client agent that sent the request.
(3) When the load-client agent gets the ACKs from load-server agents, it chooses the load-server agent, which is the first one to send its ACK, for load transfer. That means that the load-client agent chooses the closest one in order to improve the performance.
(4) If there are no available hosts in the load-client agent power range, the load-client agent sends requests to its directory agents to look for the registered load-server agents for load transfer. Then it waits for the responses from its directory agents.
(5) The directory agents find available (lightly loaded) load-server agents when they receive requests from a load-client agent. Then, the directory agents send addresses of these available load-server agents to that load-client agent for load transfer. The load-client chooses the best host to transfer its loads to the selected host.

4. Service type format and service price

In the future, it is possible that hosts have to pay if they ask for service from other hosts. In this section, I discuss this situation and define the service type format for load services and the price for each service. This format is for a directory agent to store the information in its database. Figure 8 shows the format that there are 4 fields in it – *address*, *service-type*, *number-of-tokens*, and *load*.

The *address* field is the address of a load-server agent, so that a load-client can directly connect to it. The *service-type* field indicates which kind of services that a load-server agent provides. The *number-of-tokens* shows the price of a service for a load-client to pay, and the *load* field shows the current load for a load-server agent. When a load-server agent provides load services to directory agents, it provides directory agents the information about the type(s) of services it can provides, the tokens (price) for a load-client agent to take the service, and the current load status and address for the load-server agent. A load-client agent can get the service only it matched the service type, and the price that the load-server agents ask, or it can find an available load-server agent for load transfer purpose if the load-server agent is lightly loaded and the load-client agent can pay the price.

address	*service-type*	*number-of-tokens*	*load*

Figure 8: Service Type Format Stored

There are some assumptions in our architecture for hosts.

(1) A load-client agent has to pay a load-server agent when it needs load services from that load-server agent.
(2) When sending a request to a directory agent, a host loses tokens as the price for asking load service.
(3) In order to increase the number of tokens and therefore increase the ability to ask for services, a host must try its best to gain tokens. There are two possible ways to implement it. First of all, a host can provide the services to other hosts to gain tokens. Secondly, a host should avoid sending useless requests to network to save tokens. This can be implemented by increasing the waiting time for a load-client agent to send requests. This also may avoid network congestion because the number of messages is reduced.
(4) A load-client agent may find several available load-server agents for a particular request such that those load-server agents satisfy the requirements for the load-client agent. Then the client host has to choose the best-fit one.
(5) If a host does not have enough tokens to find a load-server agent for load services, it should stop sending requests to its directory agents for asking load services until it can provide enough tokens.

The request message, which a load-client agent sends out when it needs a service, includes a price that the load-client agent can pay. The directory agent, which receives the message, finds available load-server agents by comparing the key words and the prices. For example, if a host needs a service with higher speed calculation, it sends requests to its directory agents. In these requests, the speed of the load-server agent's processor and the price the load-client agent can provide are included. Directory agents match these requirements to the information via the key words and the number of tokens in their database and therefore find the available load-server agents. The addresses of those load-server agents are sent to the requesting load-client agent. Upon receiving those addresses, the load-client agent chooses one available and sends jobs directly to that load-server agent. To choose an available load-server agent from those addresses by directory agents, the load-client agent may choose the one, which asks the lowest number of tokens for performing the requesting service.

5. Scalability

As the number of clients and servers in the network system increase, so does the burden to the system because of the increases of messages for service discovery and request. When a host joins or roams into a network, it sends out requests. If there are too many hosts that move too frequently, they may send many requests, which may cause the congestion of the network. Therefore, careful consideration of scalability issues is very important to the design of the protocols. In our system, I use the number of tokens (the price to pay) to control the scalability of load-server agents registered with directory agents and load-client agents sending load service requests. For example, a client host cannot send requests to directory agents for services if it does not have enough tokens. It should provide its services to other hosts to gain enough tokens before it sends requests.

6. Conclusion

In the paper, a new load service method in wireless ad-hoc networks is proposed using a reputation system to check nodes' reputation. Since the hosts in a wireless ad-hoc network can move anywhere by anytime, it is difficult for a host to find other host for load service or load transfer purposes. Several issues are discussed about a directory agent asking for hosts to register as load-server agents, a load-server agent registering with directory agents' databases, and a load-client agent finding available load-server agents when it need load services. The Directory agents can find the available load-servers which provide services the clients need. Also the Directory agents apply the FuzRep reputation system to check the clients and servers' reputation to load and services requests. This is to avoid free-riding situation in P2P network systems.

This paper also addresses a new concept that a host should pay the price when it needs services from other hosts in networks and how it works by using token as the price in the networks. The token concept is also used to control the scalability of networks and congestion control of network flow. Fuzzy logic control is used to check load status of hosts in the load transfer protocol.

References

[1] E. Guttman, C. Perkins, J. Veizades and M. Day, "Service Location Protocol," Version 2, IEFT, RFC 2165, November 1998.

[2] J. Waldo, "The Jini Architecture for network-centric computing," Communication of the ACM, pp 76-82, July 1999.

[3] H. Maab, "Location-Aware Mobile Application Based on Directory Services," MOBICOM 97, pp 23-33.

[4] W. Yeong, T. Howes, and S. Kille, "Lightweight Directory Access Protocol," RFC 1777, March 1995.

[5] Ross, T. J., Fuzzy Logic with Engineering Applications, McGraw Hill, 1995.

[6] Huang, M., S. H. Hosseini, and K. Vairavan, "Load Balancing in Computer Networks," Proceedings of ISCA 15th International Conference on Parallel and Distributed Computing Systems (PDCS-2002), Special session in Network Communication and Protocols. Held in the GALT HOUSE Hotel, Louisville, Kentucky, Sep. 19 - 21.

[7] Ross, T. J., Fuzzy Logic with Engineering Applications, McGraw Hill, 1995.

[8] Andy Oram et al., "Peer-to-Peer:Harnessing the Power of Disruptive Technologies," Oreilly 2001.

[9] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, 36(4):335–371, December 2004. doi:10.1145/1041680.1041681.

[10] Adar, E., and Huberman, B. A. Free riding on Gnutella. *First Monday* 5, 10 (October 2000).

[11] Hughes, D., Coulson, G., and Walkerdine, J. Free riding on Gnutella revisited: the bell tolls? In *IEEE Distributed Systems Online* 6, 6 (June 2005).

[12] Ramaswamy, L. and Liu, L. Free riding: A new challenge to peer-to-peer file sharing systems. In Proceedings of 36th Hawaii International Conference on System Sciences (HICSS'03) (January 2003)

A Path Routing Algorithm for the Basic WK-Recursive Pyramid Networks

Yi-Chun Wang and Justie Su-Tzu Juan*
Department of Computer Science and Information Engineering,
National Chi Nan University, Puli, Nantou, Taiwan.
*Corresponding author: jsjuan@ncnu.edu.tw

Abstract— *In routing, a source vertex sends a message to a target vertex and the routing algorithm decides a path from the source vertex to the target vertex. In 2013, Wang and Juan proposed a simple version of the WK-recursive pyramid networks, is called the basic WK-recursive pyramid network, which have received much attention recently. There are many literatures that study on this topology. This paper studies the path routing problem on the basic WK-recursive pyramid networks. Because each basic WK-recursive pyramid network consists of the WK-recursive networks, we also review a shortest path routing algorithm for the WK-recursive networks first.*

Keywords: Path, Routing algorithms, Basic WK-recursive pyramids, WK-recursive networks

1. Introduction

The *WK-recursive network* (WK, for short), proposed in 1987 [1], is a network that is recursively defined and is expandable to any level. This is a well-known network in network computing, there are several researches on the n dimension WK-recursive network ([2][3][4]). The definition of the WK-recursive network is reviewed as follows. Figure 1 is an illustration of $\text{WK}_{(3,3)}$.

Definition 1: [5] A *radix-t WK-recursive network with expansion level d, denoted as* $WK_{(d,t)}$, consists of a set of vertices $V(\text{WK}_{(d,t)}) = \{a_{t-1}a_{t-2}...a_1a_0 | 0 \leq a_i < d, 0 \leq i \leq t-1\}$. Each vertex $a_{t-1}a_{t-2}...a_1a_0$ is adjacent to (1) $a_{t-1}a_{t-2}\dots a_1b$, where $0 \leq b < d$ and $b \neq a_0$; and (2) $a_{t-1}a_{t-2}\dots a_{j+1}a_{j-1}(a_j)^j$, if $a_j \neq a_{j-1}$ and $a_{j-1} = a_{j-2} = \cdots = a_0$, where $(a_p)^q$ denotes q consecutive a_ps. The edges of type (1) are referred to as *substituting edges* and the edges of type (2) are referred to as *flipping edges*.

For convenience, $a_{t-1}a_{t-2}...a_1a_0$ is called the *labelling* of a vertex in $V(\text{WK}_{(d,t)})$. A vertex $a_{t-1}a_{t-2}...a_1a_0$ is called a *k-frontier* if $a_{k-1} = a_{k-2} = \cdots = a_0$, where $1 \leq k \leq t$. A k-frontier is *proper* if it is not a $(k+1)$-frontier.

In 1993, Fernandes and Kanevsky [6] proposed hierarchical WK-recursive topology, which are called the *WK-recursive pyramid network* (WKP, for short). Figure 2 shows $\text{WKP}_{(4,2,2)}$.

Definition 2: [6] A *WK-recursive pyramid network of height* l, $\text{WKP}_{(d,t,l)}$, consists of a set of vertices $V(\text{WKP}_{(d,t,l)}) = \{(0)\} \bigcup \{(k, a_{tk-1}a_{tk-2}\dots a_1a_0) |\ 1 \leq k \leq l, 0 \leq a_i < d, 0 \leq i \leq tk-1\}$, where (0) is an apex in level 0 and $\{(k, a_{tk-1}a_{tk-2}\dots a_1a_0) |\ 1 \leq k \leq l, 0 \leq a_i < d, 0 \leq i \leq tk-1\}$ arranged in l levels of the radix-tk WK-recursive networks. The apex (0) is adjacent to $(1,(0)^t)$, $(1,(1)^t)$, $\dots$, and $(1,(d-1)^t)$, other vertex is addressed as $(k, a_{tk-1}a_{tk-2}\dots a_1a_0)$ and is said to be a vertex at level k. The part $a_{tk-1}a_{tk-2}\dots a_1a_0$ of the address determines the address of a vertex within layer k of the radix-tk WK-recursive network,$\text{WK}_{(d,tk)}$. The vertices at level k form a network of $\text{WK}_{(d,tk)}$, i.e., a vertex with the address $(k, a_{tk-1}a_{tk-2}\dots a_1a_0)$ placed at level k of the $\text{WK}_{(d,tk)}$ network, is connected to adjacent vertices as defined in Definition 1. This vertex is also connected to vertices $(k+1, a_{tk-1}a_{tk-2}\dots a_1a_0(b)^t)$ for $0 \leq b \leq d-1$, in level $k+1$, as *childvertices*, and to vertex $(k-1, a_{tk-1}a_{tk-2}\dots a_{t+1}a_t)$ as the *parent*, if $a_{t-1} = a_{t-2} = \cdots = a_0$.

In 2013, Wang and Juan proposed a simple version of $\text{WKP}_{(d,t,l)}$, which is to fix the variable $t = 1$ in $\text{WKP}_{(d,t,l)}$ [7]. This topology denoted by $\text{WKP}_{(d,l)}$. The definition is shown as follows.

Definition 3: [7] A *basic WK-recursive pyramid network of height* l, $\text{WKP}_{(d,l)}$, consists of a set of vertices $V(\text{WKP}_{(d,l)}) = \{(0)\} \bigcup \{(k, a_{k-1}a_{k-2}\dots a_1a_0) |\ 1 \leq k \leq l, 0 \leq a_i < d, 0 \leq i \leq k-1\}$, where (0) is an apex in level 0 and $\{(k, a_{k-1}\ a_{k-2}\dots a_1a_0) |\ 1 \leq k \leq l, 0 \leq a_i < d, 0 \leq i \leq k-1\}$ is arranged in l levels of the radix-d

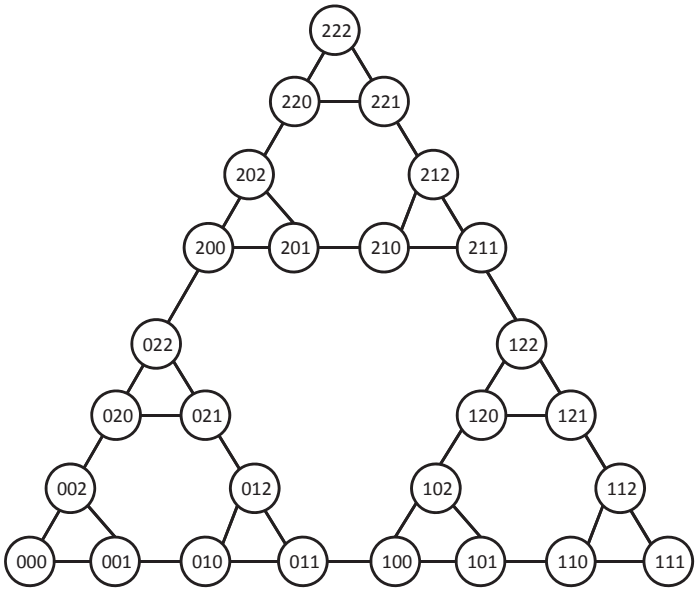


Fig. 1: The structure of $\mathrm{WK}_{(3,3)}$.

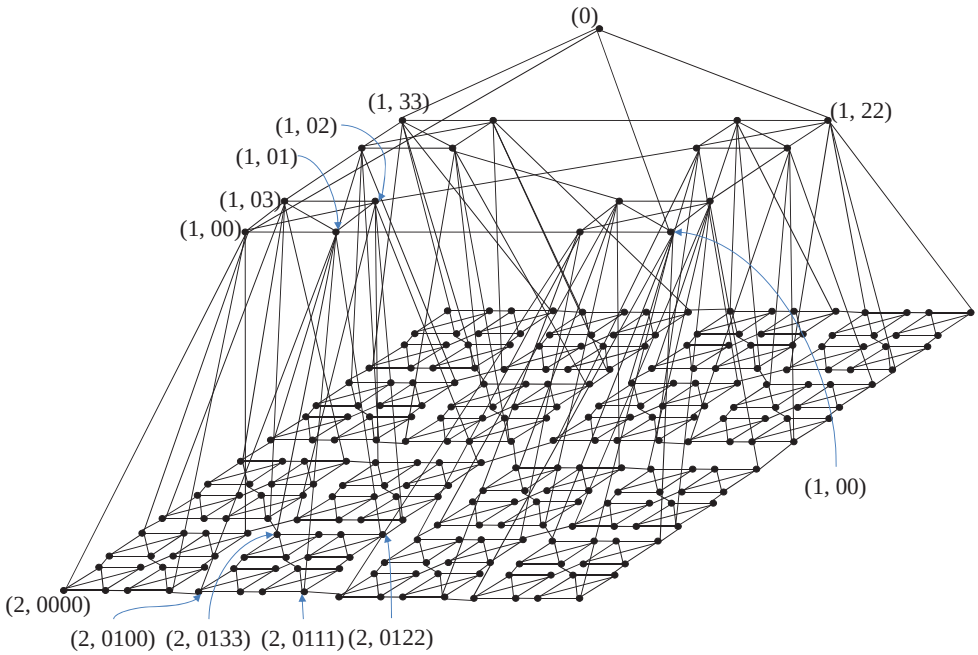


Fig. 2: The structure of $\mathrm{WKP}_{(4,2,2)}$.

WK-recursive network. The apex (0) is adjacent to $(1, 0)$, $(1, 1)$, $\ldots$, and $(1, d-1)$, other vertex v is addressed as $(k, a_{k-1}a_{k-2}\ldots a_1a_0)$ and is said to be a vertex at level k. The part $a_{k-1}a_{k-2}\ldots a_1a_0$ of the address determines the address of a vertex within the layer k of the radix-d WK-recursive network. The vertices at level k form

a network of $\mathrm{WK}_{(d,k)}$, i.e., a vertex v with the address $(k, a_{k-1}a_{k-2}\ldots a_1a_0)$ placed at level k of the $\mathrm{WK}_{(d,k)}$ network, is connected to adjacent vertices as defined in Definition 1. This vertex v is also connected to vertices $(k+1, a_{k-1}a_{k-2}\ldots a_1a_0b)$ for $0 \le b \le d-1$, in level $k+1$, as childvertices, and to vertex $(k-1, a_{k-1}a_{k-2}, \ldots, a_1)$ as the parent, $p(v)$. Conversely, v is called a *child* of $p(v)$; on the other hand, the vertex $p(v)$ has d children, and v is one of them.

Figure 3 shows $\mathrm{WKP}_{(3,2)}$. $\mathrm{WKP}_{(d,l)}$ receive much attention recently ([7][8][9]). For convenience, let $p^i(a) = p(p^{i-1}(a))$ be the parent of $p^{i-1}(a)$, level k of $\mathrm{WKP}_{(d,l)}$ be denoted by $\mathrm{WK}^*_{(d,k)}$ in this paper. That is, for $1 \le k \le l$, $V(\mathrm{WK}^*_{(d,k)}) = \{(k, a_{k-1}a_{k-2}\ldots a_1a_0) | 0 \le a_i < d$ for $0 \le i \le k-1\}$, and $E(\mathrm{WK}^*_{(d,k)}) = \{(k, a_{k-1}a_{k-2}\ldots a_1a_0)$ $(k, a_{k-1}a_{k-2}\ldots\ a_1\alpha) | 0 \le \alpha < d$ and $\alpha \ne a_0\}$ $\bigcup$ $\{(k, a_{k-1}a_{k-2}\ldots a_i\beta_1(\beta_2)^{i-1})$ $(k, a_{k-1}a_{k-2}\ldots a_i\ \beta_2$ $(\beta_1)^{i-1}) | \beta_1 \ne \beta_2$ and $2 \le i \le k\}$. For $1 \le l' \le l$, we define a subgraph of $\mathrm{WKP}_{(d,l)}$ be denoted by $\mathrm{WKP}^*_{(d,l')}$ as $V(\mathrm{WKP}^*_{(d,l')}) = \{(0)\} \bigcup (\bigcup_{1\le i\le l'} V(\mathrm{WK}^*_{(d,i)}))$, and $\mathrm{WKP}^*_{(d,l')}$ is an induced subgraph of $\mathrm{WKP}_{(d,l)}$ by $V(\mathrm{WKP}^*_{(d,l')})$.

In this paper, we review the related work about $\mathrm{WK}_{(d,t)}$ and $\mathrm{WKP}_{(3,2)}$, including some properties of them and the shortest path routing algorithm for $\mathrm{WK}_{(d,t)}$ in Section 2. In Section 3, a path routing algorithm is proposed.

2. Related Work

The following properties are not difficult to see by the definition of $\mathrm{WKP}_{(d,l)}$.

Property 1: In $\mathrm{WKP}_{(d,l)}$, for any $1 \le t \le l$, (a) all neighbours in $\mathrm{WK}^*_{(d,t)}$ of any t-frontier vertex in $\mathrm{WK}^*_{(d,t)}$ form a complete graph; (b) if any two vertices x, y in $\mathrm{WK}^*_{(d,t)}$ satisfy $p(x) = p(y)$, then $xy \in E(\mathrm{WKP}_{(d,l)})$.

Property 2: The order $N_{(d,l)}$ of $\mathrm{WKP}_{(d,l)}$ is $(d^{l+1} - 1)/(d-1)$.

Property 3: The diameter of $\mathrm{WKP}_{(d,l)}$ is $2^l - 1$.

In addition, we also define two kinds of subgraphs of $\mathrm{WKP}_{(d,l)}$. Let $c_{t-1}c_{t-2}\ldots c_m$ be a specific $(t-m)$-digit radix d number. Define $c_{t-1}c_{t-2}\ldots c_m\cdot\ \mathrm{WK}^*_{(d,m)}$ as the subgraph of $\mathrm{WK}^*_{(d,t)}$ induced by $\{(t,\ c_{t-1}c_{t-2}$ $\ldots c_m a_{m-1}a_{m-2}\ldots\ a_1a_0) |\ 0 \le a_i \le d$, for $0 \le i \le m-1\}$; that is, $c_{t-1}c_{t-2}\ldots c_m\cdot\ \mathrm{WK}^*_{(d,m)}$is an embedded $\mathrm{WK}_{(d,m)}$ with the identifier $c_{t-1}c_{t-2}\ldots c_m$. Similarly, we define $c_{t-1}c_{t-2}\ldots c_m\cdot\mathrm{WKP}^*_{(d,m)}$ for $0 \le m \le t \le l$ as the subgraph of $\mathrm{WKP}_{(d,l)}$ induced by $\{(t-m+k, c_{t-1}c_{t-2}\ \ldots c_m a_{k-1}a_{k-2}\ldots a_1a_0) |\ 0 \le k \le m$ and $a_{k-1}a_{k-2}\ldots a_1a_0$ is a k-digit radix d number$\}$; that is, $c_{t-1}c_{t-2}\ldots c_m\cdot\mathrm{WKP}^*_{(d,m)}$ is an embedded $\mathrm{WKP}_{(d,m)}$ with the identifier $c_{t-1}c_{t-2}\ \ldots c_m$. Note that $\mathrm{WKP}^*_{(d,l-1)}$ is the subgraph of $\mathrm{WKP}_{(d,l)}$ induced by all the vertices from level 0 to level $l-1$.

In Figure 3, $1\cdot\mathrm{WK}^*_{(3,1)}$ is the subgraph of $\mathrm{WK}^*_{(3,2)}$ induced by $\{(2,10),(2,11),(2,12)\}$; and $0\cdot\mathrm{WKP}^*_{(3,1)}$ is the subgraph of $\mathrm{WKP}_{(3,2)}$ induced by $\{(1,0),\ (2,00),(2,01),(2,02)\}$. Note that each $c\cdot\mathrm{WKP}^*_{(d,l-1)}$ is a subgraph with level $l-1$.

There are some preliminaries about $\mathrm{WK}_{(d,t)}$ as follows.

Lemma 1: [10] The distance between any two t-fronties with in $\mathrm{WK}_{(d,t)}$ is $2^t - 1$.

Lemma 2: [10] The diameter of $\mathrm{WK}_{(d,t)}$, which denoted by $diam(\mathrm{WK}_{(d,t)})$ is $2^t - 1$.

Because $\mathrm{WK}_{(d,t)}$ is a famous graph, many problems are discussed on it ([2][4][11]). In 1994, Chen and Duh discussed some communication algorithms on $\mathrm{WK}_{(d,t)}$ [2]. They proposed the diameter of $\mathrm{WK}_{(d,t)}$, and designed a shortest path routing algorithm for $\mathrm{WK}_{(d,t)}$. Now, we review the contents of this algorithm.

For convenience, considering any two vertices u and v, defines $u =_i v$ if they belong to the same $\mathrm{WK}_{(d,i)}$, for some $i \in \{1,\ldots,t\}$, and $u \ne_i v$ if they belong to two different $\mathrm{WK}_{(d,i)}$s. Let s' be another endpoint of flipping edge of s in this paper. Chen and Duh first proposed a simple routing algorithm as Algorithm 1. Note that when $t = 0$ and $t = 1$, $\mathrm{WK}_{(d,t)}$ is isomorphic to K_1 and K_d, respectively. Hence, we let $t \ge 2$ in the following discussion.

They also proved the length fomula between source vertex s and destination v if v is i-frontier. Here, σ is defined as a compare function. For any two numbers a and b, $\sigma(a,b) = 0$, if $a = b$; $\sigma(a,b) = 1$, otherwise.

Lemma 3: [2] Let $s = s_{t-1}s_{t-2}...s_1s_0$ and $v = v_{t-1}$ $v_{t-2}...v_1v_0$ denoted the source vertex and destination vertex, respectively, in $\mathrm{WK}_{(d,t)}$. If $s_{t-1} = v_{t-1}$, $s_{t-2} = v_{t-2}$, ..., $s_i = v_i$ (that is, $s =_i v$) and v is an i-frontier, where $1 \le i \le t$, then the length of simple routing path between s and v, $l(s,v) = d_{WK_{(d,t)}}(s,v) = \Sigma_{k=0}^{i-1} 2^k \cdot \sigma(s_k, v_k)$.

Then, they proposed Algorithm 2 to compute the distance for any two vertices s and v. Also, they gave Algorithm 3 and prove that algorithm is a shortest routing algorithm for $\mathrm{WK}_{(d,t)}$.

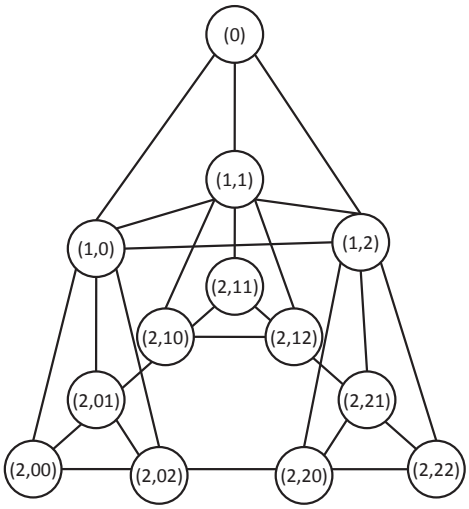


Fig. 3: The structure of $\text{WKP}_{(3,2)}$.

Algorithm 1: A simple routing algorithm for $\text{WK}_{(d,t)}$

Input: source vertex $s = s_{t-1}s_{t-2}...s_1s_0$ and destination vertex $v = v_{t-1}v_{t-2}...v_1v_0$

Output: P and $l(s,v)$ //P is the simple routing path between s and v, and $l(s,v)$ is the length of P.

1 $P \leftarrow s$, $l(s,v) \leftarrow 0$;
2 Examine $s = s_{t-1}s_{t-2}...s_{i+1}s_is_{i-1}...s_1s_0$ and $v = v_{t-1}v_{t-2}...v_{i+1}v_iv_{i-1}...v_1v_0$, find the leftmost index i such that $s_i \neq v_i$;
3 **if** *i can be found* **then**
4 **if** $s_0 \neq v_i$ **then** $r = s_{t-1}s_{t-2}...s_1v_i$;
5 **else** $r = s'$;
6 $P \leftarrow P||r$;
7 $s \leftarrow r$;
8 $l(s,v) = l(s,v) + 1$;
9 goto Line 2;
10 return P and $l(s,v)$;

3. Main Result

The shortest path routing algorithm for $\text{WK}_{(d,t)}$ is proposed by Chen and Duh [2], that is, the shortest path routing algorithm for a layer of $\text{WKP}_{(d,l)}$. Hence, this section will discuss the shortest path routing algorithm for $\text{WKP}_{(d,l)}$.

After reviewing Algorithm 1 and 2 and 3 in Section 2, there are some corollaries.

Corollary 1: For any two vertices $u = u_{t-1}u_{t-2}...u_1u_0$ and $v = v_{t-1}v_{t-2}...v_1v_0$ in $\text{WK}_{(d,t)}$, $l(u,v) = \Sigma_{k=0}^{i-1}2^k \cdot \sigma(u_k, v_k) + 1 + \Sigma_{k=0}^{i-1}2^k \cdot \sigma(u_i, v_k)$, where i is the leftmost index such that $u_{t-1} = v_{t-1}$, $u_{t-2} = v_{t-2}$, ..., $u_i = v_i$ and $u_{i-1} \neq v_{i-1}$.

Corollary 2: For any two vertices $u = u_{t-1}u_{t-2}...u_1u_0$ and $v = v_{t-1}v_{t-2}...v_1v_0$ in $\text{WK}_{(d,t)}$, $d(u,v) = \min\{l(u,v), \Sigma_{k=0}^{i-2}2^k \cdot \sigma(u_k, u_{i-1}) + 1 + (2^{i-1} - 1) + 1 + \Sigma_{k=0}^{i-1}2^k \cdot \sigma(u_{i-1}, v_k), \Sigma_{k=0}^{i-2}2^k \cdot \sigma(u_k, v_{i-1}) + 1 + (2^{i-1} - 1) + 1 + \Sigma_{k=0}^{i-1}2^k \cdot \sigma(v_{i-1}, u_k)\}$, where i is the leftmost index such that $u_{t-1} = v_{t-1}$, $u_{t-2} = v_{t-2}$, ..., $u_i = v_i$ and $u_{i-1} \neq v_{i-1}$.

By the definitions of $\text{WKP}_{(d,l)}$ and $\text{WK}_{(d,t)}$, the above corollaries are suitable on a layer of $\text{WKP}_{(d,l)}$.

Lemma 4: For any two vertices $u = (h, u_{t-1}u_{t-2}...u_1u_0)$ and $v = (h, v_{t-1}v_{t-2}...\ v_1v_0)$ in $\text{WKP}_{(d,l)}$, $1 \leq h \leq l$, $l_{\text{WK}^*_{(d,h)}}(u,v) \geq l_{\text{WK}^*_{(d,h-1)}}(p(u), p(v))$.

Proof. Let i be the leftmost index such that $u_{t-1} = v_{t-1}$, $u_{t-2} = v_{t-2}$, ..., $u_i = v_i$ and $u_{i-1} \neq v_{i-1}$. By Corolleay 1, $l_{\text{WK}^*_{(d,h)}}(u,v) = \Sigma_{k=0}^{i-1}2^k \cdot \sigma(u_k, v_i) + 1 + \Sigma_{k=0}^{i-1}2^k \cdot \sigma(u_i, v_k) \geq \Sigma_{k=1}^{i-1}2^{k-1} \cdot \sigma(u_k, v_i) + 1 + \Sigma_{k=1}^{i-1}2^{k-1} \cdot \sigma(u_i, v_k) = l_{\text{WK}^*_{(d,h-1)}}(p(u), p(v))$. Hence, this lemma is proved. □

Algorithm 2: Distance between two vertices s and v on $\mathrm{WK}_{(d,t)}$

Input: source vertex $s = s_{t-1}s_{t-2}...s_1s_0$ and destination vertex $v = v_{t-1}v_{t-2}...v_1v_0$

Output: $d(s,v)$

Examine $s = s_{t-1}s_{t-2}...s_{i+1}s_is_{i-1}...s_1s_0$ and $v = v_{t-1}v_{t-2}...v_{i+1}v_iv_{i-1}...v_1v_0$ from left to right, find the leftmost index i such that $s_i \neq v_i$;

if *i cannot be found* **then** $d_{WK_{(d,t)}}(s,v) \leftarrow 0$;

else

  Find an (i)-frontier U_r, such that $U_r =_{i-1} s$;

  Find an (i)-frontier V_s, such that $V_s =_{i-1} v$;

  Find an (i)-frontier V_r, such that $V_r =_i v$ and $V'_r =_i U'_r$;

  Find an (i)-frontier U_s, such that $U_s =_i s$ and $U'_s =_i V'_s$;

  Compute $d(s,v) \leftarrow \min\{l(s,v), l(s,U_r) + l(V_r,v) + 2^i + 1, l(s,U_s) + l(V_s,v) + 2^i + 1\}$;

return $d(s,v)$;

Lemma 5: For any two vertices $u = (h, u_{t-1}u_{t-2}...u_1u_0)$ and $v = (h, v_{t-1}v_{t-2}...\ v_1v_0)$ in $\mathrm{WKP}_{(d,l)}$, $1 \leq h \leq l$, $d_{\mathrm{WK}^*_{(d,h)}}(u,v) \geq d_{\mathrm{WK}^*_{(d,h-1)}}(p(u),p(v))$.

Proof: Let i be the leftmost index such that $u_{t-1} = v_{t-1}$, $u_{t-2} = v_{t-2}$, ..., $u_i = v_i$ and $u_{i-1} \neq v_{i-1}$. By Lemma 2, $d_{\mathrm{WK}^*_{(d,h-1)}}(p(u),p(v)) \leq 2^{i-1} - 1$. By Corollary 2, If $d_{\mathrm{WK}^*_{(d,h)}}(u,v)$ equals to $\Sigma_{k=0}^{i-2}2^k \cdot \sigma(u_k,u_{i-1}) + 1 + (2^{i-1} - 1) + 1 + \Sigma_{k=0}^{i-2}2^k \cdot \sigma(u_{i-1},v_k)$ or $\Sigma_{k=0}^{i-2}2^k \cdot \sigma(u_k,v_{i-1}) + 1 + (2^{i-1} - 1) + 1 + \Sigma_{k=0}^{i-2}2^k \cdot \sigma(v_{i-1},u_k)$, $d_{\mathrm{WK}^*_{(d,h)}}(u,v) \geq 2^{i-1} + 1 > d_{\mathrm{WK}^*_{(d,h-1)}}(p(u),p(v))$. Otherwise, we obtain $d_{\mathrm{WK}^*_{(d,h)}}(u,v) = l_{\mathrm{WK}^*_{(d,h)}}(u,v) \geq l_{\mathrm{WK}^*_{(d,h-1)}}(u,v) \geq d_{\mathrm{WK}^*_{(d,h-1)}}(p(u),p(v))$ by Lemma 4. □

According to Lemma 5, we easy to obtain corollary as follows.

Corollary 3: For any two vertices $u = (h, u_{t-1}u_{t-2}...u_1\ u_0)$ and $v = (h, v_{t-1}v_{t-2}...v_1v_0)$ in $\mathrm{WKP}_{(d,l)}$, $1 \leq m \leq h$, $d_{\mathrm{WK}^*_{(d,h)}}(u,v) \geq d_{WK^*_{(d,h-m)}}(p^m(u),p^m(v))$.

Now, we propose a routing algorithm for $\mathrm{WKP}_{(d,l)}$ as Algorithm 4. The inputs of this algorithm is source vertex $s = (k_1, s_{k_1-1}s_{k_1-2}...s_1s_0)$ and destination vertex $d = (k_2, d_{k_2-1}d_{k_2-2}...d_1d_0)$, and the output is a path $\mathrm{SP}(s,d)$.

Algorithm 3: Shortest path routing algorithm on $\mathrm{WK}_{(d,t)}$

Input: source vertex $s = s_{t-1}s_{t-2}...s_1s_0$ and destination vertex $v = v_{t-1}v_{t-2}...v_1v_0$

Output: $\mathrm{SP}(s,v)$ // Shortest path routing algorithm between s and v

Compute $l(s,v)$ by Algorithm 1 with input s and v, and compute $d(s,v)$ by Algorithm 2;

if $d(s,v) = l(s,v)$ **then**

  Executed Algorithm 1 with input s and v, and get output $P(s,v)$;

  SP = $P(s,v)$;

if $d(s,v) = l(s,U_r) + l(V_r,v) + 2^i + 1$ **then**

  Executed Algorithm 1 three times with input s and U_r, U'_r and V'_r, and V_r and v, then get output $P(s,U_r)$, $P(U'_r,V'_r)$ and $P(V_r,v)$;

  $\mathrm{SP}(s,v) = P(s,U_r)||P(U'_r,V'_r)||P(V_r,v)$;

if $d(s,v) = l(s,U_s) + l(V_s,v) + 2^i + 1$ **then**

  Executed Algorithm 1 three times with input s and U_s, U'_s and V'_s, and V_s and v, then get output $P(s,U_s)$, $P(U'_s,V'_s)$ and $P(V_s,v)$;

  $\mathrm{SP}(s,v) = P(s,U_s)||P(U'_s,V'_s)||P(V_s,v)$;

return $\mathrm{SP}(s,v)$;

The main idea is first to find a pair of vertices $s' = p^{k_1-k_2}(s)$ and $d' = d$ on the same layer k_2 if $k_1 > k_2$ or $s' = s$ and $d' = p^{k_1-k_2}(d)$, if $k_2 > k_1$. Next, algorithm computes the distance between s' and d' and distance between $p(s')$ and $p(d')$, and adds suitable vertices into the output path.

4. Conclusions

This paper presents a path routing algorithm for the basic WK-recursive pyramids. In fact, we have several reasons to believe the output path is a shortest path between two input vertices in the basic WK-recursive pyramids. We will prove it completely in the near future.

5. Acknowledgements

This research was supported in part by the Ministry of Science and Technology of the Republic of China under grant MOST 104-2221-E-260-005 - .

Algorithm 4: A path routing algorithm for $\mathrm{WKP}_{(d,l)}$

Input: source vertex $s = (k_1, s_{k_1-1}s_{k_1-2}...s_1s_0)$ and destination vertex $d = (k_2, d_{k_2-1}d_{k_2-2}...d_1d_0)$

Output: A path $\mathrm{SP}(s, d)$

$H \leftarrow$ Null; $T \leftarrow$ Null ;
while $k_1 > k_2$ **do**
    $H \leftarrow H||s$; $s \leftarrow p(s)$;
while $k_1 < k_2$ **do**
    $T \leftarrow d||T$; $d \leftarrow p(d)$;
while $d(s, d) > 2 + d(p(u), p(v))$ **do**
    $H \leftarrow H||s$; $s \leftarrow p(s)$;
    $T \leftarrow d||T$; $d \leftarrow p(d)$;
Executes Algorithm 3 with inputs s and d (ignore k_1 and k_2), and obtains outpath as $\mathrm{SP}(s, d)$ (add k_1 and k_2);
return $H||\mathrm{SP}(s, d)||T$;

References

[1] G. Della Vecchia and C. Sanges, "Recursively scalable networksfor message passing architectures," *Parallel Processing and Applications*, pp. 33–40, 1987.

[2] G. H. Chen and D. R. Duh, "Topological properties, communication, and computation on WK-recursive network," *Networks*, vol. 24, no. 6, pp. 303–317, 1994.

[3] M. H. Farahabady, N. Imani, and H. Sarbazi-Azad, "Some topological and combinatorial properties of WK-recursive mesh and wk-pyramid interconnection networks," *Journal of Systems Architecture*, vol. 54, no. 10, pp. 967–976, 2008.

[4] J. S. Fu, "Hamiltonicity of the WK-recursive network with and without faulty nodes," *IEEE Transactions on Parallel and Distribured Systems*, vol. 16, no. 9, pp. 853–865, Sep 2005.

[5] R. Fernandes, "Recursive interconnection networks for multicomputer networks," in *Proceedings of the 1992 International Conference on Parallel Processing*, vol. I, 1992, pp. 76–79.

[6] R. Fernandes and A. Kanevsky, "Hierarchical WK-recursive topologies for multicomputer systems," in *Proceedings of the 1993 International Conference on Parallel Processing*, vol. I, 1993, pp. 315–318.

[7] Y.-C. Wang and J. S.-T. Juan, "The m-pancycle-connectivity of basic WK-recursive pyramids," in *Proceeding of the 30nd Workshop on Combinatorial Mathematics and Computational Theory*, Hualien, Taiwan, Apr, 2013, pp. 103–108.

[8] Y.-C. Wang, *A Study on Basic WK-Recursive Pyramids and Triangular Pyramids*. Ph.D. Thesis, Department of Computer Science and Information Engineering National Chi Nan University, Puli, Nantou Hsien, Taiwan, 2015.

[9] Y.-C. Wang and J. S.-T. Juan, "Hamiltonicity of the basic WK-recursive pyramid with and without faulty nodes," *Theoretical Computer Science*, vol. 562, no. 11, pp. 542–556, 2015.

[10] D.-R. Duh and G.-H. Chen, "Topological properties of WK-recursive network," *Journal of Parallel and Distributed Computing*, vol. 23, pp. 468–474, 1994.

[11] D. Zivkovic, "Hamiltonianicity of the tower of hanoi problem." *Univ. Beograd. Publ. Elektrotehn. Fak. Ser. Mat.*, vol. 17, pp. 31–37, 2006.

An Algorithm for k-pairwise Cluster-fault-tolerant Disjoint Paths in a Burnt Pancake Graph

Masato Tokuda, Yuki Hirai, and Keiichi Kaneko
Graduate School of Engineering, Tokyo University of Agriculture and Technology, Koganei-shi, Tokyo, Japan
{s149510s@st.go,yhirai@cc,k1kaneko@cc}.tuat.ac.jp

Abstract— *In this paper, we focus on the pairwise cluster-fault-tolerant disjoint paths routing problem in a burnt pancake graph, and propose an algorithm that solves the problem in a polynomial time of the degree of the graph. That is, in a n-burnt pancake graph with $(n-2k+1)$ faulty clusters whose diameters are at most 3, the algorithm can construct fault-free disjoint paths between k pairs of nodes. The time complexity of the algorithm is $O(kn^3)$ and the maximum path length is $2n+13$. We have conducted a computer experiment and its results showed that there was not any path that attained the theoretical maximum path length and the average time complexity of the algorithm is $O(n^{2.2})$.*

Keywords: Cayley graph, multicomputer, interconnection network, parallel processing

1. Introduction

In the near future, processing performance of a sequential computer is expected to reach a ceiling because of limitations in technology. With this expectation, the field of parallel and distributed computation is taking on increasing importance, and studies on massively parallel computers are eagerly conducted recently. An interconnection network provides a topology to construct a massively parallel computer, and many topologies have been proposed and studied to interconnect many computers.

One of the factors that determine the performance of an interconnection network is fault tolerance. As the number of processors in a parallel computer increases, the probability of existence of faulty processors also increases. In practice, we face with the situation where not only the single processor fault but also a set of faulty nodes will arise. Hence, to address a fault-tolerant routing problem in a graph with multiple cluster faults has a merit to establish a fault-free communication, and there are many research activities about it. Similarly, to address a disjoint paths problem has a merit to establish full-bandwidth communication that gets no interference from other communication, and it is also studied very hard.

To solve the pairwise disjoint paths in a given graph with degree n is to find $\lceil n/2 \rceil$ disjoint paths $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_i$ $(1 \leq i \leq \lceil n/2 \rceil)$ for two arbitrary sets of nodes $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{\lceil n/2 \rceil}\}$ and $T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{\lceil n/2 \rceil}\}$ in the graph. Practically, it is crucial to find disjoint paths in a network since they make it possible to use the full bandwidth and enhance communication reliability. Therefore, solving the pairwise disjoint paths problem [1], [2], [3] is important as well as solving the node-to-node disjoint paths problem [4], [5], [6], [7], [8], the node-to-set disjoint paths problem [9], [10], [11], [12], [13], [14], and the set-to-set disjoint paths problem [15], [16], [17].

In this paper, we have focused on a burnt pancake graph [18], [19], [20], [21], [5], which is derived from a pancake graph [22], [23], [24], [25], [26], [27], [8] of a Cayley graph. A burnt pancake graph can connect many nodes with a small degree. Also, burnt pancake graphs are expected to fill in the gaps of incremental expandability of pancake graphs because they can connect different numbers of nodes from pancake graphs. However, there are many unsolved problem with a burnt pancake graph such as the shortest-path routing problem, the pairwise cluster-fault-tolerant disjoint paths routing problem, and so on.

In this paper, we pick up the pairwise cluster-fault-tolerant routing problem among the unsolved problems in a burnt pancake graph. For this problem, we propose an algorithm that solves it in a polynomial time of the degree of the burnt pancake graph. That is, in a n-burnt pancake graph with at most $(n-2k+1)$ faulty clusters whose diameters are at most 3, for k pairs of the source and destination nodes, we prove that our algorithm can construct k fault-free disjoint paths between them. We also prove that the time complexity of the algorithm is $O(kn^3)$ and the maximum path length is $2n+13$.

2. Preliminaries

In this section, we first introduce a definition of a burnt pancake graph and related definitions.

Definition *1:* A permutation $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ that satisfies that $\{|u_1|, |u_2|, \ldots, |u_n|\} = \langle n \rangle$ is called a signed permutation where $\langle n \rangle = \{1, 2, \ldots, n\}$.

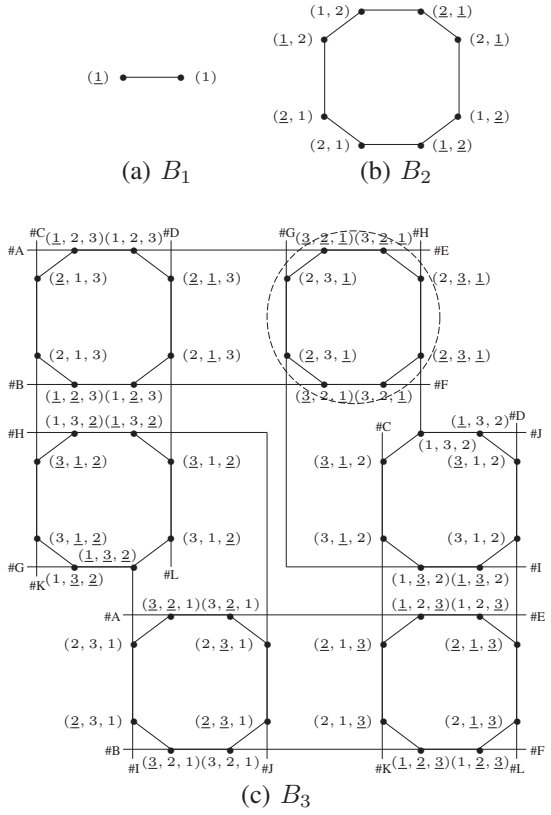
Definition *2:* For a signed permutation $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and an integer i $(1 \leq i \leq n)$, the signed prefix reversal operation $\boldsymbol{u}^{(i)}$ is defined by $\boldsymbol{u}^{(i)} = (-u_i, -u_{i-1}, \ldots, -u_2, -u_1, u_{i+1}, \ldots, u_n)$.

ISBN: 1-60132-444-8, CSREA Press ©

We use the notation $\boldsymbol{u}^{(i,\ldots,j,k)}$ as a short hand of a signed prefix reversal operation $\boldsymbol{u}^{(i,\ldots,j)(k)}$. A signed prefix reversal operation is invertible and $\boldsymbol{u}^{(i,i)} = \boldsymbol{u}$ holds.

Definition *3:* If a graph $G(V, E)$ satisfies the conditions that $V = \{(u_1, u_2, \ldots, u_n) | (u_1, u_2, \ldots, u_n)$ is a signed permutation of $\langle n \rangle\}$ and $E = \{(\boldsymbol{u}, \boldsymbol{u}^{(i)}) | \boldsymbol{u} \in V, 1 \leq i \leq n\}$, G is called an n-burnt pancake graph.

In this paper, we denote B_n and $\underline{i}$ to represent an n-burnt pancake graph and $-i$, respectively. Figure 1 shows examples of burnt pancake graphs, B_1, B_2, and B_3.



(a) B_1 (b) B_2

(c) B_3

Fig. 1: Examples of burnt pancake graphs.

A B_n is a symmetric graph, and the number of nodes, the number of edges, the degree, and the connectivity are $n! \times 2^n$, $n! \times n \times 2^{n-1}$, n, and n, respectively. There is no shortest-path routing algorithm found for a B_n in time complexity of the polynomial order of n. However, the fact that $d(B_n) \leq 2n$ is proved.

Definition *4:* In a B_n, for an arbitrary node $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and an arbitrary integer k ($1 \leq |k| \leq n$), an extended signed prefix reversal operation $\boldsymbol{u}^{([k])}$ is defined by

$$\boldsymbol{u}^{([k])} = \begin{cases} \boldsymbol{u}^{(i)} & (u_i = \underline{k}) \\ \boldsymbol{u}^{(i)} \rightarrow \boldsymbol{u}^{(i,1)} & (u_i = k) \end{cases}$$

Definition *5:* In a B_n, the sub graph induced by the subset of nodes that have k at the rightmost positions in their permutations is isomorphic to a B_{n-1}. The sub graph is specified by $B_{n-1}(k)$ by using the k as its index. A B_n is decomposable into $2n$ B_{n-1}'s that are mutually disjoint. Each sub graph is called a sub burnt pancake graph. In addition, the sub burnt pancake graph that contains the node $\boldsymbol{u} (\in B_n)$ is denoted by $B_{n-1}(\boldsymbol{u})$.

In Figure 1, the sub graph indicated by the dashed circle is specified by $B_2(\underline{1})$, which is isomorphic to a B_2.

Definition *6:* A connected sub graph in a graph is called a cluster. If all of the nodes in a cluster are faulty, the cluster is called a faulty cluster. In addition, in a graph $G(V, E)$ the nodes defined by $\arg\min_{\boldsymbol{c} \in V} \max_{\boldsymbol{v} \in V} d(\boldsymbol{c}, \boldsymbol{v})$ are called the centers of the graph G.

Definition *7:* In a B_n with faulty clusters, if a $B_{n-1}(k)$ does not include any center of the faulty clusters, it is called a candidate sub burnt pancake graph and denoted by $CB_{n-1}(k)$.

Definition *8:* The set that consists of the nodes that have $\underline{j}$ and i at the leftmost and rightmost positions in their permutations, respectively, is called a portset from $B_{n-1}(i)$ to $B_{n-1}(j)$, and denoted by $P(i, j)$.

Theorem *1:* In a B_n, for two non-faulty nodes $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$, $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)$ and a set of faulty nodes F ($|F| \leq n-1$), we can construct a fault-free path between $\boldsymbol{s}$ and $\boldsymbol{t}$ of length at most $2n+4$ in time complexity $O(n^2)$.
(Proof) See [19]. □

3. Algorithm

In this section, we show an algorithm that solves the pairwise cluster-fault-tolerant disjoint paths problem in a burnt pancake graph.

3.1 Lemmas

Lemma *1:* For two distinct nodes $\boldsymbol{u}$ and $\boldsymbol{v}$ in a port set $P(l, m)$ ($1 \leq |l|, |m| \leq n$, $|l| \neq |m|$), the distance between them $d(\boldsymbol{u}, \boldsymbol{v})$ is no less than 3.
(Proof) Let $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ be two distinct nodes in a port set $P(l, m)$. From assumption, $u_1 = v_1 = \underline{m}$ and $u_n = v_n = l$ hold. If $d(\boldsymbol{u}, \boldsymbol{v}) = 0$, $\boldsymbol{u} = \boldsymbol{v}$ holds. It is contradictory to the fact that $\boldsymbol{u}$ and $\boldsymbol{v}$ are distinct. If $d(\boldsymbol{u}, \boldsymbol{v}) = 1$, $\boldsymbol{u}$ and $\boldsymbol{v}$ are adjacent. For any two adjacent nodes in B_n, the elements in their leftmost positions are different. However, it is contradictory to the

fact that $\boldsymbol{u}, \boldsymbol{v} \in P(l, m)$. If $d(\boldsymbol{u}, \boldsymbol{v}) = 2$, $\exists i, j$ $(i \neq j)$ such that $\boldsymbol{u} \rightarrow \boldsymbol{u}^{(i)} \rightarrow \boldsymbol{u}^{(i,j)} = \boldsymbol{v} \in P(l, m)$. However, if $i \neq j$, $u_1 \neq v_1$ holds. It means that $\boldsymbol{u}$ and $\boldsymbol{v}$ do not belong to a single port set $P(l, m)$. Consequently, Lemma 1 holds. □

Lemma *2:* There is not a cycle in a B_n whose length is less than 8.
(Proof) We prove this lemma based on mathematical induction. In B_1, there is no cycle and B_2 is a cycle of length 8. Hence, this lemma holds for $n \leq 2$. Now, we prove that Lemma 2 holds for B_n with the hypothesis that the lemma holds for B_{n-1} $(n \geq 3)$. If there is a cycle C whose length is less than 8, C has an edge between two sub burnt pancake graphs since this lemma holds for any sub burnt pancake graphs by hypothesis. To be a cycle, C must have at least two such edges. If C has exactly two edges $(\boldsymbol{a}_0, \boldsymbol{b}_0)$ and $(\boldsymbol{a}_1, \boldsymbol{b}_1)$ between sub burnt pancake graphs, we can assume that $\boldsymbol{a}_0, \boldsymbol{a}_1 \in P(l, m)$ and $\boldsymbol{b}_0, \boldsymbol{b}_1 \in P(m, l)$ without loss of generality. Then, from Lemma 1, $d(\boldsymbol{a}_0, \boldsymbol{a}_1) \geq 3$ and $d(\boldsymbol{b}_0, \boldsymbol{b}_1) \geq 3$ hold. Therefore, the length of C is at least 8. It is a contradiction, and it means that C does not exist. If C has exactly three edges between sub burnt pancake graphs $(\boldsymbol{a}_0, \boldsymbol{b}_0)$, $(\boldsymbol{a}_1, \boldsymbol{b}_1)$, and $(\boldsymbol{a}_2, \boldsymbol{b}_2)$, we can assume that $\boldsymbol{a}_0 \in P(j, l)$, $\boldsymbol{b}_0 \in P(l, j)$, $\boldsymbol{a}_1 \in P(l, m)$, $\boldsymbol{b}_1 \in P(m, l)$, $\boldsymbol{a}_2 \in P(m, j)$, and $\boldsymbol{b}_2 \in P(j, m)$ without loss of generality. Since the length of C is less than 8, at least two of the distances $d(\boldsymbol{a}_0, \boldsymbol{b}_2)$, $d(\boldsymbol{a}_1, \boldsymbol{b}_0)$, and $d(\boldsymbol{a}_2, \boldsymbol{b}_1)$ must be 1. Here, without loss of generality, we can assume that $d(\boldsymbol{a}_0, \boldsymbol{b}_2) = 1$ and $d(\boldsymbol{b}_0, \boldsymbol{a}_1) = 1$ hold. Then, since $\boldsymbol{b}_2 \in P(j, m)$, $\boldsymbol{b}_2 = (\underline{m}, \ldots, j)$ holds. In addition, because $\boldsymbol{a}_0 \in P(j, l)$ and $d(\boldsymbol{a}_0, \boldsymbol{b}_2) = 1$ hold, $\boldsymbol{a}_0 = (\underline{l}, \ldots, m, \ldots, j)$ holds. Hence, $\boldsymbol{b}_0 = \boldsymbol{a}_0^{(n)} = (\underline{j}, \ldots, \underline{m}, \ldots, l)$ holds. On the other hand, $\boldsymbol{a}_1 = (\underline{m}, \ldots, \overline{l})$ holds since $\boldsymbol{a}_1 \in P(l, m)$. Then, from the sign of m, $\boldsymbol{b}_0$ and $\boldsymbol{a}_1$ cannot be adjacent. Therefore, $d(\boldsymbol{b}_0, \boldsymbol{a}_1) \neq 1$ holds. It is a contradiction, and C does not exist. Finally, if C has 4 or more edges between sub burnt pancake graphs, still 4 or more edges are necessary to make C be a cycle. It means that the length of C is at least 8. From the discussion above, we have proved this lemma.□

Lemma *3:* In a B_n, if there are at most $(n - 2k + 1)$ faulty clusters whose diameters are at most 3, there are at least $(4k - 2)$ candidate sub burnt pancake graphs.
(Proof) There are $2n$ sub burnt pancake graphs. Because a cluster whose diameter is at most 3 has at most two centers, there are at least $(2n - 2(n - 2k + 1) = 4k - 2)$ candidate sub burnt pancake graphs. □

Lemma *4:* In a B_n, for a node $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$, we can construct n disjoint paths of length at most 3 from $\boldsymbol{u}$ to n distinct sub burnt pancake graphs $B_{n-1}(k)$ $(k \neq |u_n|)$.
(Proof) We can construct the paths of lengths at most 3 from $\boldsymbol{u}$ to $(2n - 2)$ sub burnt pancake graphs as follows:

$$\begin{cases} \boldsymbol{u} \rightarrow \boldsymbol{u}^{(n)} \in B_{n-1}(\underline{u}_1) \\ \boldsymbol{u} \rightarrow \boldsymbol{u}^{(i)} \rightarrow \boldsymbol{u}^{(i,n)} \in B_{n-1}(u_i) \quad (1 \leq i \leq n-1) \\ \boldsymbol{u} \rightarrow \boldsymbol{u}^{(i)} \rightarrow \boldsymbol{u}^{(i,1)} \rightarrow \boldsymbol{u}^{(i,1,n)} \in B_{n-1}(\underline{u}_i) \\ \qquad\qquad\qquad\qquad\qquad\qquad (2 \leq i \leq n-1) \end{cases}$$

Among these $(2n - 2)$ paths we can select n disjoint paths $\boldsymbol{u} \rightsquigarrow \boldsymbol{u}^{(n)}$, $\boldsymbol{u} \rightsquigarrow \boldsymbol{u}^{(1,n)}$, $\boldsymbol{u} \rightsquigarrow \boldsymbol{u}^{(2,n)}$, $\ldots$, $\boldsymbol{u} \rightsquigarrow \boldsymbol{u}^{(n-1,n)}$, for instance. □

Figure 2 shows $(2n - 2)$ candidate paths from a node $\boldsymbol{u}$ to $(2n - 2)$ distinct sub burnt pancake graphs.

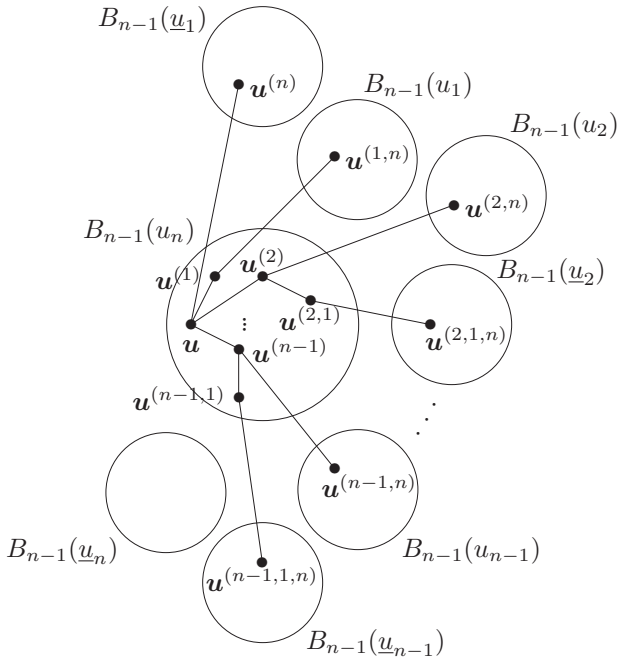


Fig. 2: $(2n - 2)$ paths from a node $\boldsymbol{u}$ to $(2n - 2)$ distinct sub burnt pancake graphs constructed in Lemma 4.

Lemma *5:* In a B_n, for a non-faulty node $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and n candidate sub burnt pancake graphs $CB_{n-1}(k)$ $(k \neq |u_n|)$, a faulty cluster whose diameter is at most 3 can overlap at most one of the n paths of length at most 3 that are given in Lemma 2.
(Proof) Let P: $\boldsymbol{u} \rightsquigarrow \boldsymbol{v}$ and Q: $\boldsymbol{u} \rightsquigarrow \boldsymbol{w}$ are two distinct paths among the n paths given in Lemma 4. Then, from Lemma 2, a cluster cannot overlap simultaneously the nodes on P and Q inside $B_{n-1}(u_n)$. Also, from Lemma 1, it is impossible for a cluster to overlap a node on P inside $B_{n-1}(u_n)$ and the node $\boldsymbol{w}$ on Q simultaneously. Finally, let us construct a path of length 2 R: $\boldsymbol{v} \rightarrow \boldsymbol{x} \rightarrow \boldsymbol{x}^{(n)}$ so that $\boldsymbol{x}^{(n)}$ is in the same sub burnt pancake graph as $\boldsymbol{w}$. Then, because $\boldsymbol{v} = (\underline{u}_n, \ldots)$, $\boldsymbol{x}$ contains u_n and $\boldsymbol{x}^{(n)}$ contains $\underline{u}_n$. Since $\boldsymbol{w} = (\underline{u}_n, \ldots)$, $\boldsymbol{w}$ and $\boldsymbol{x}^{(n)}$ cannot be adjacent. Hence, $d(\boldsymbol{v}, \boldsymbol{w}) > 3$. From the discussion above, this lemma holds.

Lemma *6:* In a B_n, for a node $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and a sub burnt pancake graph $B_{n-1}(k)$, $(k \neq u_1, |u_n|)$, we can construct n disjoint paths of length at most 5 from $\boldsymbol{u}$ to the $B_{n-1}(k)$ in $O(n^2)$ time complexity. (Proof) If $|k| \neq |u_1|, |u_n|$, we can construct n paths of lengths at most 3 as follows:

$$\begin{cases} \boldsymbol{u} \to \boldsymbol{u}^{(i)} \to \boldsymbol{u}^{(i,[k])} \to \boldsymbol{u}^{(i,[k],n)} & (1 \leq i \leq n, |u_i| \neq |k|) \\ \boldsymbol{u} \to \boldsymbol{u}^{(i)} \to \boldsymbol{u}^{(i,n)} & (u_i = k) \\ \boldsymbol{u} \to \boldsymbol{u}^{(i)} \to \boldsymbol{u}^{(i,1)} \to \boldsymbol{u}^{(i,1,n)} & (u_i = \underline{k}) \end{cases}$$

If $k = \underline{u_n}$, we can construct n paths of lengths at most 4 as follows:

$$\begin{cases} \boldsymbol{u} \to \boldsymbol{u}^{(i)} \to \boldsymbol{u}^{(i,n)} \to \boldsymbol{u}^{(i,n,i)} \to \boldsymbol{u}^{(i,n,1,n)} \\ \qquad\qquad\qquad\qquad\qquad\qquad (1 \leq i \leq n-1) \\ \boldsymbol{u} \to \boldsymbol{u}^{(n)} \to \boldsymbol{u}^{(n,1)} \to \boldsymbol{u}^{(n,1,n)} \quad (i = n) \end{cases}$$

If $k = u_1$, we can construct n paths of lengths at most 5 as follows:

$$\begin{cases} \boldsymbol{u} \to \boldsymbol{u}^{(1)} \to \boldsymbol{u}^{(1,n)} & (i = 1) \\ \boldsymbol{u} \to \boldsymbol{u}^{(i)} \to \boldsymbol{u}^{(i,1)} \to \boldsymbol{u}^{(i,1,i)} \to \boldsymbol{u}^{(i,1,i,1)} & \\ \to \boldsymbol{u}^{(i,1,i,1,i)} & (2 \leq i \leq n) \end{cases}$$

From above discussion, we can construct n paths from $\boldsymbol{u}$ to $B_{n-1}(k)$ of lengths at most 5 that are disjoint except for $\boldsymbol{u}$. Also, it takes $O(n)$ time to construct a path. Hence, it takes $O(n^2)$ in total to construct n paths.

Lemma *7:* For $CB_{n-1}(p)$ and distinct $(n-1)$ $CB_{n-1}(l_i)$'s $(1 \leq |l_i| \leq n,\ 1 \leq i \leq n,\ |p| \neq \forall |l_i|$, and $|l_i| \neq |l_j|$ for $i \neq j)$ such that each CB_{n-1} has at most $(n-2)$ faulty nodes, we can construct $(n-1)$ disjoint fault-free paths of lengths 6 each of which is from an arbitrary node $\boldsymbol{u}_{l_i}$ in each $CB_{n-1}(l_i)$ to $CB_{n-1}(l_i)$ via $CB_{n-1}(p)$ in the time complexity $O(n^3)$.
(Proof) Because $CB_{n-1}(l_i)$, $CB_{n-1}(\underline{l_i})$, and $CB_{n-1}(p)$ are sub burnt pancake graphs, $P(l_i, p)$, $P(p, l_i)$, $P(p, \underline{l_i})$, and $P(\underline{l_i}, p)$ do not include any faulty node. Therefore, if $\boldsymbol{u}_{l_i,1} = \underline{p}$, we can construct a fault-free path $\boldsymbol{u}_{l_i,1} \to \boldsymbol{u}_{l_i,1}^{(n)} (\in CB_{n-1}(p)) \to \boldsymbol{u}_{l_i,1}^{(n,1)} \to \boldsymbol{u}_{l_i,1}^{(n,1,n)} (\in CB_{n-1}(\underline{l_i})$ of length 3. Moreover, if $\boldsymbol{u}_{l_i,1} = p$, we can construct a fault-free path $\boldsymbol{u}_{l_i,1} \to \boldsymbol{u}_{l_i,1}^{(1)} \to \boldsymbol{u}_{l_i,1}^{(1,n)} (\in CB_{n-1}(p)) \to \boldsymbol{u}_{l_i,1}^{(1,n,1)} \to \boldsymbol{u}_{l_i,1}^{(1,n,1,n)} (\in CB_{n-1}(\underline{l_i})$ of length 4.

For each $\boldsymbol{u}_{l_i}$ with $u_{l_i,j} = \underline{p}$ and $j \geq 2$, we can construct $(n-1)$ disjoint paths R_h $(1 \leq h \leq n-1)$ from $\boldsymbol{u}_{l_i}$ to $CB_{n-1}(\underline{l_i})$ via $CB_{n-1}(p)$ as follows:

$$\begin{cases} \boldsymbol{u}_{l_i} \to \boldsymbol{u}_{l_i}^{(h)} \to \boldsymbol{u}_{l_i}^{(h,j)} \to \boldsymbol{u}_{l_i}^{(h,j,1)} \\ \to \boldsymbol{u}_{l_i}^{(h,j,1,n)} (\in CB_{n-1}(p)) \to \boldsymbol{u}_{l_i}^{(h,j,1,n,1)} \\ \to \boldsymbol{u}_{l_i}^{(h,j,1,n,1,n)} (\in CB_{n-1}(\underline{l_i})) \qquad (1 \leq h \leq j-1) \\ \boldsymbol{u}_{l_i} \to \boldsymbol{u}_{l_i}^{(h)} \to \boldsymbol{u}_{l_i}^{(h,1)} \to \boldsymbol{u}_{l_i}^{(h,1,n)} (\in CB_{n-1}(p)) \\ \to \boldsymbol{u}_{l_i}^{(h,1,n,1)} \to \boldsymbol{u}_{l_i}^{(h,1,n,1,n)} (\in CB_{n-1}(\underline{l_i})) \qquad (h = j) \\ \boldsymbol{u}_{l_i} \to \boldsymbol{u}_{l_i}^{(h)} \to \boldsymbol{u}_{l_i}^{(h,h-j+1)} \to \boldsymbol{u}_{l_i}^{(h,h-j+1,1)} \\ \to \boldsymbol{u}_{l_i}^{(h,h-j+1,1,n)} (\in CB_{n-1}(p)) \to \boldsymbol{u}_{l_i}^{(h,h-j+1,1,n,1)} \\ \to \boldsymbol{u}_{l_i}^{(h,h-j+1,1,n,1,n)} (\in CB_{n-1}(\underline{l_i})) \\ \qquad\qquad\qquad\qquad\qquad\qquad (j+1 \leq h \leq n-1) \end{cases}$$

For each $\boldsymbol{u}_{l_i}$ with $u_{l_i,j} = p$ and $j \geq 2$, we can construct $(n-1)$ disjoint paths R_h $(1 \leq h \leq n-1)$ from $\boldsymbol{u}_{l_i}$ to $CB_{n-1}(\underline{l_i})$ via $CB_{n-1}(p)$ as follows:

$$\begin{cases} \boldsymbol{u}_{l_i} \to \boldsymbol{u}_{l_i}^{(h)} \to \boldsymbol{u}_{l_i}^{(h,j)} \to \boldsymbol{u}_{l_i}^{(h,j,n)} (\in CB_{n-1}(p)) \\ \to \boldsymbol{u}_{l_i}^{(h,j,n,1)} \to \boldsymbol{u}_{l_i}^{(h,j,n,1,n)} (\in CB_{n-1}(\underline{l_i})) \\ \qquad\qquad\qquad\qquad\qquad\qquad (1 \leq h \leq j-1) \\ \boldsymbol{u}_{l_i} \to \boldsymbol{u}_{l_i}^{(h)} \to \boldsymbol{u}_{l_i}^{(h,n)} (\in CB_{n-1}(p)) \to \boldsymbol{u}_{l_i}^{(h,n,1)} \\ \to \boldsymbol{u}_{l_i}^{(h,n,1,n)} (\in CB_{n-1}(\underline{l_i})) \qquad (h = j) \\ \boldsymbol{u}_{l_i} \to \boldsymbol{u}_{l_i}^{(h)} \to \boldsymbol{u}_{l_i}^{(h,h-j+1)} \\ \to \boldsymbol{u}_{l_i}^{(h,h-j+1,n)} (\in CB_{n-1}(p)) \to \boldsymbol{u}_{l_i}^{(h,h-j+1,n,1)} \\ \to \boldsymbol{u}_{l_i}^{(h,h-j+1,n,1,n)} (\in CB_{n-1}(\underline{l_i})) \\ \qquad\qquad\qquad\qquad\qquad\qquad (j+1 \leq h \leq n-1) \end{cases}$$

Then, we can find a fault-free path among the above $(n-1)$ paths for each $\boldsymbol{u}_{l_i}$. Each path construction takes $O(1)$ time. It takes $O(n)$ time to check whether a path is fault-free or not. Hence, it takes $O(n^2)$ time to find a fault-free path for one $\boldsymbol{u}_{l_i}$. Therefore, construction of $(n-1)$ paths takes $O(n^3)$ time in total.

Figure 3 shows the $(n-1)$ fault-free disjoint paths of length at most 6 constructed in Lemma 7.

3.2 Algorithm Description

In this section, we describe the details of the algorithm for cluster-fault-tolerant k-pairwise disjoint path routing, and estimate the maximum path length and its time complexity. In a B_n, for k pairs of source and destination nodes $(3 \leq k \leq \lceil n/2 \rceil)$, the algorithm first constructs paths $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}'_i$ and $\boldsymbol{t}_i \rightsquigarrow \boldsymbol{t}'_i$ where $\boldsymbol{s}'_i$ and $\boldsymbol{t}'_i$ belong to a same B_{n-1} and the paths do not include any node on the other paths $\boldsymbol{s}_j \rightsquigarrow \boldsymbol{s}'_j$ nor $\boldsymbol{t}_j \rightsquigarrow \boldsymbol{t}'_j$ where $j \neq i$. Then, it connects $\boldsymbol{s}'_i$ and $\boldsymbol{t}'_i$ by a fault-free path in the B_{n-1} by using the fault-tolerant routing algorithm. The B_{n-1} is called the target sub burnt pancake graph for the pair of nodes $\boldsymbol{s}_i$ and $\boldsymbol{t}_i$, and denoted by $B_{n-1}(l_i)$ $(1 \leq |l_i| \leq n,\ 1 \leq i \leq k)$. Also, in the rest of this paper, we assume that the candidate sub burnt pancake graph for $\boldsymbol{s}_i$ satisfies the condition that it does not include any nodes on the other paths $\boldsymbol{s}_j \rightsquigarrow \boldsymbol{s}'_j$ nor $\boldsymbol{t}_j \rightsquigarrow \boldsymbol{t}'_j$ in

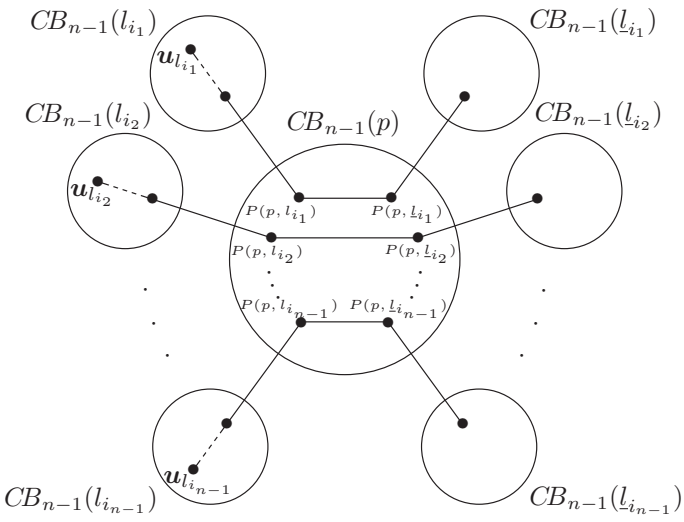


Fig. 3: $(n-1)$ fault-free disjoint paths of length at most 7 constructed in Lemma 7.

addition to the condition that it does not include any center of faulty clusters.

The algorithm consists of the following four steps.

Step 1) If there is a $B_{n-1}(m)$ that contains $\boldsymbol{s}_i$ or $\boldsymbol{t}_i$ and the $B_{n-1}(m)$ is a candidate sub burnt pancake graph for $\boldsymbol{s}_i$ or $\boldsymbol{t}_i$, assign the $B_{n-1}(m)$ to the target sub burnt pancake graph $B_{n-1}(l_i)$. If $\boldsymbol{s}_i$ and $\boldsymbol{t}_i$ are included in distinct $B_{n-1}(p)$ and $B_{n-1}(q)$, respectively, and both of $B_{n-1}(p)$ and $B_{n-1}(q)$ satisfy the conditions of candidate sub burnt pancakes for $\boldsymbol{s}_i$ and $\boldsymbol{t}_i$, either of them are assigned to $B_{n-1}(l_i)$. We can assign a target sub burnt pancake graph for each pair of source and destination nodes in $O(n)$ time.

Step 2) For each pair of the source node $\boldsymbol{s}_i = (s_{i1}, s_{i2}, \ldots, s_{in})$ and the destination nodes $\boldsymbol{t}_i$ to which any target sub burnt pancake graph is not assigned, construct a path from either of the source or destination nodes to a candidate sub burnt pancake graph for it. Here, we assume that we found a candidate sub burnt pancake graph for $\boldsymbol{s}_i$. Then, we can assign a target sub burnt pancake graph and construct a path by the following three sub steps.

Sub Step 2a) If the $B_{n-1}(\underline{s}_{i1})$ is a candidate sub burnt pancake graph for $\boldsymbol{s}_i$, we assign $B_{n-1}(\underline{s}_{i1})$ to the target sub burnt pancake graph $B_{n-1}(l_i)$, construct a path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}_i^{(n)}$ of length 1 to the sub burnt pancake graph, and let $\boldsymbol{s}_i^{(n)} = \boldsymbol{s}'_i$.

Sub Step 2b) If the $B_{n-1}(s_{ip})$ $(1 \leq p < n)$ is a candidate sub burnt pancake graph for $\boldsymbol{s}_i$, we try to construct a path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}_i^{(p)} \rightsquigarrow \boldsymbol{s}_i^{(p,n)}$ of length 2. If this path is fault-free and disjoint from other paths, we can assign $B_{n-1}(s_{ip})$ to the target sub burnt pancake graph $B_{n-1}(l_i)$, and let $\boldsymbol{s}_i^{(p,n)} = \boldsymbol{s}'_i$.

Sub Step 2c) If the $B_{n-1}(\underline{s}_{ip})$ $(1 < p < n)$ is a candidate sub burnt pancake graph for $\boldsymbol{s}_i$, we try to construct a path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}_i^{(p)} \rightsquigarrow \boldsymbol{s}_i^{(p,1)} \rightsquigarrow \boldsymbol{s}_i^{(p,1,n)}$ of length 3. If this path is fault-free and disjoint from other paths, we can assign $B_{n-1}(\underline{s}_{ip})$ to the target sub burnt pancake graph $B_{n-1}(l_i)$, and let $\boldsymbol{s}_i^{(p,1,n)} = \boldsymbol{s}'_i$.

In Step 2, we can assign a target sub burnt pancake graph for each pair of source and destination nodes by constructing a path of length at most 3 in $O(n^3)$ time.

Step 3) By Steps 1 and 2, for k pairs of nodes $\boldsymbol{s}_i$ and $\boldsymbol{t}_i$, target sub burnt pancake graphs $B_{n-1}(l_i)$ are assigned and at least one path from either of the nodes is constructed. Here, we construct a path to $B_{n-1}(l_i)$ from either of $\boldsymbol{s}_i$ or $\boldsymbol{t}_i$ from which a path to $B_{n-1}(l_i)$ has not been constructed. For simplicity, we assume that a path from $\boldsymbol{s}_i$ to $B_{n-1}(l_i)$ has been already constructed, and a path from $\boldsymbol{t}_i$ has not been constructed without loss of generality. Here, if $t_{i,1}, \underline{t}_{i,1}, \underline{t}_{i,n} \neq l_i$, consider the $(n-1)$ paths of lengths at most 4 given in Lemma 6 excluding one path that includes $\boldsymbol{t}_i^{(n)}$. If there is a path among them that is fault-free and disjoint from other constructed paths, let the path be $\boldsymbol{t}_i \rightsquigarrow \boldsymbol{t}'_i$. If $t_{i,1} = l_i$, check whether the path of length 2, $\boldsymbol{t}_i \rightarrow \boldsymbol{t}_i^{(1)} \rightarrow \boldsymbol{t}_i^{(1,n)}$ is fault-free and disjoint from other constructed paths. If it is fault-free and disjoint from other constructed paths, let the path be $\boldsymbol{t}_i \rightsquigarrow \boldsymbol{t}'_i$. If there is not such path, or if $\underline{t}_{in} = l_i$, we construct a path to a candidate sub burnt pancake graph for $\boldsymbol{t}_i$ as similar to the Sub Steps 2a), 2b), and 2c). Let this candidate sub burnt pancake graph be $B_{n-1}(l'_i)$.

Then, this step is divided into two cases depending on l_i and l'_i to construct the path.

Case 1)$(l_i = \underline{l}'_i)$ For pairs of the nodes such that $l_i = \underline{l}'_i$ hold, we can construct disjoint paths of lengths at most 7 that pass a candidate sub burnt pancake graph $B_{n-1}(p)$ that does not include any source nor destination node from Lemma 7. Note that if $l_i = l'_i$, in case that there is a path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}'_i$ $(\subset B_{n-1}(l'_i)$ of length 3 is constructed among the paths given in Lemma 4, it is possible to construct the path $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}'_i$ $(\subset B_{n-1}(l'_i)$ of length 2. Similar discussion holds for $\boldsymbol{t}_i$. From Step 2, the destination sub burnt pancake graph $B_{n-1}(l_i)$ is selected among the candidate sub burnt pancake graphs so that it can be reached from the node $\boldsymbol{s}_i$ or $\boldsymbol{t}_i$ with the shortest path. Therefore, if $l_i = l'_i$, the lengths of the paths $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{s}'_i$ and $\boldsymbol{t}_i \rightsquigarrow \boldsymbol{t}'_i$ are

both 2. Therefore, the sum of the paths from s_i and t_i to $B_{n-1}(l_i)$ is at most 2 + 2 + 7 = 11.

Case 2)($l_i \neq l'_i$) If $l_i \neq l'_i$, consider the paths from $B_{n-1}(l'_i)$ to $B_{n-1}(l_i)$ of lengths at most 5 given by Lemma 6. Then, there is at least one fault-free path among them. Hence, if $l_i \neq l'_i$, the sum of the paths from s_i and t_i to $B_{n-1}(l_i)$ is at most 3 + 3 + 5 = 11.

In this step, we can construct a path of length at most 11 between a pair of a source node and a destination node in $O(n^3)$ time.

Step 4) For k pairs of nodes s_i and t_i ($1 \leq i \leq k$), from Steps 1 to 3, we have constructed paths $s_i \rightsquigarrow s'_i (\in B_{n-1}(l_i))$ and $t_i \rightsquigarrow t'_i (\in B_{n-1}(l_i))$ where $B_{n-1}(l_i)$ is the target sub burnt pancake graph for s_i and t_i. $B_{n-1}(l_i)$ does not include any node on $s_j \rightsquigarrow s'_j$ or $t_j \rightsquigarrow t'_j$ ($j \neq i$), and contains at most $(n - 2k + 1)$ faulty nodes. Therefore, from Theorem 1, we can construct a path $s'_i \rightsquigarrow t'_i$ of length at most $2n + 2$ in $O(n^2)$ time.

Consequently, our algorithm can construct each path $s_i \rightsquigarrow t_i$ of length at most $2n + 13$ in $O(n^3)$ time. Therefore, it takes $O(kn^3)$ time to construct k paths.

4. Evaluation

To evaluate performance of our algorithm, we conducted a computer experiment. The algorithm constructed k disjoint fault-free paths between the k pairs of source and destination nodes in a n-burnt pancake graph with $(n - 2k + 1)$ faulty clusters whose diameters are 3. In this section, we give the method, the results, and consideration.

4.1 Method

In the experiment, we applied our algorithm to solve the k-pairwise cluster-fault-free disjoint paths problem ($3 \leq k \leq \lceil n/2 \rceil$) in a B_n ($5 \leq n \leq 40$). We repeated the following steps for 10,000 times and measured the average execution time and the maximum path length as well as the average path length.

1) We first set up $(n - 2k + 1)$ disjoint faulty clusters whose diameter is fixed to 3.
2) Then we select k source nodes s_1, s_2, ..., s_k and k destination nodes t_1, t_2, ..., t_k among non-faulty nodes.
3) We apply the algorithm to construct k disjoint fault-free paths $s_i \rightsquigarrow t_i$ ($1 \leq i \leq k$) and measure the execution time, the maximum path length, and the average path length.

4.2 Results

Figure 4 shows the results of the maximum path lengths and the average path lengths for $5 \leq n \leq 40$. In addition, Figure 5 shows the result of the average execution time for $5 \leq n \leq 40$ and $3 \leq k \leq \lceil n/2 \rceil$. From Figure 4, we can see that there is no path whose length attained the theoretical maximum path lengths. From Figure 5, the average execution time seems to converge to $O(n^{2.2})$.

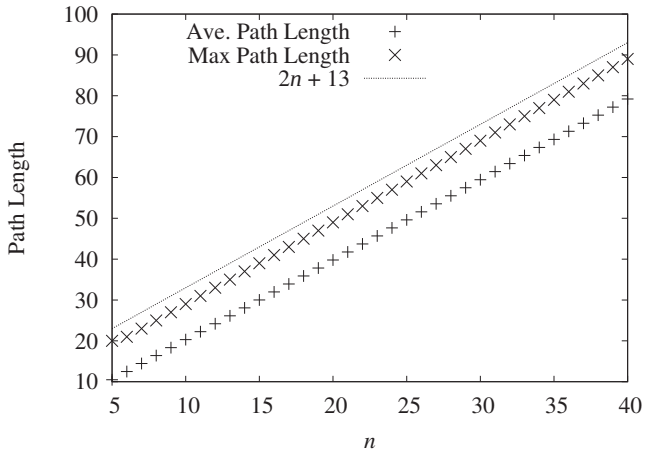


Fig. 4: Maximum and average path lengths of our algorithm

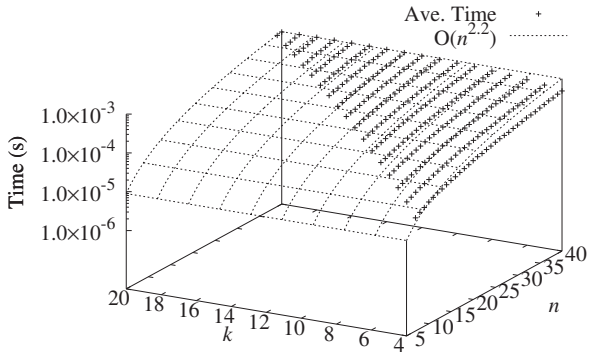


Fig. 5: Execution time of our algorithm

4.3 Limitation

The algorithm that we have proposed cannot solve the k-pairwise cluster-fault-tolerant disjoint paths problem in a burnt pancake graph with $k = 2$. Our algorithm makes use of a redundant candidate sub burnt pancake graph. From Lemma 3, if there are at most $(n - 2k + 1)$ faulty clusters whose diameters are at most 3 in B_n, there are at least $(4k - 2)$ candidate sub burnt pancake graphs, which do not include any center of the faulty clusters. If there are at least two source or destination nodes in a candidate sub burnt pancake graph and they are not the corresponding pair, the candidate sub burnt pancake graph is unavailable. Therefore, in the worst case, $2k$ candidate sub burnt pancake graphs are not available. Moreover, it is necessary to assign one distinct candidate sub burnt pancake graph to each of the source and destination pair. Therefore, at least k candidate

sub burnt pancake graphs must be required. In addition, as shown in Lemma 7, one candidate sub burnt pancake graph is used to connect paths between two B_{n-1}'s that do not have direct edges between them. Therefore, in total, $(3k+1)$ candidate sub burnt pancake graphs are necessasry. Then, from $4k-2 \geq 3k+1$, $k \geq 3$ is a necessary condition to apply our algorithm.

5. Conclusions and Future Works

In this paper, we have proposed an algorithm that solves the k-pairwise disjoint paths problem in an n-burnt pancake graph with $(n-2k+1)$ faulty clusters whose diameters are at most 3. The time complexity of the algorithm is $O(kn^3)$ and the maximum path length is $2n+13$. We have conducted a computer experiment and its results showed that there was not any path that attained the theoretical maximum path length and the average time complexity of the algorithm is $O(n^{2.2})$.

Future works include extension of the algorithm so that it can address the cluster-fault-tolerant disjoint paths problem with two pairs of nodes in B_n as well as improvement of the maximum path lengths.

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] A. Bossard and K. Kaneko, "k-pairwise disjoint paths routing in perfect hierarchical hypercubes," *Journal of Supercomputing*, vol. 67, no. 2, pp. 485–495, Feb. 2014.

[2] Q.-P. Gu and S. Peng, "An efficient algorithm for k-pairwise disjoint paths in star graphs," *Information Processing Letters*, vol. 67, no. 6, pp. 283–287, Sep. 1998.

[3] ——, "An efficient algorithm for the k-pairwise disjoint paths problem in hypercubes," *Journal of Parallel and Distributed Computing*, vol. 60, no. 6, pp. 764–774, Jun. 2000.

[4] K. Kaneko, "Internally-disjoint paths problem in bi-rotator graphs," *IEICE Transactions on Information and Systems*, vol. E88-D, no. 7, pp. 1678–1684, Jul. 2005.

[5] K. Kaneko and N. Sawada, "An algorithm for node-to-node disjoint paths problem in burnt pancake graphs," *IEICE Transactions on Information and Systems*, vol. E90-D, no. 1, pp. 306–313, Jan. 2007.

[6] K. Kaneko and Y. Suzuki, "Node-to-node disjoint paths problem in a pancake graph," in *Proceedings of the Second International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing*, Aug. 2001, pp. 572–579.

[7] D. Kocík, Y. Hirai, and K. Kaneko, "An algorithm for node-to-node disjoint paths problem in a möbius cube," in *Proceedings of the 2015 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2015, pp. 149–155.

[8] 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, Mar. 2003.

[9] A. Bossard and K. Kaneko, "Node-to-set disjoint-path routing in hierarchical cubic networks," *The Computer Journal*, vol. 55, no. 12, pp. 1440–1446, Dec. 2012.

[10] ——, "Time optimal node-to-set disjoint paths routing in hypercubes," *Journal of Information Science and Engineering*, vol. 30, no. 4, pp. 1087–1093, Jul. 2014.

[11] 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, Apr. 1997.

[12] D. Kocík, Y. Hirai, and K. Kaneko, "Node-to-set disjoint paths problem in a möbius cube," *IEICE Transactions on Information and Systems*, vol. E99-D, no. 3, p. in press, Mar. 2016.

[13] L. Lipták, E. Cheng, J.-S. Kim, and S. W. Kim, "One-to-many node-disjoint paths of hyper-star networks," *Discrete Applied Mathematics*, vol. 160, no. 13-14.

[14] Y. Xiang and I. A. Stewart, "One-to-many node-disjoint paths in (n,k)-star graphs," *Discrete Applied Mathematics*, vol. 158, no. 1, pp. 62–70, Jan. 2010.

[15] 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, Mar. 2014.

[16] 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, Jun. 2012.

[17] ——, "Set-to-set disjoint paths routing in hierarchical cubic networks," *The Computer Journal*, vol. 57, no. 2, pp. 332–337, Feb. 2014.

[18] D. S. Cohen and M. Blum, "On the problem of sorting burnt pancakes," *Discrete Applied Mathematics*, vol. 61, no. 2, pp. 105–120, 1995.

[19] T. Iwasaki and K. Kaneko, "Fault-tolerant routing in burnt pancake graphs," *Information Processing Letters*, vol. 110, no. 14-15, pp. 535–538, Jul. 2010.

[20] 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.

[21] ——, "Hamiltonian cycles and hamiltonian paths in faulty burnt pancake graphs," *IEICE Transactions on Information and Systems*, vol. E90-D, no. 4, pp. 716–721, Apr. 2007.

[22] 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, Apr. 1989.

[23] D. W. Bass and I. H. Sudborough, "Pancake problems with restricted prefix reversals and some corresponding cayley networks," *Journal of Parallel and Distributed Computing*, vol. 63, no. 3, pp. 327–336, 2003.

[24] W. H. Gates and C. H. Papadimitriou, "Bounds for sorting by prefix reversal," *Discrete Mathematics*, vol. 27, pp. 47–57, 1979.

[25] M. H. Heydari and I. H. Sudborough, "On the diameter of the pancake network," *J. Algorithms*, vol. 25, no. 1, pp. 67–94, 1997.

[26] K. Kaneko and Y. Suzuki, "Node-to-set disjoint paths problem in pancake graphs," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 9, pp. 1628–1633, Sep. 2003.

[27] K. Qiu, H. Meijer, and S. G. Akl, "Parallel routing and sorting on the pancake network," in *Proceedings of International Conference on Computing and Information*, ser. Lecture Notes in Computer Science, vol. 497. Springer Verlag, 1991, pp. 360–371.

An Efficient Fault Tolerant Scheme for Mobility Management in Wireless Networks

Abel DIATTA[1], **Ibrahima NIANG**[1], **and Mandicou BA**[2]
[1]Département de Mathématiques et Informatique, Laboratoire d'Informatique de Dakar (LID)
[2]Département Génie Informatique, Ecole Supérieure Polytechnique (ESP)
Université Cheikh Anta Diop, Dakar, Sénégal

Abstract—*Mobile communications are nowadays highly developed thanks to the multiplicity of mobile devices. Some communications pass directly between mobile nodes (because the latters have direct connections between them), while for others, the mobile node must pass through a point of attachment (PoA). In the latter case, when the PoA falls down, all mobile nodes that are attached to him losing communication with their counterparts.*

In this paper, we develope a way to avoid this loss of mobile nodes communications even when their PoA falls down. This, thanks to an algorithm that we propose to strengthen the capacity of the Media Independent Handover Function (MIHF) in terms of managing the handover, especially during a failure of a PoA. In other words, our algorithm combines management of the continuity of communication during handover and managing the fault tolerance of the PoA.

Keywords: Fault Tolerance, Mobility management, Wireless Networks, MIHF

1. Introduction

With the proliferation of mobile devices (smartphones, tablets, laptops, ...), wireless networks have become essential nowadays. Indeed, the ability of users to communicate, send files, ... while moving, do that users are genuinely interested over these networks. In most cases, the communications are via applications using P2P networks (Skype, viber, WhatsApp ...) [1].These two opportunities (possibility of movement during communication, use of P2P technology) offered to the user, are problematic. Indeed, in P2P technologies you need a good strategy for (i) fault tolerance because the nodes arrive and depart at any time. Meanwhile, in an environment characterized by high mobility of nodes, it is essential to take into account the (ii) frequent disappearances of links, but especially (iii) communications interruptions and loss of packets due to change coverage areas (handover) or disruption of an Attachment Point (PoA).

In the existing literature, work that included fault tolerance [2], [3], [4], [5] do not integrate mobility communication into their work. In other words, in these works, two users who are communicating are forced to stay on one place until the end of their communication. This is a heavy constraint. In the same vein, research works that tried to manage mobility nodes [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], did not take into account the fault tolerance.

Yu Liu et *al.* in [13] used a P2P technology to manage mobility. However, their study focused on how to avoid interference between the nodes. Abhishek Dhiman et *al.* in [14] treated the vertical and horizontal handover but they were interested in the throughput and delay (when a node moves at a given speed from one Access Point (AP) to another or from one Base Station (BS) to another) in Wi-Fi and WiMAX. They not only did not address the continuity of communication between two mobile nodes, but they have especially not integrated the possibility that a BS or an AP goes down while mobile nodes attached to it are in communication (ie fault tolerance). It is the same for papers [16], [17], [18], [19] in which the authors conduct studies on the performance of applications such as FTP, Video conference ... in WLANs, WiMAX, UMTS in terms of delay for the handover but also, in terms of traffic sent and received. In [20], autors ensure the continuity of service during handoff but not take into account the fault tolerance.

In this paper, we propose a solution to ensure the continuity of communication between two mobile nodes even when their PoA falls down i.e we integrate both mobility management and fault tolerance.

The rest of the paper is organized as follows. In section II, we give the related works on Mobility and fault tolerance in Wireless mobile Networks. In section III, we give our contribution. Section IV gives details on our solution for Mobility and Fault Tolerant Management in Wireless mobile Networks. A performance analysis of our solution is done in this section. Section V is devoted to the conclusion and our future works.

2. Related Work

2.1 Survey on Mobility in Wireless Networks

In [8], authors, about searching for files have set up a cluster system based on mobile Ad hoc networks approach. Their solution is mainly based on the creation of clusters head (CH) and secondary CH. It presents notorious limits on the managing of CH failures. Indeed in their approach, the secondary CH must wait a time t seconds, if it does not

receive messages from the CH, it considers it down and takes over. Thus, in their approach, a member of a cluster will wait $2 \times t$ seconds (t for CH and t for secondary CH) before constating that its CH is down and that it will seek another CH. In a file-sharing system, this is tolerable. By cons in the case of real-time communications that is unacceptable.

In [9], the solution established by Jabbar et *al.* allow mobile devices to connect themselves without a point of attachemnt (Wi-Fi Direct). Therefore, in their solution, mobile nodes are to be confined within a small geographical area. What is not suitable for internet or social networks.

In [10] Kim et *al.* have set up a system to ensure the handover but only in the context of data loss. Their solution avoids data loss by selecting a peer agent to store the data of the mobile node (MN) which is moving to another area. Once it will be connected to another point of attachment (PoA), peer agent transmits it the data it had guarded and the transmission continues. In their approach, when the MN enters into handover, it does not receive data until it establishes a connection with another PoA. In other words, during this time the communication is interrupted. In the case of video transmission, as in their case the solution is relevant. However, in the case of communication via Skype calls, viber ... for example, this is unacceptable.

Kuo et *al.* in [11] and Angoma et *al.* in [20] have provided solutions to ensure service continuity during the handover. However in [11] they took into account only the horizontal handover. While in [20], their deployment in a real environment has not taken into account the possibility that a PoA fails during communication or that the access point is moved in the case of Wi-Fi. It is very possible especially now with the existence of wireless routers.

In [12], authors considered the horizontal and vertical handover (HHO and VHO). However, their study was limited mainly to show the impact of the movement speed of the mobile not only in terms of packet loss but also the terms of time required for handover. They have not implemented a strategy to ensure the continuity of communication during handover.

In [21], authors use the packet retransmission system for managing fault tolerance. Indeed if after some time an ACK is not received, they retransmit the packet. However, it should be noted that their solution does not solve the problem when the cluster is down. Because we can retransmit the packet as many times as we want, it will always be the same scenario.

In [22], Zayaraz et al. based their study on the comparison in terms of signal strength and handover (HO) delay. They applied the comparison of the two types of network integration namely the loose coupling and tight coupling but also on WiFi and WiMAX networks. To manage the HO, authors set up a system which, when the link between MN and PoA is lost, the Media Independent Handover user (MIH user) initiates the discovery of a candidate network. The MN checks the RSS (Received Signal Strength) and the bandwidth of the WiMAX network. If the bandwidth is greater than a threshold defined in the MN, then MN starts the execution of the HO. The problem with their solution is that if there are several WiMAX networks that cover the area and whose bandwidth is greater than the threshold set in the MN, there will create a conflict because the MN will attempt to connect to all these networks at the same time.

2.2 Related Work on Fault Tolerance Mechanisms

In [2], [3], messages ping / pong are used to verify the breakdown of nodes. In fact, if after some time, a node does not respond with a message pong, it is declared down. We know that the response time is strongly dependent on the quality of the network. Otherwise, in their solution, a node can be declared out when it is not (simply because the response has been slow to happen due to poor network quality or a temporary disconnection of the link).

According to [4], a system in which the degree of distribution (degree of connectivity) is high, is more vulnerable to attack. But conversely, it provides a much more efficient communication and better fault tolerance. By cons, a system where the degree distribution is not strong is more resistant to attack, but less effective in terms of communication and fault tolerance. So be in the middle (ie a high degree of constant distribution). This is what Suto et *al.* in [4] wanted to manage by implementing the hub nodes with a high degree and non-hub nodes with a low degree. However, their solution does not solve the problem. Indeed their degrees depend on the total number of nodes in the network. However, in networks with the size of the internet, characterized by high Chuns, the number of nodes in the network continuously changes. By applying their solution, the system will be unstable. Sometimes it has better fault tolerance and therefore more vulnerable to attack because the degree became high, sometimes communication is lacking because the degree is again low (because of several departures of nodes).

In [6], Lun et *al.* have developed a method for detecting failures of nodes. To do this, they put up a message storage tree (ms-tree). Each node sends a message to others. Each node receiving the message saves the message source in the ms-tree; in turn sends the message to others, and then adds the source node in a NFLP list (Non-Faulty-Like Peer). They realize the transmission message during three steps and after that, each node counts the number of times each node appears in the list NFLP. If a node appears a number of times less than $n - \left\lfloor \frac{n-1}{3} \right\rfloor$, then this one is considered down (n is the number of peers in the network). A large incovenient in this system is that in highly dynamic systems (arrival and departure at any time), a node can arrive in the system at

the third step of sending the message. It is clear that in this case, it will appear in the NFLP a number of times less than $n - \left\lfloor \frac{n-1}{3} \right\rfloor$, and therefore will be declared down while it is not.

3. Contribution

We propose a model combining both the management of mobility and fault tolerance of nodes (MNs and PoAs) based on a hierarchical wireless network. Unlike flooding message used, our routing system avoids overloading the network because each PoA, by sending a message to its neighbors, precise to them in a list, all other neighbors to which it sent the same message. Therefore, the later PoA will not send the same message at the same latters even if they are its neighbors.

In addition, our solution helps strengthen the $MIHF$ (Media Independent Handover Function) protocol by adding new features including those to continue communication even when the PoA falls down. This, due to the use of priority and backup addresses.

4. An efficient Fault Tolerant Solution for Mobile Wireless Networks

4.1 Basic Idea

The idea of our solution is very simple. We started with the following conclusion:

1) All the papers that have dealt fault tolerance [2], [3], [4], [5], [23]
   - consider that a node is down, if it is physically defective or does not respond to a ping message that was sent to it,
2) All the papers that have worked on the mobility of nodes in wireless networks [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] did not take into account the possibility that a PoA falls down during one of its son communicates

Considering all this, our solution sets a time t after which if a node does not respond, the source node asks two of its neighbors if they can contact the destination. If their answer is "NO", then the later is considered broken. By cons, if at least one of the two answers with "YES" this means that it is the connection between the source and the destination which is a problem.

This verification is especially important that declare a node as failed while it is not, creating unnecessary additional operations. In fact, if a node is declared down, all the nodes under its responsibility (especially when it is a backbone) will undertake updates to their routing table when it was not necessary since the node always work.

In our solution a node can be considered as a PoA, if at least k nodes can connect to it but also if it has a fixed IP address. PoAs are linked together randomly. The mobile nodes are connected to different $PoAs$ via Wi-Fi network, WiMAX, UMTS, etc. Whenever a MN connects, its distance from the PoA is relieved. Unlike a lot of work, when this distance changes (ie when the MN moves), the HO is triggered, in our case, when the absolute value of this distance increases, the PoA by flooding sends a message to its neighbors to tell them that it has a son which wish to connect to them. This message contains the MN information.

The MN connects to the first PoA that responds and then informs its former PoA that it is connected to such PoA. Each PoA contains a list of MNs that are attached to it. In addition, each PoA contains information about its neighbors (distance, bandwidth, ...).

Drawing inspiration from [14], a MN has multiple IP addresses according to networks to which it belongs (Wi-Fi, WiMAX, UMTS,...). However, there is an address that is marked as a priority. This is the address obtained in the network to which the MN is attached. To find which network the MN is attached, we compare the absolute values of the distances between the MN and the PoA. MN will be attached to the PoA with which the absolute value is the smallest. It is this address that will be considered as a priority until the MN moves to another network. Figure 1 shows our hierarchical wireless architecture.

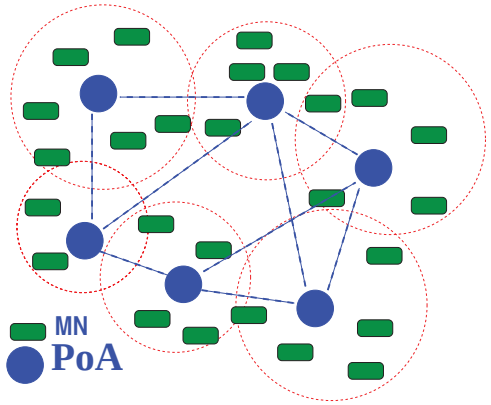


Fig. 1: Hierarchical Wireless Networks

4.2 Architecture operating

4.2.1 Routing

In our architecture, a MN wishing to contact another, sends a call message (audio, video ...) to its PoA. The message contains the information of the source and the destination. The PoA checks whether the destination is not attached to it. If attached to it, the message is sent directly to the MN, otherwise the PoA sends, by flood, the message to all its neighbors. In this message, the PoA precises the list of all neighbors to whom it sent the message. This will prevent other neighbors to send the same message to those which

have already received (to avoid overloading the network). This list will be re-initialized at each PoA. The PoA checks whether the destination is not attached to it. If attached to it, the message is sent directly to the MN, otherwise the PoA sends, by flood, the message to all its neighbors except those which have already received, so on (see Algorithm 1).

For sending the response, we use the same scenario. The destination (ie the source of the response) sends the response to the PoA. This later verifies if the MN destination of the response (ie the source of the original message) is not attached to it (because after sending the message, it can move to another PoA). If attached to it, it sends it the response, otherwise the PoA sends the response to all its neighbors, stating in the message the list of other neighbors to which it also sent the response. Each of the neighboring check in turn if the MN destination of the response is attached to it. Otherwise, it sends the response to all its neighbors except those that have already been traversed by the response. The same process continues until the MN destination.

msg	: Message
MN_{src}	: MN source
MN_{Dest}	: MN destination
$List_{PoA}(MN)$	: List of MNs attached to PoA
$idMN$	: MN Identifier
$idPoA$	: Identifier of PoA source
$idPoA_i$	: Identifier of PoA i
num_neigh	: Number of neighbors
$List_{PoA}(Neighbor)$	: List of PoA neighbors
$List^{msg}_{PoA}(Neighbor)$	: List of neighbors receiving the msg
$function : send(source, msg, destination)$	

***Algorithm 1**: Sending msg from MN_{src} to MN_{Dest}*

send(MN_{src}, msg,$idPoA$) %MN sends msg to its PoA
if ($MN_{Dest} \in List_{PoA}(MN)$) then
  send ($idPoA$, msg, $idMN_{Dest}$)
else
  foreach PoA neighbor
    send ($idPoA_i$, msg, $List_{PoA}(idNeighbor)$
  end for
% When receiving a msg, each PoA does the following
  repeat
  if ($MN_{Dest} \in List_{PoA}(MN)$) then
    send ($idPoA$, msg, $idMN_{Dest}$)
  else
    foreach PoA neighbor
      if (($idPoA_i \in List_{PoA}(idNeighbor)$) and ($idPoA_i \notin List^{msg}_{PoA}(Neighbor)$))
        send ($idPoA$, msg, $idPoA_i$)
      end if
    end for
  end if
  until (($MN_{Dest} \in List_{PoA}(MN)$) $\vee$ (all PoA receive the msg))
end if

To send the response, the same algorithm is used. We simply reverse the roles of MN_{src} and MN_{Dest}.

The worst case complexity of this algorithm is obtained on lines 12, 13 and 14. Let n be the number of PoA.
For each PoA, we must go through the list of its neighbors. Therefore the complexity is a function of $n\times$(size of each PoA neighbors list). These instructions will be in the worst case until all PoA receive the message. In other words, the complexity is $n \times n\times$(size of each PoA neighbors list). However, the size of the PoA neighbors list is a most equal to connectivity degree of that PoA.
Let k be the maximum connectivity degrees of PoA.
So complexity is $O(k \times n^2)$. We recall that n is the number of PoA and not the total number of nodes.

4.2.2 Node departure

For node departures, if it is a MN that is leaving the network, it informs its PoA and then leaves. The PoA removes it from the list of MN that are attached to it. However, in the case of it is a PoA that leaves the network (e.g. an AP), it sends information of its neighbors (relative distances, bandwidth, ...) to every MN which are attached to it, informs its neighbors and then leaves. Each MN attempts to connect to the PoA closest to him.

The complexity in the worst case is obtained when it is a PoA leaving the system. This should send a message to all its sons. So the complexity depend on the *(number of MNs attached to PoA)*. In addition, the PoA leaving the system inform all its neighbors. Therefore the complexity is also dependent on the *(number PoA neighbors)*.
Then, the complexity depends on the *(number of MNs attached to PoA) + (number of PoA neighbors)*. This is at most equal to the degree of connectivity of the PoA.
Let k be the connectivity degree of PoA.
Therefore the complexity is $O(k)$.

4.2.3 Node joining

When a MN arrives, it sends a broadcast message. In the message, it specifies that it wants to connect to a PoA. If there is a PoA which responds, it tries to connect to it. The response contains the PoA information thereof (position, bandwidth, number of remaining connection, ...). If multiple PoA respond, the MN chooses to connect to the PoA which has the largest bandwidth. If they have the same bandwidth, the MN then connects to the nearest PoA.
If it is a PoA which arrives, it sends a broadcast message stating its position, bandwidth. Each PoA receiving the message is considered as its neighbor. The latter responds to arriving PoA stating in turn its position, bandwidth, etc. It saves it in its neighbor list. The arriving PoA also recorded in its neighbor list, all the PoA that responded.

Complexity is based at most on the number of degree of connectivity (ie the number of neighbors of incoming PoA and the number of MNs attached to it). Therefore the complexity is $O(k)$.

4.2.4 Fault-tolerance

As we said above, the mobility management works consider that a node is down when it does not respond to a "ping" message during a time t. But the arrival of a message (request or response) depends on other factors such as the quality of bandwidth, link status, etc. That is why in our case, all nodes send to their neighbors (MN or PoA) ping messages at regular time interval t. When a node does not respond to a "ping" message for a time t, the source asked two of its neighbors if they can contact the destination. If their answer is "NO", then the later is considered broken. All its neighbors suppress it in their neighbor list. By cons, if at least one of the two answers is "YES" this means that it is the connection between the source and the destination which is a problem.
When a node fails, the following steps are performed:

1) If it is a MN that is down:
   - its PoA removes it from its MNs list
   - its neighbors suppress it in their neighbors list
2) If it is a PoA that is down:
   - each MN connects to the PoA of its backup address
   - each PoA which was connected to the down one removes it from its neighbor list and attempts to connect to another PoA

The complexity in the worst case is obtained when it is a PoA that fails. Each node that was connected to the latter changes its neighbor table by removing the failed PoA in the neighbor list.
Let k be the connectivity degree of PoA.
There are at most k nodes that were connected to the failed PoA. Each of these k nodes must go through the list of its neighbors ($k\times$ (size of list)). The size of this list is at most equal to k. Therefore the complexity is $O(k^2)$

4.2.5 Managing Continuity of communication during handover

Handover management includes three stages [22]:

- Initiation of handover: When the MN moves
- Available networks Discovery
- Execution of handover: Connect to an available network

In papers which worked on the management of handover, when the MN moves, the handover is triggered (ie the implementation of the Media Independent Handover Function (MIHF)). This sometimes creates unnecessary operations (thus overloading the network). Because the MN can move by approaching more its PoA, and thus its signal becomes better. It is unnecessary in this case to trigger a handover.
In our case, we trigger the handover if and only if the absolute value of the distance of the MN with its PoA increases (as this shows that the MN moves away from its PoA).

To implement our strategy for ensuring the continuity of communiction, we are inspired by [14] where authors use Master IP.
When a node integrates the network, the address that was provided to it in its PoA network, is considered as a priority. If the MN straddles several other networks, it will have other addresses it got from other PoA. Among these addresses, the one it obtained from the PoA which has the largest bandwidth will be marked as a backup address. What will serve this backup address ?

Response: When the PoA of the MN (ie the one of its priority address) is faulty, backup address is used directly to continue the communication. Hence, we avoid the interruption of communication.

In summary, in our solution, we trigger the handover in two cases:

- If the distance between the MN and its PoA grows
- When the PoA of at least one of the communicating MN fails (even if the MN does not move).

This is summarized in the following algorithm:

***Algorithm 2**:Continuity of Communication Management*

```
1: If distance (MN, PoA) grows
2:    trigger MIHF
3: Else if PoA falls down      /%MN does not move %/
4:    MN connects, meanwhile, to 2nd priority network
5:    MN checks the best network in terms of bandwidth
6:    If best network <> from 2nd priority network
7:          MN connects to the best network
8:    End if
9: End if
```

We have added a new feature in the $MIHF$ (Media Independant Handover Function), these are lines 3 to 9 of algorithm 2. Indeed, some fault tolerance of a PoA was previously not included in the $MIHF$. In other words, before our algorithm, a failure of a PoA causes the breakdown of communication.

Similarly, the complexity depends on the number of PoA neighbors to which was attached the communicating MN.
Let k be the connectivity degree of PoA.
The MN in communication is connected to the PoA. Therefore the number of other PoAs which are neighbors of the latter PoA is at most equal to $k-1$. Since this is one of those $(k-1)$ PoAs neighbors that the communicating MN seeks an alternate; the complexity is therefore $O(k-1)$.

4.3 Performances Analysis

In this section, we first give a summary table of the costs of the different algorithms we have implemented, then we make a criticism of our solution.

- FT: Fault Tolerance
- MCC^{HO}: Managing Continuity of Communication during Handover
- Let K be the set of degrees of connectivity of all PoAs
- $k = Max(i)$, $i \in K$
- n: number of PoA

Algorithm	Routing	Arrival nodes	Nodes departure	FT	MCC^{HO}
Complexity	$O(k \times n^2)$	$O(k)$	$O(k)$	$O(k^2)$	$O(k-1)$
Memory Occupation	high	low	low	medium	low

Table 1: Complexity of our algorithms

The core strength of our solution is that it is very suitable for scaling. The more the number of nodes are, the higher the degree of connectivity of the system is. Thus, for operations such as routing, for example, there are more routes to which the messages are sent. Thus, it becomes faster to find a destination.

However, the main disadvantage of our solution is that it takes up too much memory space especially in the routing. Because each PoA must consult two lists: its neighbors list and the list of these neighbors which have already received the message.

5. Conclusion and future works

In this paper, we have set up a management architecture of both mobility management, fault-tolerance and ensuring communication continuity in wireless mobiles networks.
Most solutions that use the flooding routing are major polluters of bandwidth. By cons, in our cas we have given a routing algorithm (algorithm 1) which, although it uses flooding messages, avoids overloading the network. This is due to the sending by each PoA to its neighbors, the list of other PoAs to which it sent the message. In addition, we have given another one (algorithm 2) which, if embedded on a wireless network adapter of a mobile node, will allow it to continue communication even in case of failure of the PoA.

In the case of our very close perspective, we will focus on an experimental simulation analysis first and then by deployment.

References

[1] "Skype replaces P2P supernodes with Linux boxes hosted by Microsoft (updated)," http://arstechnica.com/business/2012/05/skype-replaces-p2p-supernodes-with-linux-boxes-hosted-by-microsoft/ - March 10, 2016.

[2] C. DOBRE, "A cluster-enhanced fault tolerant peer-to-peer system," *International Journal of Innovative Computing, Information and Control*, vol. 10, no. 2, pp. 417 – 436, April 2014.

[3] T. T. Nguyen and D. El-Baz, "Fault tolerant implementation of peer-to-peer distributed iterative algorithms," in *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, Dec 2012, pp. 137–145.

[4] K. Suto, H. Nishiyama, X. Shen, and N. Kato, "Designing p2p networks tolerant to attacks and faults based on bimodal degree distribution," *Journal of Communications, SI on Security and Privacy in Communication Systems and Networks*, vol. 7, no. 8, pp. 587 – 595, Aug 2012.

[5] O. Karaca and R. Sokullu, "A cross-layer fault tolerance management module for wireless sensor networks," *Journal of Zhejiang University SCIENCE C*, vol. 13, no. 9, pp. 660–673, 2012. [Online]. Available: http://dx.doi.org/10.1631/jzus.C1200029

[6] M. L. Chiang and H. C. Hsieh, "A new approach to the fault detection problem for mobile p2p network," *INFORMATION TECHNOLOGY AND CONTROL*, vol. 41, no. 2, pp. 151 – 161, 2012.

[7] S. Ferretti, "Modeling Self-Organizing, Faulty Peer-to-Peer Systems as Complex Networks," ser. Technical Report UBLCS-2010-03.

[8] T. B. Noor, M. R. Salehin, and S. R. Islam, "A clustering scheme for peer-to-peer file searching in mobile ad hoc networks," *International Journal of Advanced Research in Computer and Communication Engineering*, October 2012.

[9] W. A. Jabbar, M. Ismail, and R. Nordin, "Framework for Enhancing P2P Communication Protocol on Mobile Platform," *The International Conference on Informatics and Applications (ICIA2012)*, 2012. [Online]. Available: http://sdiwc.net/digital-library/framework-for-enhancing-p2p-communication-protocol-on-mobileplatform

[10] E. Kim, S. Kim, and C. Lee, "Supporting Seamless Mobility for P2P Live Streaming," *The Scientific World Journal*, vol. 2014, p. 8.

[11] J.-L. Kuo, C.-H. Shih, and Y.-C. Chen, "A cross-layer design for p2p live streaming with graceful handover in mobile ip network," in *ITS Telecommunications (ITST), 2013 13th International Conference on*, Nov 2013, pp. 456–461.

[12] B. S. and R. Daruwala, "Experimental analysis of horizontal and vertical handovers in wireless access networks using ns2," in *Information and Communication Technologies (WICT), 2011 World Congress on*, Dec 2011, pp. 594–599.

[13] Y. Liu, B. Guo, C. Zhou, and Y. Cheng, "Network-coded cooperative information recovery in cellular/802.11 mobile networks," *Journal of Network and Computer Applications*, vol. 51, pp. 59 – 67, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804514000320

[14] A. Dhiman and K. S. Sandha, "Vertical and horizontal handover in heterogeneous wireless networks using opnet," *International Journal of Engineering Research and Technology (IJERT)*, vol. 2, no. 6, pp. 842 – 846, June 2013.

[15] D. R. Dandekar and P. Deshmukh, "Relay node placement for multi-path connectivity in heterogeneous wireless sensor networks," *Procedia Technology*, vol. 4, pp. 732 – 736, 2012, 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212017312003982

[16] P. Babel, M. A. K. Bhola, and D. C. K. Jha, "Delay and Throughput Comparison between Hard Handover and Soft Handover by Varying the Speed in Mobile WIMAX," *International Journal of Research*, vol. 2, no. 4, pp. 742 – 746, April 2015.

[17] P. Mehta and S. Baghla, "Performance Evaluation of Heterogeneous Networks for Various Applications Using OPNET Modeller," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 3, no. 8, pp. 4003 – 4006, August 2014.

[18] M. V. Verma and S. Baghla, "Performance Evaluation of QoS in WLAN-UMTS Network Using OPNET Modeller," *International Journal of Science and Research (IJSR)*, vol. 3, no. 6, pp. 4003 – 4006, June 2015.

[19] D. J. Kadhim and S. S. Abed, "PERFORMANCE AND HANDOFF EVALUATION OF HETEROGENEOUS WIRELESS NETWORKS (HWNS) USING OPNET SIMULATOR," *International Journal of Electronics and Communication Engineering and Technology (IJECET)*, vol. 4, no. 2, pp. 477 – 496, March - April 2013.

[20] B. Angoma, M. Erradi, Y. Benkaouz, A. Berqia, and M. C. Akalay,

"A Vertical Handoff Implementation in a Real Testbed," *Mobile Computing*, vol. 1, no. 1, pp. 1 – 14, November 2012.

[21] B. Heep, M. Florian, J. Volz, and I. Baumgart, "Overdrive: An overlay-based geocast service for smart traffic applications."

[22] G. Zayaraz, J. K. Devi, V. Vijayalakshmi, and V. Hemamalini, "MOBILITY MANAGEMENT IN HETEROGENEOUS WIRELESS NETWORKS," *IJRET: International Journal of Research in Engineering and Technology*, vol. 03, no. 07, pp. 761 – 768, May 2014.

[23] I. Diane, I. Niang, and B. Gueye, "A hierarchical dht for fault tolerant management in p2p-sip networks," in *Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, ser. ICPADS '10, 2010, pp. 788–793.

A Trustworthy Information Publication and Search System for Large-Scale & Mobile Wireless Networks

Yung-Ting Chuang*, Qian-Wei Wu
Department of Information Management, National Chung Cheng University, Chia-Yi County, Taiwan

Abstract - *As ubiquitous networked devices continue to play an increased role in the daily lives of most people, people can use any mobile device to access or share any information at anytime and from anywhere. However, most of the search services still belong to centralized systems. In addition, it is very difficult to distribute or retrieve data when considering enormous number of mobile nodes over the large-scale mobile wireless networks. In order to address these problems, we present a Trustworthy Information Publication and Search System, an efficient decentralized search and retrieval system for the large-scale and mobile wireless networks. Our goals are to: 1) ensure robustness and effectiveness of the system which cannot easily to be censored or filtered; 2) provide high search and retrieval rate even when in a high mobility and density network; 3) require reasonable message costs and delay.*

Keywords: P2P, Membership, decentralized mobile search and retrieval, distributed systems

1 Introduction

Currently, our trust in the accessibility of information over the Internet depends on benign and unbiased administration of centralized search engines and indexes. Unfortunately, the experience of history, shows that we cannot depend on such administrators to remain benign and unbiased forever. To ensure the free flow of information over the Internet, some decentralized search and retrieval systems [3,6,10] have previously proposed, so that it would be difficult to censor or filter information accessed over the Internet. These decentralized search systems give better assurance to the users of the Internet, because a small number of administrators cannot prevent them from exchanging their information with others.

The network have slowly shifted to a large-scale wireless network (e.g., wireless sensor network (WSN), mobile ad-hoc network (MANET), etc.), where these networks contain massive number of sensors gathered together to sense, and communicate their environmental data with others. However, one of the biggest problems behind such wireless sensor and mobile ad-hoc networks is that it is difficult to distribute or retrieve information in such decentralized and large-scale mobile wireless networks. [9,15,11,2] have therefore proposed solutions to address the above problems in both WSNs and MANET environment. Unfortunately, these systems generate too much overhead for data distribution and database maintenance, and do not guarantee high retrieval rate when it is in a highly mobile wireless networks.

*Corresponding author: ytchuang@mis.ccu.edu.tw

2 Related Work

2.1 Peer-to-Peer Network

Mischke and Stiller [13], provide comparisons of distributed search methods for peer-to-peer networks. The structured approach [1,7] requires the nodes to be organized in an overlay network based on distributed hash tables (DHTs), trees, rings, which is efficient but is vulnerable to manipulation by untrustworthy administrators. The unstructured approach [3,6,10], is typically based on gossiping, uses randomization, and requires the nodes to find each other by exchanging messages over existing links. Our system uses the unstructured approach, which is less vulnerable to manipulation.

2.2 Online Social Networks

Currently a number of recent studies, such as [5], addressed the privacy concerns in current online social networks. Similarly, some other studies have proposed to provide better privacy and trustworthiness. Diaspora allows users to choose where to store their data with a number of different providers without a centralized control. Similarly, other systems [4,12], allow users to choose whether to store data on their or friends' devices. Thus, users can have better control over their information, and also can use the system locally without the Internet access. Other commercial applications, such as Tribler, Wuala, and 2Peer, were presented to allow users to connect and share information with their friends in a decentralized manner. Similarly, our system provides better trustworthiness by having users to share and control their data in a decentralized way.

2.3 Data Search Systems

Geographic routing based data search systems, such as [15], are proposed which aim for high scalability. In these works, a file is mapped to a geographic location using the distributed hash table (DHT) data mapping policy, and applies geographic routing methods [8] to store the file to a node closes to this geographic location. In order to retrieve a file, the requesting node first calculates the mapped

geographic location, and then applies the geographic routing methods to deliver the query to the mapped location. However, the file might have to constantly transfer to the new location when it is in a highly mobile network, thus creating a lot of overhead, delaying the data mapping, and creating query failure. In our system, we allow nodes to publish their metadata to a mapped geographic location, do not require file to constantly transfer to the new location when it is in a highly mobile network, guarantee high retrieval rate, and ensure low overhead when in a large-scale and mobile wireless networks.

3 Methodology

Therefore, in this paper, we present a Trustworthy Information Publication and Search System for large-scale and mobile wireless networks. The goal of our system is to provide robust and effective search and retrieval in a large-scale mobile wireless networks. Our system first divides the entire network into a number of regions. Next, our system have source nodes to generate metadata, where the metadata that includes a list of keywords and the URL of a file. After that, our system applies Locality Sensitive Hash (LSH) functions [14], which maps metadata to a geographical region, and then stores it in $2\sqrt{n}$ nodes in that region. Therefore, our system could reduce data replicas in a region, but still can suffice to achieve high retrieval rate and ensure mobility resilience. Similarly, the requesting node first generates its requests, applies LSH functions to map its requests to a geographical region, and then distributes its requests to $2\sqrt{n}$ nodes in the mapped region. The nodes receive such requests would compare this request with the metadata that they currently hold, and if there is a match, they would return the URL of the source node back to the requesting node. After receiving the match results, the requesting node can further retrieve the file. Having successfully retrieved the file, the requesting node becomes one of the sources nodes, and in this way, it would further publish the metadata of the file to its mapped region. In addition, our system applies extensive back-tracking method [9] to address the issues when either source node or requesting node moves to other region.

4 Validation and Future Work

So far we only conducted the literature survey and developed the Mobile and Search Algorithm. Next, in order to demonstrate the effectiveness of the proposed algorithm, we plan to implement our model, and then simulate our proposed algorithm in a highly mobile and large wireless networks. After that, we plan to evaluate the performance metrics and compare our system against other decentralized search and retrieval systems, and demonstrate that our system could achieve better scalability, overhead, and mobility resilience.

Acknowledgment

This research is supported by Ministry of Science & Technology (MOST 104-2410-H-194-090-MY2).

5 References

[1] S. Bianchi, P. Felber, and M. Gradinariu. Content-based publish/subscribe using distributed r-trees. In Proceedings of Euro-Par, pages 537–548, Rennes, France, August 2007.

[2] P. Sharma, Daniel Souza, Evan Fiore, Jeffrey Gottschalk, and D Marquis. A case for manet-aware content centric networking of smartphones. In World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a, pages 1–6. IEEE, 2012.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, pages 46–66, Berkeley, CA, July 2001.

[4] L. A. Cutillo, R. Molva, and M. Onen. Safebook: A distributed privacy preserving online social network. In Proceedings of the IEEE World of Wireless, Mobile and Multimedia Networks Conference, Lucca, Italy, June 2011.

[5] N. B Ellison et al. Social network sites: Definition, history, and scholarship. Journal of Computer-Mediated Communication, 13(1):210–230, 2007.

[6] R. A. Ferreira, M. K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with probabilistic guarantees in unstructured peer-to-peer networks. In Proceedings of 5th IEEE International Conference on Peer-to-Peer Computing, pages 165–172, Konstanz, Germany, August 2005.

[7] A. Gupta, O. Sahin, D. Agrawal, and A. El Abbadi. Meghdoot: Content-based publish/subscribe over P2P networks. In Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, pages 254–273, Toronto, Canada, October 2004.

[8] H. Frey and Ivan Stojmenovic. On delivery guarantees and worst-case forwarding bounds of elementary face routing components in ad hoc and sensor networks. Computers, IEEE Transactions on, 59(9):1224–1238, 2010.

[9] H. Y. Shen, Ze Li, and Kang Chen. A scalable and mobility-resilient data search system for large-scale mobile wireless networks. Parallel and Distributed Systems, IEEE Transactions on, 25(5):1124–1134, 2014.

[10] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy preserving P2P data sharing with OneSwarm. In Proceedings of the ACM SIGCOMM Conference, pages 111–122, New Delhi, India, September 2010.

[11] N. Shah and Depei Qian. An efficient unstructured p2p overlay over manet using underlying proactive routing. In Mobile Ad-hoc and Sensor Networks (MSN), 2011 Seventh International Conference on, pages 248–255. IEEE, 2011.

[12] A. Loupasakis, N. Ntarmos, P. Triantafillou, and D. Makreshanski. eXO: Decentralized autonomous scalable social networking. In CIDR, pages 85–95, 2011.

[13] J. Mischke and B. Stiller. A methodology for the design of distributed search in P2P middleware. IEEE Network, 18(1):30–37, 2004.

[14] A. Rajaraman and Jeffrey D Ullman. Mining of massive datasets, volume 77.

[15] S. Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin, and Fang Yu. Data-centric storage in sensornets with ght, a geographic hash table. Mobile networks and applications, 8(4):427–442, 2003.

A New Group Membership Protocol in Synchronous Distributed Systems

Sung-Hoon Park[1] and Yeong-Mok Kim[2]
[1,2]Department of Computer Engineering, Chungbuk National Unvi., Cheongju, ChungBuk, Korea

Abstract - *In distributed systems, a group of computer should continue to do cooperation in order to finish some jobs. In such a system, a group membership protocol is especially practical and important elements to provide processes in a group with a consistent common knowledge about the membership of the group. Whenever a membership change occurs, processes should agree on which of them should do to accomplish an unfinished job or begins a new job. The problem of knowing a stable membership view is very same with the agreeing common predicate in a distributed system such as the consensus problem. Based on the termination detection protocol that is traditional one in asynchronous distributed systems, we present the new group membership protocol in arbitrary wired networks.*

Key-words: Synchronous Distributed Systems; Group membership; Fault Tolerance; Wired Arbitrary Network Environment

1. Introduction

In distributed systems, a group of computer should continue to do cooperation in order to finish some jobs. A group membership protocol is especially helpful tools to allocate processes in a same group with a same view of the membership of the group. Whenever a membership change occurs, processes can consent to which of them should do to finish a waiting job or begin a new job. The problem of getting a stable membership view is very same with the one of getting common knowledge in a synchronous distributed system such as the consensus problem [1].

The Group membership protocol [2] is that every process connected in a network requires getting a stable same group membership view if all connected process are belong to just one group. The problem was widely discussed at the study community. The reason for this great study is that many distributed systems need a group membership protocol [3,4,5,6,7]. In spite of such practically usefulness, to our knowledge there is only a few research that have been committed to this problem in a wired arbitrary connected computing environment.

Depending on process failure and recover, network topologies is changed and process may dynamically connect and disconnect over a wired network. In such wired networks, group membership can be changed so much, making it a special critical module of system software part. In wired arbitrary network systems, a lot of environmental adversities are more common than the static wired network systems such as that can cause loss of messages or data [8]. In particular, a process can easily get to fault by hardware or software problem and disconnect from the wired network. Implementing fault-tolerant distributed applications in such an environment is a complex and difficult behavior [9,10].

In this paper, we propose a new protocol to the group membership protocol in a specific wired distributed computing system. Based on the termination detection protocol that is traditional one in asynchronous distributed systems, we address the new group membership protocol. We make up of the rest of this paper as follows. In Section 2 we address the system model we use. In Section 3, we describe a specification to the group membership problem in a traditional synchronous distributed system. We also address a new protocol to solve the group membership problem in a wired arbitrary computing system in Section 4. In Section 5, we address conclude.

2. Computing System Model, Definition and Assumptions

In this section, we describe our models for capturing behavior of distributed systems. We use these models foe reasoning about correctness of our protocol as well as for analysis of distributed computations. Our model for distributed systems is based on message passing, and all of protocol is around that concept. Many of these kinds of protocol have analogs in the shared memory computing system but will not be addressed in this paper.

First, we define our system model based on some assumptions and after that we address our goals. We model a distributed system as a loosely coupled messing-passing system without shared memory and a global clock. Our distributed computation model for a wired network is made up of as an undirected graph. That is, the undirected graph is described as $G = (V, E)$, in which vertices V facing each other with set of process $\{1, 2,\ldots,n\}$ ($n > 1$) with unique identifiers and edges E between a pair of process correspond the fact that the two process are in each other's transmission radii. Hence, our distributed system has a channel to directly communicate with each other which changes over time when processes move.

Every process i has a variable N_i, which denotes the neighboring processes, with that i can *directly* communicate the neighboring processes. Every process communicates with a channel that is

bidirectional; $j \in N_i$ iff $i \in N_j$. More accurately, in the network $G = (V, E)$, we decide E such that for all $i \in V$, $(i, j) \in E$ if and only if $i \in N_j$. Depending on process's movement, the graph could be disconnected that means that the network is partitioned. Because the processes may alternate their position, N_i position would be unexpectedly changed and therefore G also may be changed accordingly. The assumptions about the processes, wired network and system architecture are followings.

Every process is distinguished by a unique identifier. The unique identifiers are used to distinguish processes during operating the group membership search process. Channels and links are bidirectional that means first n first out, i.e. every process receives messages based on the sequence that are delivered over a link between two neighboring processes. Many topology changes may be arbitrary occurred when the process stops or recovers in wired networks. That makes a lot of network partitioning and merging. Processes can make a fault to be crash arbitrarily at random and can recover again at any time.

Without network partition, the sender and the receiver do successful message delivery that means the message would be successfully delivered only when the two processes remain connected for the all period of message transfer. Every process has a big receiving buffer enough to avoid buffer overflow all the time in its lifetime. Even though a finite number of topology changes, every process *i eventually* has a same view of group membership of the group to which *i* belongs.

3. Group Membership Specification

We assume that our specification is as followings, it is consist of four properties for a group membership protocol.

Safety(1) : At any time, all processes in the group have a stable consistent view.

Progress : If there are no more changes in the each views of the processes in one group, they eventually getting to their stable consistent views.

Validity : If all processes in a event know a view as their local view and they have eventually reached their stable states, then the last process of their sequences of global views are all at same position and must be equal to each other.

Safety(2) : When a view is committed as a global view, it cannot be changed.

The first property describes agreement. Consistent history must be an unchanged one for any program that satisfies the specification. The second property shows termination of global view. When the state and event of all processes are unchanged, the processes are eventually getting to close changing their output results. The third property removes trivial solutions where protocols never getting on any new view or always determine on the consistent view.

4. Group Membership Algorithm in Wired 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.

4.1 A Group Membership in a Wired Networks

We first address our group membership protocol in the wired network settings. In which we assume that process and channels have no faults.

The protocol is made up of three phases running at the process that starts the group membership protocol.

1) The first phase that is a diffusing phase and it works by first diffusing the "who" messages.

2) The second phase that is a searching phase and it runs by then accumulating the id of every process that is consist of the wired networks. We represent this computation starting processes as the *start process*.

3) The third phase is a closing phase that is managed by deciding the same view and announcing it as a stable new view to all process.

The start process will have the information enough to decide a uniform group membership view after taking all process' ids completely and the start process will then broadcast it to the rest of the process in the network. The three kinds of message, *Who*, *Ack* and *View* are used to manipulate the operations.

As the first phase is diffusing computing phase, *Who* message is used to make a start of the group membership protocol by diffusing the *Who message*.

1) The first Phase: When group membership protocol is launched at a start process *s*, the start process makes a replying queue *wl* and a accepted queue *rl* and starts a *scattering computation* by forwarding a *Who message* to all of its immediate neighboring processes. At the starting point, the replying queue makes up of only its most close neighboring process's ids and the accepted queue has nothing.

When process *i* receives a *Who message* from the neighboring process for the first time, it immediately sends the *Ack* message to the start process and propagates the *Who message* to all its neighboring process except the process from which it first accepted an *Who message*.

The *Ack* message sent by process *i* to the start process contains the ids of all its neighboring process that are needed for the start process to decide the stable view of the process connected with a distributed network. After that, any *Who message* accepted by other neighboring process will be ignored.

2) The Second Phase: Searching phase. When the start process receives the *Ack* message was taken out from the process *j*, it takes *j* out from the replying

queue and gets *j* into the accepted queue and as soon as possible it detects sequentially the each process's id included in the *Ack* message. If there is the some process in the *Ack* message which has already been accepted, i.e. that means it is in the accepted queue, it is dismissed. If it is not in the accepted queue, it is inserted into the replying queue of start process. The start process will be suspends for the *Ack* message from one.

The replying queue is increasing and decreasing repeatedly when it was accepted based on the accepted *Ack* messages, however the replying queue is continually increasing by accepting the *Ack* messages. But the replying queue at the end could have no element and the replying queue could insert all ids of processes connected to the wired networks whenever the start process accepted the *Ack* messages from all other processes. Therefore the start process eventually has much information enough to decide the stable view of the group based on the replying queue. That is because the replying queue could be eventually unoccupied and it means that the start process has accepted the *Ack* messages from all the process.

3) The Third Phase: Once the start process has accepted *Ack*s from all other process, it decides the stable view based on the replying queue and forwards a *View* message to all other process to let know the current view of the group. We show some sample running protocol as the protocol execution to explain more specific features. We address the protocol in synchronous setting even though all the behaviors of the protocol are practically asynchronous. We assume that the network shown in Figure 1(a) is asynchronous. In this shape, and for the all of the paper, thin arrows denote the route of *Who message'*s move and dotted arrows denotes the way of route of *Ack* messages to the start process.

As shown in Figure 1, process A is a start process that starts wl_a and rl_b with {B,C} and {A} at each and starts a scattering computation with forwarding out *Who message*s (indicated as "E" in the shape) to its immediate neighbors, viz. process B and C, shown in Figure 1(a). As indicated in Figure 1(b), process *B* and *C* in turn forward the *Who message* to its most close neighbors only except the start process. It sends the *Ack* message with close neighboring process queue to the start process *A*. Hence *B* and *C* also send *Who message*s to each other.

But B and C do not acknowledge to the start process about the *Who message*s because process *B* and *C* have already accepted *Who message*s from the start process at each. The information of neighboring process is piggybacked upon the *Ack* message sent by all process. Upon hearing *Ack* messages from B and C, process A renews wl_a = { B,C }, rl_b = { A } with the close neighboring process information piggybacked at the *Ack* messages. The *Who message*s is transmitted over the arrows at the edges and the dotted arrows going parallel with the edges denotes *Ack* messages. In Figure 1(c), the process D and F also send the *Ack* messages to the starts process at the time they accepted the *Who message* s from the B and C one by one.

Each of these *Ack* messages includes the ids of the neighbor. All the time, the start *A* accepts all acknowledgments from all of other process except itself in Figure 1(d) and then determines the stable view between the group and forwards it, that is the *View* message displayed in Figure 1(d).

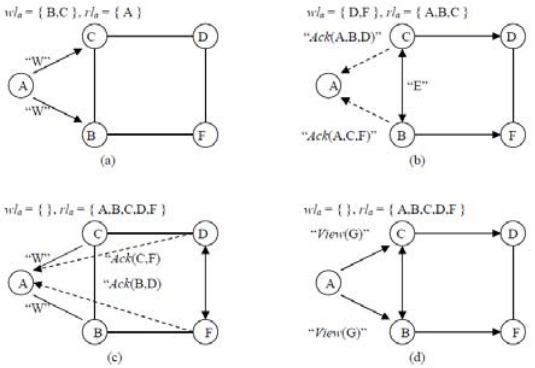


Figure 1: An example of group membership protocol execution on the process search protocol.

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.

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.

Efficient Load Balancing Algorithm for the Arrangement-Star Network

Ahmad M. Awwad[1], Jehad Al-Sadi[1]

[1]CS Dept., University of Petra, Amman, Jordan
[2]CS Dept., Arab Open University, Amman, Jordan- E-mail

Abstract — *the Arrangement-Star is a known network in literature and it is one of the promising interconnection networks for future super computers, it is expected to be one of the attractive alternatives in the future for High Speed Parallel Computers. The Arrangement-Star network having a smaller diameter, node degree, and number of links, it has a lower broadcasting cost and more flexibility in choosing the desired network size. In spite that some of the research work has been done on Arrangement-Star promising network, it still needs more time and efforts to be done on the issue of load balancing. In this paper we attempt to fill this gap by proposing an efficient algorithm for load balancing among different processors of the Arrangement-Star network. The proposed algorithm is named as Arrangement Star Clustered Dimension Exchange Method ASCDEM presented and implemented on the Arrangement-Star network. The algorithm is based on the Clustered Dimension Exchange Method (CDEM). The ASCDEM algorithm is shown to be efficient in redistributing the load balancing among all different processors of the network as evenly as possible.*

Keywords: Interconnection Networks, Arrangement Network, Star Network. Arrangement-Star, Load balancing.

1. INTRODUCTION

The arrangement-star network as a case of study on vertex product networks [1, 7, 8], it is constructed from the cross product of the star and arrangement graphs. It has shown to have superior topological properties over its constituents: the star and arrangement graphs [3, 2, 24]. Besides having a smaller diameter, node degree, and number of links, it has a lower broadcasting cost and more flexibility in choosing the desired network size.

Although some algorithms proposed for the arrangement-star graph such as distributed fault-tolerant routing algorithm [2]. But still one of the important problems that the arrangement-star network still needs more efforts and researchers time is the issue of load balancing among different processors of this network. Since there is no enough research work in literature for proposing efficient algorithms for load balancing on arrangement-star network. In this research efforts we move one more step in filling this gap by investigating and proposing the ASCDEM algorithm on the arrangement-star network, the proposed algorithm is based on the CDEM algorithm which was able to redistribute the load balance among all node of the networks on OTIS-Hypercube network as evenly as possible [17]. A reasonable and efficient implementation of the ASCDEM algorithm on our network will make the arrangement-star network more attractive for the issue of load balancing problem.

This paper is organized as follows: In the next section we present the necessary basic notations and definitions, in section III introduces the related work on load balancing, section IV presents the implementation of the ASCDEM algorithm on the arrangement-star network, finally section V concludes this research work.

2. DEFINITIONS AND Basic Topological Properties

During the last two decades a big number of interconnection networks for High Speed Parallel Computers (HSPC) investigated and proposed in literature [3, 4, 5]. As an example one of these networks was the hypercube interconnection network [6, 17]. Also a well know example is the star graph [3]. Some properties of this network have been studied in the literature including its basic topological properties, parallel path classification, node connectivity and embedding [10, 11, 13, 14]. The authors Akers and Krishnamurthy have proved that the star graph has several advantages over the hypercube network including a lower degree for a fixed network size of the comparable network sizes, a smaller diameter, and smaller average diameter. Furthermore they showed that the star graph is maximally fault tolerant edge, and vertex symmetric [3].

The star graph, however, has few drawbacks [23]. One of the major problems of the star graph is related to its scalability. The size of the star graph increases according to a factorial function, and thus grows widely very rapidly; for example, the value of 7! is equal to 5040 while the value of 11! is about forty million. Despite its attractive topological properties, the star graph has not been used in practical systems yet.

In an attempt to address the scalability problem in the star graph, Day and Tripathi [24] have proposed the arrangement graph as a generalization of the star graph. The arrangement graph is a family of undirected graphs that contains the star graph family. It slightly brings a solution to the problem of

[1] awwad@uop.edu.jo
[2] j_alsadi@aou.edu.jo

the scalability, which the star graph suffers from (i.e. the problem of growth of the number n! of nodes in the n-star). It also preserves all the nice qualities of the star graph topology including, hierarchical structure, vertex and edge symmetric, simple shortest path routing and many fault tolerance properties [24]. Still a common drawback of the star and arrangement graphs is the restriction on the number of nodes: n! for the star graph and m!/(m-k)! for the arrangement graph. The set of values of n! (or m!/(m-k)!) is spread widely over the set of integers; so, one will be faced with the choice of too few or too many available nodes.

However, there has been relatively a limited research efforts have been dedicated to design efficient algorithms for the arrangement-star graph including broadcasting [13], selection and sorting [14, 22], Fast Fourier Transform [12], and Matrix Multiplications [15] and load balancing. In an attempt to overcome the load balancing problem we present an efficient algorithm for load balancing problem on arrangement-star graph to redistribute the load balancing among all processors of the network as evenly as possible.

An arrangement graph is specified by two parameters m and k, satisfying $1 \le k \le m$. For simplicity let $\langle m \rangle = \{1,2,\ldots,m\}$ and $\langle k \rangle = \{1,2,\ldots,k\}$.

***Definition* 1:** The (m,k)-arrangement graph $A_{m,k} = (V_1, E_1)$, $1 \le k \le m-1$ is defined as follows [24]:

$V_1 = \{p_1 p_2 \ldots p_k \mid p_i \in \langle m \rangle$ and $pi \ne p_j$ for $i \ne j\} = P_k^m$, and

$E_1 = (p,q) \mid p$ and q in V_1 and for some i in $\langle k \rangle$, $p_i \ne q_i$ and $p_j = q_j$ for $j \ne i\}$.

That is, the nodes of $A_{m,k}$ labelled with a unique arrangements of k elements out of m symbols $\langle m \rangle$, and the edges of $A_{m,k}$ connect arrangements which differ in exactly one of their k positions An edge of $A_{m,k}$ connecting two arrangements which differ only in position i called an i-edge. In this case, p and q are i-adjacent and q is called (i, q_i)-neighbour of p. The (m,k)-arrangement graph $A_{m,k}$ is regular of degree $k(m-k)$ and of size $m!/(m-k)!$, and diameter $\lfloor 3k/2 \rfloor$. The $(m, m\text{-}1)$-arrangement graph $A_{m,m-1}$ is isomorphic to n-star graph S_n [8, 24], and the $(m,1)$-arrangement graph is isomorphic to the complete graph with m nodes [24].

Definition 2: The n-star graph, denoted by S_n, has $n!$ nodes each labelled with a unique permutation on $\langle n \rangle = \{1,\ldots,n\}$. Any two nodes are connected if, and only if, their corresponding permutations differ exactly in the first and one other position.

The diameter, δ, and the degree, α, of the star graph are as follows [3]:

δ, of n-star graph $= \lfloor 1.5\ (n\text{-}1) \rfloor$

α, of the n-star graph $= n-1$, where $n>1$.

***Definition* 3:** The arrangement-star graph is the cross product of the n-star graph and the (m, k)-arrangement graph, and is given by $AS_{n,m,k} = A_{m,k} \otimes S_n$ such that $n>1$ and $1 \le k \le m$.

Note that if G_1 and G_2 are two undirected graphs then for any node $X = \langle x_1, x_2 \rangle$ in the cross product graph, $G = G_1 \otimes G_2$, has an address consisting of two parts, one coming from G_1 and the other coming from G_2. We will denote the earlier part by $lp(X)=x_1$ and the later part by $rp(X)=x_2$.

Figure 1 shows the topology of $AS_{2,3,2}$ that is obtained from the graph product of S_2 and $A_{3,2}$ networks. A node $X = \langle u, v \rangle$ in $AS_{2,3,2}$ consisting of two parts, left part coming from the star graph and the right part coming from the arrangement graph (*lp and rp*). Two nodes $X = \langle u, v \rangle$ and $Y = \langle u', v' \rangle$ are connected if, $lp(X) = lp(Y)$ and $rp(X)$ is connected $rp(Y)$ in $A_{m,k}$ (in this case X and Y are said arrangement-connected) or $rp(X) = rp(Y)$ and $lp(X)$ is connected $lp(Y)$ in S_n (in this case X and Y are said star-connected). For instance in Figure 1 the node *ab*13 is connected to the node *ab*12, and the node *ab*23 is connected to the node *ba*23.

3. Background and Related Work

Many attractive properties for the arrangement-star graph have been shown in the literature enabled it to be one of the candidate's networks for the High Speed Parallel Computers (HSPC) and a reasonable choice for any real life applications [2]. This outcome about arrangement-star network has motivated us to spend more time and do some research on it for some important class of algorithms such as: the load balancing because still this networks suffers from shortening in number of algorithms for the load balancing problem in general and for load balancing problem in specific. This algorithm has been studied and proposed for many HSPC infrastructure ranging from electronic networks [2] and also for Optoelectronic networks [20, 21].

The Load balancing algorithm is a famous type of problems that is needed by all HSPC infrastructures. The load balancing problem have been investigated from many angles and point views. As an example on the literature work this problem was investigated by the researchers Ranka, Won, and Sahni [6, 16, 17], As conclusion of their work they come out with an efficient algorithm to be implemented on HSPC called the Dimension Exchange Method (DEM) on the hypercube topology. This algorithm (DEM) constructed and developed by issuing and getting the average load of neighbors' nodes, where the symmetric degree of the hypercube is n, All adjacent nodes which are connected on the n^{th} dimension they will exchange their task loads to redistribute the task load and as evenly as possible, the processor with extra load will share any extra amount of the load to its adjacent neighbor node. The DEM algorithm main advantage that it was able to redistribute the load balances of processors among all neighbors as evenly as possible. Furthermore Ranka and *et al* have enhance the load balance in the DEM algorithm in its worst case to achieve log2n on the cube network [18].

Zaho, Xiao, and Qin have investigated and proposed hybrid structure of diffusion and dimension exchange called DED-X which worked in a perfect manner for the load balancing algorithm on Optoelectronic networks [19]. The

DED-X problem main task was to redistribute the load balancing between different nodes of the network to three different phases. The achieved outcome on Optical Transpose Interconnection System networks proved that the redistribution of load balance between all nodes of the topology was efficient and mostly even. Furthermore the reached outcome and the issued results of the simulation from Zaho et al of the proposed algorithms on load balancing has shown a considerably big improvements in enhancement in redistribution the load balancing of the processors of the topology [19]. In a different literature and research done by Zaho and Xiao they investigated a different algorithm named t DED-X for load balancing on homogeneous optoelectronic technology and they proposed new algorithm framework, Generalized Diffusion-Exchange- Diffusion Method, this framework was efficient for the load balancing distribution on the Heterogeneous optoelectronic technology [6, 18].

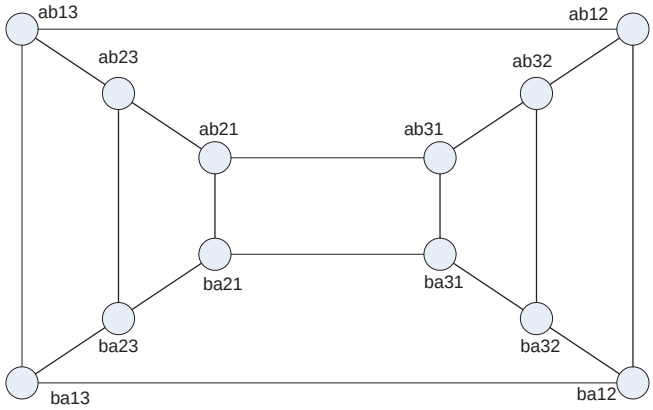


Fig.1: Arrangement-star graph, $AS_{2,3,2}$.

On the other hand Zaho, Xiao, and Qin have investigated and proved that the efficiency of the new investigated load balancing algorithms to be more effective than the X old load balancing algorithm [19].

The target of this research effort is to investigate a new algorithm for the load balancing among the nodes of the arrangement-star networks named Arrangement Star Clustered Dimension Exchange Method (ASCDEM). The algorithm is based on the Clustered Dimension Exchange Method (CDEM) [6].

4. the implementation of the ASCDEM algorithm on the arrangement-star NETWORK

The algorithm we present in this paper ASCDEM is based on the Clustered Dimension Exchange Method CDEM for load balancing for the Arrangement-Star Interconnection networks [17].

The main achievement of the new presented ASCDEM is to obtain even load balancing for the $AS_{n,m,k}$ network by redistributing the load size to reach an equal load size at each node within the whole network. The structure of the $AS_{n,m,k}$ network consists of S_n network as a first level structure of the hierarchal $AS_{n,m,k}$ network, the first level of S_n consists of $n!$ Sub-graphs, each sub graph represented by an $A_{m,k}$ Arrangement graph. The links and edges between the nodes of the whole graph have been identified and described in the above section.

The ASCDEM load balancing algorithm is based on the following two phases:

- Phase 1: Distributing the load balancing among all sub-graphs of the first level hierarchal S_n graph, we start by balancing the load of every two nodes via the edges that connect these sub-graphs within the Star topology structure. By the end of this phase we guarantee that all sub-graphs will have almost the same total number of loads since each sub-graph is represented as if it is a single node of the Star network structure in the first level hierarchy. It worth to mention here, that the load within each sub graph is not sorted at this stage. To complete this phase we need to make $n!/2$ parallel redistribution steps of load among every two nodes via a star structure edge. But at each of these parallel steps, there will be an n-1 sequential exchanges for each node with its n-1 neighbors within the star structure.
- Phase 2: Distributing the load size within each subgraph, this will the second level of the hierarchal $AS_{n,m,k}$ network, where each subgraph is an Arrangement graph representation, by the end of the phase 1, all subgraphs will have the same load size, then by redistributing the load sizes among these Arrangement graphs, the whole $AS_{n,m,k}$ network will have almost equal load sizes at each node. This phase requires $m!/2(m-k)!$ parallel redistribution steps of load among every two nodes via an Arrangement structure edge. But at each of these parallel steps, there will be a $k*(m-k)$ sequential exchanges for each node with its $k*(m-k)$ neighbors within the arrangement structure. By the end of this phase, all nodes will have almost the same load size, the following algorithm in Feg 2 describe the ASCDEM method of load balancing.

Note that n-1 is the number of neighbors of any processor in S_n:

1. *for* $p1 = 1; p1 \le n\text{-}1; p1{+}{+}$ // Start of phase#1
2.
3. *for* all neighbour nodes p_i and p_j which they differ in 1st and n+1 position of S_n do in parallel
4. Give-and-take p_i and p_j total load sizes of the two nodes
5. *TheAverageLoad* $p_{i,j}$ = Floor (*Load* p_i + *Load* p_i)/2
6. *if* (*Totalload* p_i >= excess *AverageLoad* $p_{i,j}$)
7. *Send* excess load p_i to the neighbour node p_i
8. Load p_i = Load p_i – extra load
9. Load p_j = Load p_j + extra load
10. *else*
11. Receive extra load from neighbour p_j
12. Load p_i = Load p_i + extra load
13. Load p_j = Load p_j – extra load

14. Repeat steps (1 to 12) $n!/2$ times // End of phase#1

15. *for* $p2 = 1$; $p2 \leq k*(m-k)$; $p2{+}{+}$ // Start of phase#2
16. *for* all neighbor nodes p_{ki} and p_{kj} which they differ in exactly one k position of $A_{m,k}$ do in parallel
17. Give-and-take p_{ki} and p_{kj} total load sizes of the two nodes
18. *TheAverageLoad* $p_{ki,kj}$ = Floor (*Load* p_{ki} + *Load* p_{ki})/2
19. *if* (*Totalload* p_{ki} >= excess *AverageLoad* $p_{ki,kj}$)
20. *Send* excess load p_{ki} to the neighbour node p_{ki}
21. Load p_{ki} = Load p_{ki} – extra load
22. Load p_{kj} = Load p_{kj} + extra load
23. *else*
24. Receive extra load from neighbour p_{kj}
25. Load p_{ki} = Load p_{ki} + extra load
26. Load p_{kj} = Load p_{kj} – extra load
27. Repeat steps (15 to 26) $m!/2(m-k)!$ times // End of phase#2

Fig. 2: The ASCDEM load balancing Algorithm

ASCDEM algorithm works on redistributing load balancing among all processors of the network, the two phases are done in parallel.

• Phase 1: The load balancing between the processors; subgraphes; of S_n based on ASCDEM algorithm is exchanged as in steps 2 to 12 in parallel, in first step the load exchange will be between all the processors in which they differ in 1st position and 2nd position for all the factor networks of S_n i.e. $S_{n\ -1}$. Then the same process will be repeated continually until it reach the neighbours p_j that is n positions far away from p_i. By the end of this phase all subgraphs will have almost the same total number of load sizes.

• Phase 2: The load balancing within the processors of each subgraph where each subgraph is an $A_{m,k}$ network. The ASCDEM algorithm in steps 15 to 26 performed in parallel, in first step the load exchange will be between all the processors in which they differ in exactly one k position for any two neighboring nodes, which means they are connected via an arrangement structure. Then the same process will be repeated continually all of the $m!/2(m-k)!$ neighbors. By the end of this phase all nodes of the network will have almost the same load size.

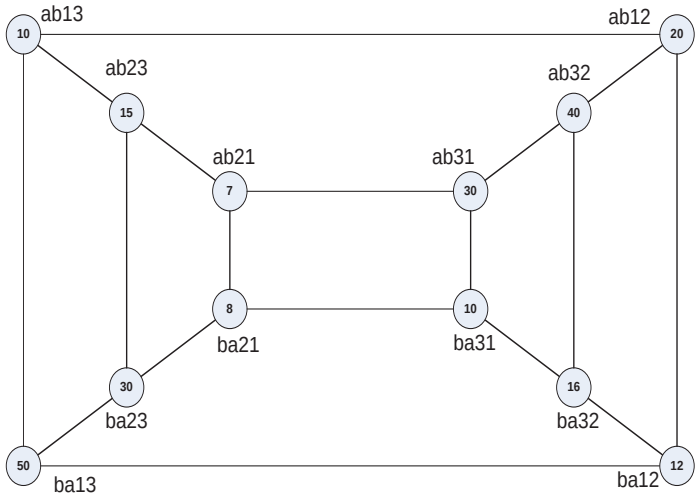


Fig. 3: Arrangement-star graph, $AS_{2,3,2}$ with initial loads

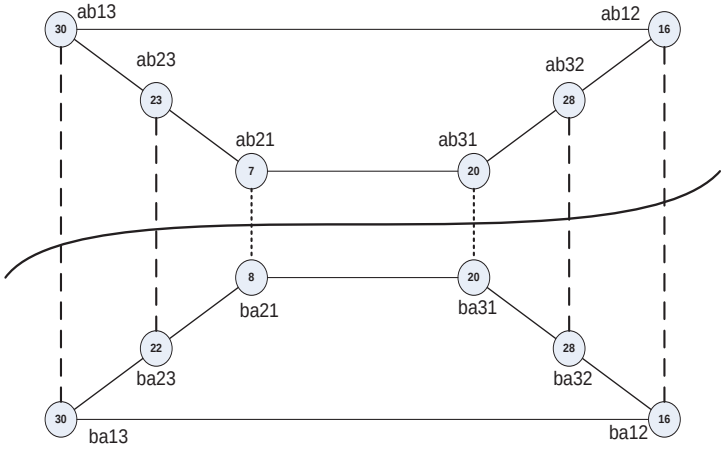


Fig. 4: Arrangement-star graph, $AS_{2,3,2}$ after performing ASCDEM phase 1

5. Conclusion

In This research we have investigated and proposed an algorithm named Arrangement-Star Clustered Dimension Exchange Method (ASCDEM), the proposed algorithm in based on the well-known efficient algorithm SCDEM which we proposed by Mahafza and et al which was named (CDEM).The main target of the ASCDEM algorithm is to redistribute the load balancing among all the processors of the Arrangement-star network as evenly as possible. As shown above the algorithm was able to redistribute the load balance among all the nodes of the $AS_{n,m,k}$ in an efficient approach.

A further work will be done on the proposed algorithm includes: total execution time, efficient load balancing accuracy, latency, number of communication moves and complexity speed to show that the ASCDEM efficiency in terms of mathematically analysis.

References

[1] 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.

[2] Ahmad Awwad, "vertex Product networks", University of Glasgow, Computer Science Dept. thesis, 2001.

[3] 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.

[4] K. Day and A. Tripathi, "A Comparative Study of Topological Properties of Hypercubes and Star Graphs", IEEE Trans. Parallel & Distributed Systems, vol. 5.

[5] Kaled 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.

[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] 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] S. Rajasekaran and D. Wei, "Selection, Routing, and Sorting on the Star Graph," *J. Parallel & Distributed Computing*, vol. 41, 1997, pp. 225-33.

[15] S. Lakshmivarahan, and S.K. Dhall, "Analysis and Design of Parallel Algorithms Arithmetic and Matrix Problems," McGraw-Hill Publishing Company, 1990.

[16] N. Imani et al, "Perfect load balancing on star interconnection network", J. of supercomputers, Volume 41 Issue 3, September 2007. pp. 269 – 286.

[17] Jehad Al-Sadi, "Implementing FEFOM Load Balancing Algorithm on the Enhanced OTIS-n-Cube Topology", Proc. of the Second Intl. Conf. on Advances in Electronic Devices and Circuits - EDC 2013, 47-5.

[18] Ranka, Y. Won, S. Sahni, "Programming a Hypercube Multicomputer", IEEE Software, 5 (5): 69 – 77, 1998.

[19] Zhao C, Xiao W, Qin Y (2007), "Hybrid diffusion schemes for load balancing on OTIS networks", In: ICA3PP, pp 421–432

[20] G. Marsden, P. Marchand, P. Harvey, and S. Esener, "Optical Transpose Interconnection System Architecture," *Optics Letters*, 18(13), 1993, pp. 1083-1085.

[21] Qin Y, Xiao W, Zhao C (2007), "GDED-X schemes for load balancing on heterogeneous OTIS networks", In: ICA3PP, pp 482–492.

[22] A. Menn and A.K. Somani, "An Efficient Sorting Algorithm for the Star Graph Interconnection Network," Proc. Intl. Conf. on Parallel Processing, 1990, pp.1-8.

[23] A. Al-Ayyoub and K. Day, "The Hyperstar Interconnection Network," J. Parallel & Distributed Computing, vol. 48, no. 2, 1998, pp. 175-199.

[24] K. Day and A. Tripathi, "Arrangement Graphs: A Class of Generalised Star Graphs," Information Processing Letters, vol. 42, 1992, pp. 235-241.

ISBN: 1-60132-444-8, CSREA Press ©

SESSION

CLOUD COMPUTING AND NOVEL APPLICATIONS

Chair(s)

TBA

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

Study of point-to-point communication latency for MPI implementations in cloud

F. Gomez-Folgar, G. Indalecio, N. Seoane, A. J. Garcia-Loureiro and T. F. Pena
Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)
Universidade de Santiago de Compostela, Santiago de Compostela, Galicia, Spain

Abstract— *This paper proposes a Heuristic Latency Model that characterizes MPI point-to-point communications in both cloud and non-virtualized infrastructures and analyzes the influence on the latency of VM allocation policies in cloud for MPI point-to-point communications. PingPong and PingPing communication patterns were analyzed over commodity hardware in two types of infrastructures: a cloud based on Apache CloudStack and a non-virtualized cluster. Two opposite allocation policies were considered: concentrated, in which the Virtual Machines are allocated in the same host, and scattered, in which the Virtual Machines are spread using different hosts. The MPI communications were analyzed for message sizes ranging from 1 byte to 4096 KiB. The results show that the cloud layer introduces a considerable overhead in the latency. However, if the cloud is employed to execute MPI applications, the concentrated allocation policy is between 30% to 90% better than the scattered policy for messages up to 4 KiB.*

Keywords: Cloud Computing, MPI, MPICH, OpenMPI, Performance

1. Introduction

Historically, cluster computing has played an important role on the support of high performance enterprise services and it has also become a fundamental tool in the support of scientific research. In this field, to facilitate the scientific community the access to the resources they need, several solutions have been released. Some of the proposed solutions have been focused on a specific research field such as physics simulations [1], bioinformatics [2], chemistry [3], oceanography [4] or climate modeling [5]. Others, such as science gateways [6], provided a more general approach. In this type of computing model, Message Passing Interface (MPI) has become the de-facto standard used for programming parallel computers and communicating processes on distribute memory systems. MPI supports point-to-point communications and collective operations and allows an efficient usage of NUMA architectures since it promotes memory locality. Two popular MPI implementations are MPICH and OpenMPI. MPICH is a high performance and portable implementation of the MPI standard. Currently, MPICH is used in nine of the top ten supercomputers of the TOP-500 ranking, including Tianhe-2 supercomputer. OpenMPI is an open-source MPI implementation developed and maintained by a consortium of academic, research and industry partners.

On the other hand, cloud technologies are of great interest in several fields (e.g. science, private companies, computing service providers) because of the trend to virtualize the services offering the computational capacity in the form of Virtual Machines (VMs), under the Infrastructure as a Service (IaaS) paradigm. In the scientific field, MPI applications are widely used on the cloud in a plethora of research areas such as nanodevice simulations [7] and high energy physics. An example of the latter is the CERN LHCb project [8], in which cloud resources have been integrated in the LHCb Distributed Computing. Also, research centers are offering computing facilities to their users via cloud in form of VMs and in some cases commodity hardware is employed. In these cloud platforms, the scheduler is a key component because it is responsible of selecting the host in which the user VM will be executed. Usually, the schedulers implement several scheduling policies in order to deploy VMs, and the police employed has an impact on the performance of the VMs and in the cloud.

In order to study the suitability of cloud infrastructures based on commodity hardware for executing MPI applications, in this work we will present: i) the MPI Heuristic Latency Model (MHLM) that allows characterizing MPI point-to-point communications in both cloud and non-virtualized infrastructures, ii) a study of the overhead that the cloud layer introduces on the latency of point-to-point MPI communications using MPICH and OpenMPI, iii) the impact on the latency of point-to-point MPI communications that employ two opposite cloud scheduling policies: concentrated allocation policy (in which the VMs are deployed in the same host), and scattered allocation policy (in which the VMs are deployed horizontally in different hosts), and iv) the impact on the latency of MPI point-to-point communications that the used MPI implementation, MPICH vs OpenMPI, introduce.

This paper is structured as follows: Section 2 describes the related work in this topic and our contributions to the research field. Section 3 introduces the proposed MPI communication model. Section 4 details the platforms employed (cloud and bare-metal cluster infrastructures). Section 5 presents the experimental results obtained in a cloud infrastructure based on Apache CloudStack [9] under different VM allocation schemes for MPICH and OpenMPI and their

comparison. The results obtained in two bare-metal clusters employing Intel and Realtek based Gigabit Ethernet Network Cards, and the comparison with the cloud MPICH results are also included in this section. Finally, the conclusions are discussed in the last section.

2. Related work

The related work in this area can be classified, mainly, in three topics: i) performance comparison of MPI over different networks, ii) performance evaluation of MPI using Amazon EC2, and iii) MPI models for MPI communication.

2.1 Performance comparison of MPI over different networks

There are studies analyzing the performance of MPI employing different networks such as 10 GbE [10], in which the authors have employed a simple two-node cluster environment based on dual AMD Opteron 246 processors running at 2.0 GHz with 2 GiB of PC3200 RAM. The 10 GbE network adapters were Intel PRO/10GbE LR that are based on Intel's 82597EX single-chip 10 GbE controller. The network cards were placed in the PCI-X slots of the computers. Both NetPipe and MPIBench benchmarks were executed. The MPI implementation selected to provide the MPI support was LAM-MPI over TCP. The obtained results showed that the performance of this type of network is fairly competitive with technologies like MPI over Quadrics.

In [11] a comparison of MPI implementations over Infiniband, Myrinet and Quadratics is presented using micro-benchmarks and level application benchmarks. Both NAS parallel Benchmarks and sweep3D benchmark were used employing 8-node clusters and, in this study, InfiniBand offered performance improvements for some applications compared with Myrinet and Quadrics.

2.2 Performance evaluation of MPI over Amazon EC2

Other studies are focused on evaluating the feasibility of running HPC applications in clouds comparing Amazon Cluster Compute Instances (CCI) and typical local clusters [12]. The CCI had 2 quad-core Intel Xeon X5570 processors, with 32 GiB of memory. The instances were allocated to users in a dedicated way, and were interconnected with 10 GbE networks. The local cluster employed for comparison purposes had 2 Intel Xeon X5670 6-core processors on each compute node with 32 GiB of RAM and were interconnected by a QDR InfiniBand network. In this local cluster, NFS was used as the shared file system. In this study the benchmarks used were NAS NPB, three real-world applications and Intel MPI Benchmarks (IMB). Results showed that applications with heavily message passing might fail to achieve good performance on the CCI platform employed due to the high latency that reduced the performance of applications that employed a lot of small MPI messages.

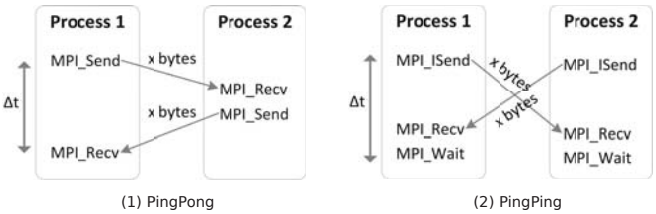


Fig. 1: PingPong and PingPing communication patterns.

2.3 MPI communication models

The last type of studies are focused on proposing models to characterize the MPI communications. Some of them extended the LogGP model [13] in order to include detailed hardware performance factors and also for including overheads due to different data structures. The modified LogGP model aimed to address the complete costs of the full communication path, including the communication cost inside the memory hierarchy. These models allowed evaluating and predicting the performance.

In [14], a new LogP-based model, LoOgGP is presented. LoOgGP extends the existing LogP model for long messages providing an accurate characterization of MPI applications based on micro-benchmark measurements.

The usage of these LogP-based MPI communication models needs the extraction of different communication parameters, but obtaining these parameters is usually not straightforward and sometimes not even possible. Furthermore, the required parameters have a strong dependence on the underlying hardware architectures.

2.4 Contributions of our work

Our work differs from the previous ones in the following:

1) Proposal of the MPI Heuristic Latency Model which allows characterizing the latency of point-to-point communications in a direct way for MPICH and OpenMPI implementations. The proposed model is simple and effective, and avoids collecting low level MPI parameters. Using this model, we have estimated the behavior of MPI communications on clouds based on commodity hardware.
2) Study of the overhead that cloud layer introduces over the latency of MPI point-to-point communications.
3) Study of the impact on the MPI point-to-point communications of two paradigmatic cloud allocation policies: concentrated and scattered.
4) Study of the impact of the MPI implementation, MPICH vs OpenMPI, on the latency of point-to-point communications.

3. MPI Heuristic Latency Model

The LogP-based MPI performance models described in the related work section are too exhaustive for being applied to a cloud infrastructure which introduces several virtualization layers and it is also usually composed by heterogeneous hardware. In most of these models, the parameters required to characterize the communications are very difficult to obtain in both homogeneous and heterogeneous infrastructures.

In this work, we characterize the MPI communications via a heuristic model which can be obtained almost in a direct way. In this model, the latency of communications is described as the sum of three contributions:

$$\mathrm{Lat}(b) = \mathrm{Lat}_0 \cdot + S_H \cdot \frac{1 + \mathrm{erf}(5 \cdot (Log(b) - S_P))}{2} + P_L \cdot (b^{P_S} - 1) \quad (1)$$

where Lat_0, S_H, S_P, P_L and P_S are the fitting parameters. In this model, b is the message size in bytes, Lat_0 represents the latency of the minimum message size employed, 1 byte, and S_H and S_P characterize, using the error function (erf), the sharp step-shaped increase in the latency that sometimes appears for message sizes between 32 bytes and 8 KiB. S_P is the position where the step appears, and S_H is the change in the latency due to the step. For larger sized messages, the latency exhibits a quasi-linear behavior that is characterized by P_L and P_S parameters, with P_S near to one. P_L represents the packet latency and P_S represents the packet slope. As we could see later, this model will be able to characterize the MPI communications for both cloud and bare-metal cluster infrastructures.

4. Platforms

In this section the platforms we have employed are presented: a cloud infrastructure based on Apache CloudStack using commodity hardware and also two cluster infrastructures employing two types of network cards, an Intel NIC and a Realtek NIC, for comparison purposes.

4.1 Cloud infrastructure

A cloud infrastructure based on Apache CloudStack has been employed. Apache CloudStack is an open-source software architecture that allows building several types of clouds: public, private and hybrid. Our cloud infrastructure uses Apache CloudStack 4.4 and KVM as the hypervisor employed in the CNs. The Apache CloudStack CNs employed have Intel Core i7-2600@3.4 GHz processors with 8 GiB of RAM and CentOS 6.3 64 bit as Operating System (OS). This processor has four cores and eight threads, a L1 cache size of 4x32 KiB for instructions and a 4x32 KiB for data, a L2 cache of 4x256 KiB, a L3 shared cache of 8 MiB, Intel Virtualization Technology (VT-x) and virtualization for directed I/O (VT-d). A 12 TiB NAS provides the NFS version 4 shared storage system for the infrastructure. The interconnection network of this infrastructure is based on RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 06).

4.2 Bare-metal cluster infrastructures

Two cluster infrastructures have been employed for comparison purposes. The first one employs Realtek NICs based on RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller. This infrastructure has Intel Core i7-2600@3.4 GHz processors with 8 GiB of RAM and CentOS 6.3 64 bit as OS. The second one employs Intel NICs based on Intel 82574L1 PCI Express Gigabit Ethernet Controller. This infrastructure has Intel Core i7-3770@3.4 GHz processors with 16 GiB of RAM and CentOS 6.3 64 bit as OS.

5. Case studies

In this work we have considered two testbeds: MPI on cloud and MPI on bare-metal clusters. In both cases, only point-to-point communications are considered: PingPong and PingPing. The first testbed, MPI on cloud, is intended to study the impact of the cloud allocation policies over the latency of MPI point-to-point communications in order to determine the most suitable cloud policy. The second testbed is intended to study the MPI performance on the bare-metal clusters, allowing comparing these results with the cloud ones and determining the overhead of using cloud on the MPI communications.

In order to evaluate the performance of MPI communications Intel MPI Benchmarks [15] have been employed in both cloud and bare-metal cluster infrastructures. The Intel MPI Benchmarks perform a set of MPI performance measurements for point-to-point and global communication operations for a range of message sizes. The benchmark aims to fully characterize the performance of a cluster system including node performance, network latency, throughput and the efficiency of the MPI implementation used. In order to characterize the communications, we have executed ten series of measurements for PingPong and PingPing communication patterns. For each series, 1000 repetitions have been executed for message sizes between 1 and 32 KiB, whereas for larger messages different repetitions were selected. For example, 640 repetitions for messages of 64 KiB, 320 for messages of 128 KiB, 160 for messages of 256 KiB, 80 for messages of 512 KiB, 40 for messages of 1024 KiB, 20 for messages of 2048 KiB, and, finally, 10 repetitions for messages of 4096 KiB.

In this work, we have been focused on PingPong and PingPing point-to-point communications. Using PingPong, depicted in Fig. 1(1), is possible to measure the startup and throughput of a single message sent between two processes, whereas PingPing, depicted in Fig. 1(2), can be used to measure start-up and throughput of single messages that are obstructed by oncoming messages. Note that in point-to-point communication patterns only two processes are involved.

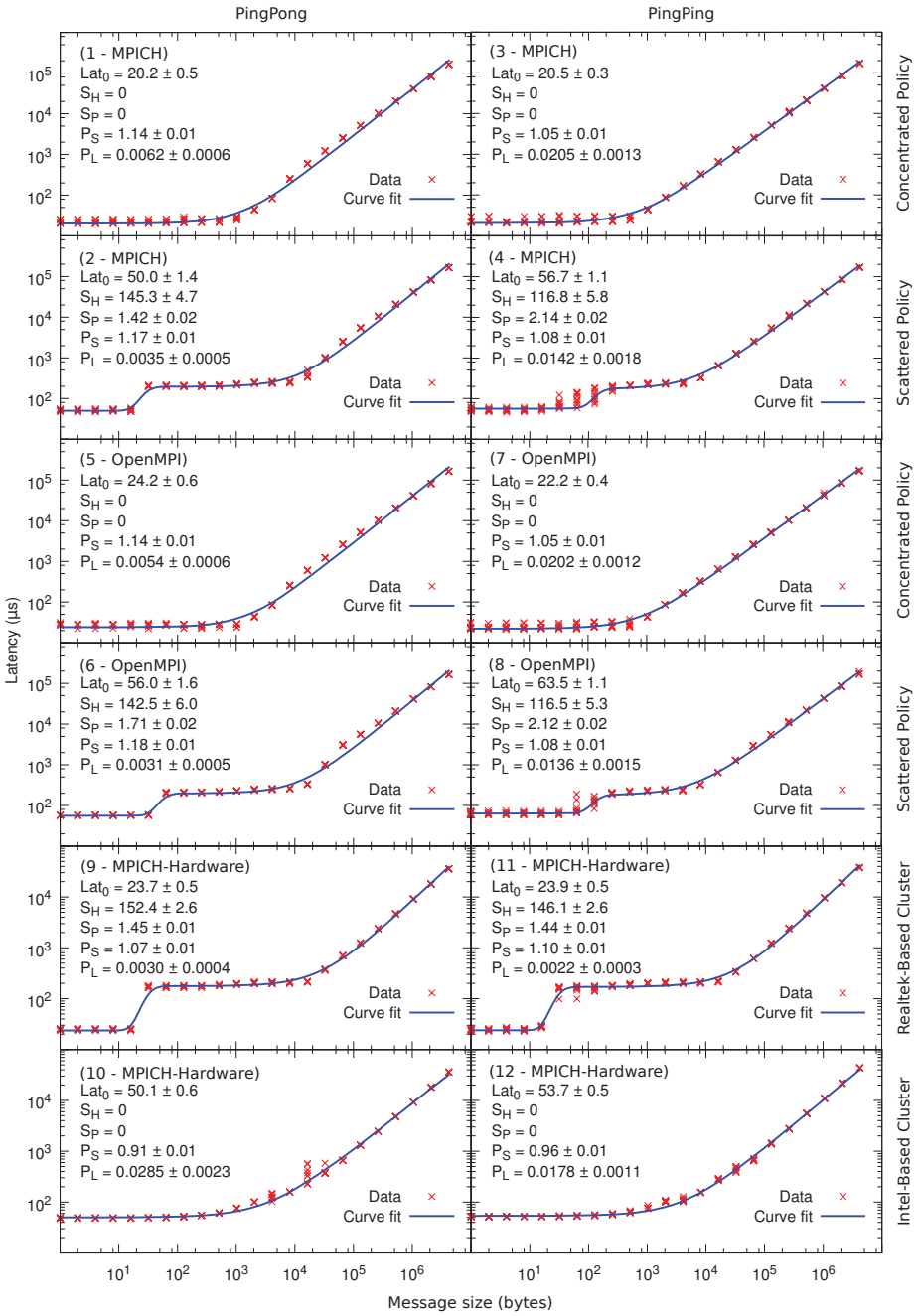


Fig. 2: Latency of PingPong and PingPing communication patterns for MPICH and OpenMPI under concentrated and scattered policies in cloud and for MPICH in Realtek-based and Intel-based bare-metal clusters.

5.1 MPI on Cloud

In order two study the latency of point-to-point MPI communications on cloud, a Virtual Cluster (VC) composed by two virtual nodes was deployed in the Apache CloudStack infrastructure described in 4.1. Each virtual node has 1 Core, 1 GiB of RAM and 10 GiB of hard disk. The operating system is CentOS 6.6 64 bit.

Two allocation policies were employed in order to study the MPICH and OpenMPI performance: concentrated and scattered. In the first policy, both virtual nodes are allocated in the same CN. In the second policy each VM is allocated in different CNs.

The concentrated policy is usually used to consolidate several VMs in the same Compute Node (CN) in order to reduce the number of CNs required to host the VMs, contributing to reduce the power consumption of the cloud infrastructure. The scattered policy is usually employed to distribute VMs horizontally among several CNs, in order to

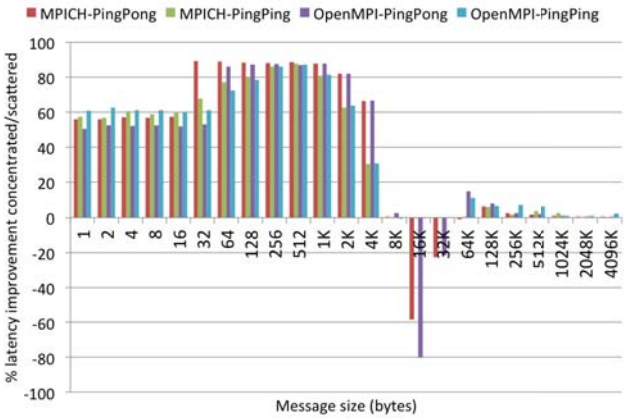


Fig. 3: Percentage of latency improvement for the concentrated allocation policy in comparison with the scattered policy vs the message size in bytes.

promote the availability of the virtual computing resources.

5.1.1 Concentrated allocation policy

Figs. 2(1) and 2(3) show the comparison of the experimental latency and the curve fit obtained from our proposed communication model (MHLM, eq. 1) using the concentrated allocation policy in cloud for MPICH for both PingPong and PingPing, whereas for OpenMPI, the results are depicted in Figs. 2(5) and 2(7). Note, for all the analyzed cases, the excellent agreement between the experimental data and the fitting provided by the model. In both MPICH and OpenMPI implementations, for messages up to 1 KiB the latency remains practically constant, around 20 μs for both communication patterns in the MPICH case and around 24 μs for PingPong and 22 μs for PingPing in the OpenMPI case, as described by the Lat_0 parameter of the MHLM model. For larger messages the latency increases with the message size with a quasi-linear behavior, as the P_S parameter of the model is near to one for PingPong and PingPing in both MPICH and OpenMPI implementations.

5.1.2 Scattered allocation policy

In Figs. 2(2) and 2(4) are depicted the comparison of the experimental latency and the curve fit obtained from our proposed communication model using the scattered allocation policy in cloud for MPICH for both PingPong and PingPing, whereas for OpenMPI, the results are depicted in Figs. 2(6) and 2(8). Note, again, for all the analyzed cases, the excellent agreement between the experimental data and the fitting provided by the model. In both MPICH and OpenMPI implementations, for short messages (less than 32 bytes) the latency remains practically constant, around 50 μs for PingPong and 57 μs for PingPing in the MPICH case, and around 56 μs for PingPong and 63 μs for PingPing in the OpenMPI case, as described by the Lat_0 parameter of the model. When the size of the message is 32 bytes for MPICH and 64 bytes for OpenMPI, there is a sharp step-shaped increase in the latency, as described by the S_H and S_P parameters of the model. For messages ranging from 32 bytes to 8 KiB, the latency slightly rises, whereas for messages larger than 8 KiB, the latency experiments a quasi-linear behavior as the P_S parameter of the model is near to one in both MPI communication patterns.

In the current bibliography, a step in the latency for message sizes around 32 bytes has been previously captured in [11], seen in Myrinet networks for the AlltoAll communication pattern, although no clear explanation for this phenomena has been offered to the best of our knowledge. As this step is not present when the concentrated policy is employed in the cloud, it can not be caused by the Virtio network device with the Virtio-PCI Kernel driver used in the virtual compute nodes of the VC. However, when the scattered allocation scheme is used, the underlying network hardware adapter must be used to communicate the virtual nodes among them. Due to the fact that the underlying hardware is using the same OS as the VMs and the only difference is the Realtek-based NIC, we believe that this step has to be introduced by the Realtek-based NIC or the r8169 network kernel module.

5.1.3 Influence of the cloud allocation policy

In this subsection, the impact of the two cloud allocation policies on the latency of the MPI point-to-point communications has been analyzed. The percentage of the improvement in the latency for the concentrated allocation scheme in comparison with the scattered allocation policy is depicted in Fig. 3 for PingPong and PingPing communication patterns under MPICH and OpenMPI implementations. As we can see, for messages up to 4 KiB, the use of the concentrated policy produces an important improvement in the latency of the communications ranging from 30% to 90% for both MPI implementations and communication patterns. For larger messages, the latency improvement is limited to 15% in the best case, as seen for 64 KiB messages using OpenMPI-PingPong. However, for messages ranging from 16 KiB to 32 KiB, the use of the concentrated allocation policy produces an increase in the latency of the PingPong communication pattern up to 80% when compared to the scattered one.

5.1.4 Influence of the MPI implementation for concentrated and scattered cloud policies

In this subsection, we analyze the impact on the latency of the MPI implementation used. The percentage of the improvement in the latency for MPICH in comparison with OpenMPI under concentrated and scattered allocation policies vs the message size is depicted in Fig. 4. With some exceptions, in general, the MPICH implementation obtains better latency, with up to 21% improvement for

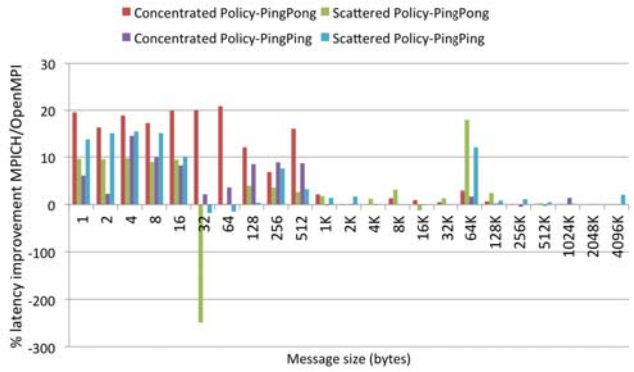


Fig. 4: Percentage of latency improvement for MPICH in comparison with OpenMPI vs message size in bytes. Note the negative axis has been changed by a factor of ten in order to improve the display quality.

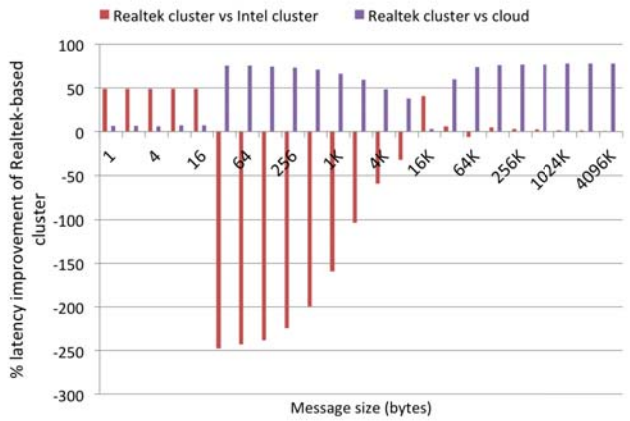


Fig. 5: Percentage of latency improvement of PingPong for MPICH between both bare-metal clusters, and between the Realtek-based cluster and the cloud under scattered allocation policy vs message size in bytes.

messages up to 1 KiB. Also, the concentrated allocation policy produces the best results. For larger messages, for the MPI implementations analyzed, there is no clear winner and the influence of the allocation police used is even not remarkable. Note the exception that happens for PingPong in the scattered allocation scheme for messages of 32 bytes that produces a degradation of MPICH performance up to -249%, when compared to the OpenMPI result. This phenomena is due to the fact that the sharp step-shaped increase in latency happens for messages of 32 bytes for MPICH, by contrast with OpenMPI, in which the step is seen at 64 bytes. For example, for messages of 32 bytes, the latency obtained for OpenMPI is around 58 μs, whereas for MPICH is around 202 μs.

5.2 MPI on bare-metal clusters

In this subsection, the MPICH results obtained employing the two bare-metal cluster architectures described in section 4.2 are presented. The first cluster employs the popular Realtek RTL8111/8168/8411 NIC, whereas the second one employs Intel 82574L1 NICs. The communications among two compute nodes for PingPong and PingPing point-to-point patterns were compared.

5.2.1 Realtek NIC based cluster

A comparison between the experimental latency and the curve fit obtained from our proposed communication model for PingPong and PingPing communication patterns is presented in Figs. 2.9 and 2.11, respectively. Results show, again, a sharp step-shaped increase in latency of MPI of around 152 μs for PingPong and 146 μs for PingPing, for messages larger than 32 bytes, whereas for shorter messages the latency remains around 24 μs. For messages larger than 8192 bytes, the latency experiments a quasi-linear increase as the P_S parameter of the model is near to one. Note, again, for all the analyzed cases, the excellent agreement between the experimental data and the fitting provided by the model.

5.2.2 Intel NIC based cluster

Figs. 2.10 and 2.12 show a comparison between the experimental latency and the curve fit obtained from our proposed communication model for PingPong and PingPing communication patterns. Results do not show the step-shaped increase in the latency of MPI, previously seen in the Realtek case. Instead, the latency of the messages smaller than 8192 bytes remains practically constant, around 50 μs for PingPong and around 54 μs for PingPing. For messages larger than 8192 bytes, the latency experiments a quasi-linear increase. As the OS of the bare-metal clusters is the same but employs different NICs, as said before, the sharp step-shaped increase in the latency, seen in the Realtek case, should be due to the different Kernel driver or the NIC used.

5.2.3 MPICH bare-metal clusters and cloud comparison

In this subsection the comparison of PingPong MPICH is performed for two bare metal clusters (Realtek-based and Intel-based), and also for a Realtek-based cluster and the cloud infrastructure under scattered allocation policy. In Fig. 5 the obtained results are shown.

On one hand, the comparison between Realtek-based and Intel-based bare-metal clusters is depicted as red bars. As we can see, for messages between 1 and 16 bytes the use of the Realtek NIC introduces an improvement of 50% in the latency if the Intel NIC is used as reference. However, for messages between 32 bytes and 8 KiB the latency obtained for the Realtek NIC is worse than the Intel one, up to -250%. For larger message sizes, the latency remains almost the same for both NICs. The different behavior between the Intel and Realtek bare-metal clusters can be due to the different NIC designs and/or different network kernel

module implementations.

On the other hand, comparing the results obtained in Realtek-based cluster vs cloud it can be seen for message sizes ranging from 1 to 16 bytes that the cloud does not introduce a remarkable overhead, as the improvement on the latency for the Realtek-based cluster is around 7%. However, for larger messages, the use of the cloud layer introduces an important overhead on the latency. For these ranges of message sizes, the latency obtained in the physical infrastructure is up to 78% less than the cloud ones. This phenomena can be due to the network virtualization layers that the cloud introduces, producing an overhead in the communications.

6. Conclusion

In this paper, a MPI Heuristic Latency Model that allows characterizing the MPI point-to-point communications on clouds based on standard commodity hardware has been proposed. As clouds hides the underlying hardware, this model avoids the inherent overhead and complexity of the hardware-dependent LogP-based models, and was able of characterize all studied cases, MPICH and OpenMPI, considering two different VM allocation policies: concentrated and scattered. Also, we have studied: i) the overhead that the cloud layer introduces on the latency of MPI point-to-point communications in comparison with the underlying hardware, ii) the impact of two paradigmatic cloud allocation policies, concentrated and scattered, on the MPI point-to-point communications, and iii) the impact of the MPI implementation, MPICH vs OpenMPI, on the latency.

After analyzing the effects on the latency for MPI point-to-point communications on the platforms studied, we can conclude that i) the use of the cloud introduces an important overhead in the latency, as the latency obtained in the physical infrastructure is up to 78% less than the cloud ones, ii) the use of different allocation policies on cloud has an impact on the latency of point-to-point communications, as the latency for message sizes ranging from 1 byte to 4 KiB is improved in the range between 30% and 90% when the concentrated policy is used (for larger messages the maximum latency improvement is limited to 15% in the best case), but the use of this allocation police can produce a degradation in the latency up to 80%, for messages ranging from 16 KiB to 32 KiB, iii) in general, the MPICH implementation, in comparison with OpenMPI, obtains latency improvements up to 25% for messages up to 1 KiB, whereas for larger messages there is not a clear winner and the influence of the cloud allocation police used is not noticeable.

As seen, the use of cloud layer and the VM deployment policies have an important impact on the performance of MPI point-to-point communication. Therefore, it is necessary to have this fact in mind when deploying MPI applications in this type of infrastructures. The optimal situation would require the analysis of the MPI application's execution profile in order to select the most appropriate scheduling policy in the cloud that allows improve the latency of MPI point-to-point communications.

As future work, we plan to extend our model to collective MPI communications, including other performance parameters.

Acknowledgment

This work has been supported by FEDER funds and by Spanish Government (MCYT) under projects TEC2010-17320, TIN-2013-41129-P and TEC2014-59402-JIN, and by the Spanish Ministry of Education, Culture and Sports under FPU grants FPU12/05190 and FPU12/02916.

References

[1] A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees, "DIRAC: A Scalable Lightweight Architecture for High Throughput Computing," in *Fifth IEEE/ACM International Workshop on Grid Computing*. IEEE, 2004, pp. 19–25.

[2] M. Mirto, S. Fiore, I. Epicoco, M. Cafaro, S. Mocavero, E. Blasi, and G. Aloisio, "A Bioinfomatics Grid Alignment Toolkit," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 752–762, jul 2008.

[3] C. Manuali, A. Laganà, and S. Rampino, "GriF: A Grid framework for a Web Service approach to reactive scattering," *Computer Physics Communications*, vol. 181, no. 7, pp. 1179–1185, jul 2010.

[4] C. Cotelo, A. Gómez, J. I. López, D. Mera, J. M. Cotos, J. P. Marrero, and C. Vázquez, "Retelab: A geospatial grid web laboratory for the oceanographic research community," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1157–1164, oct 2010.

[5] D. Bernholdt et al., "The Earth System Grid: Supporting the Next Generation of Climate Modeling Research," *Proceedings of the IEEE*, vol. 93, no. 3, pp. 485–495, mar 2005.

[6] N. Wilkins-Diehr, "Special Issue: Science Gateways - Common Community Interfaces to Grid Resources," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 6, pp. 743–749, apr 2007.

[7] F. Gomez-Folgar, E. Comesana, R. Valin, A. Garcia-Loureiro, and T. F. Pena, "Nanodevice simulations on CloudStack," in *2013 Spanish Conference on Electron Devices*. IEEE, feb 2013, pp. 9–12.

[8] M. Ú. García, V. M. Muñoz, F. Stagni, B. Cabarrou, N. Rauschmayr, P. Charpentier, and J. Closier, "Integration of Cloud resources in the LHCb Distributed Computing," *Journal of Physics: Conference Series*, vol. 513, no. 3, p. 032099, jun 2014.

[9] Apache CloudStack. http://cloudstack.apache.org

[10] J. G. Hurwitz and W.-C. Feng, "Analyzing MPI performance over 10-Gigabit Ethernet," *Journal of Parallel and Distributed Computing*, vol. 65, no. 10, pp. 1253–1260, oct 2005.

[11] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda, "Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing - SC '03*. New York, New York, USA: ACM Press, 2003, p. 58.

[12] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen, "Cloud versus in-house cluster," in *State of the Practice Reports on - SC '11*. New York, New York, USA: ACM Press, 2011, p. 1.

[13] T. T. Le and J. Rejeb, "A detailed MPI communication model for distributed systems," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 269–278, feb 2006.

[14] D. R. Martinez, J. C. Cabaleiro, T. F. Pena, F. F. Rivera, and V. Blanco, "Accurate analytical performance model of communications in MPI applications," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, may 2009, pp. 1–8.

[15] Intel MPI Benchmarks. https://software.intel.com/en-us/articles/intel-mpi-benchmarks

Runtime network-level monitoring framework in the adaptation of distributed time-critical Cloud applications

Salman Taherizadeh[1,4], Andrew C. Jones[2], Ian Taylor[2], Zhiming Zhao[3], Paul Martin[3], and Vlado Stankovski[4]
[1]Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia
[2]School of Computer Science and Informatics, Cardiff University, Cardiff, United Kingdom
[3]Informatics Institute, University of Amsterdam, Amsterdam, Netherlands
[4]Faculty of Civil and Geodetic Engineering, University of Ljubljana, Ljubljana, Slovenia
{Salman.Taherizadeh, Vlado.Stankovski}@fgg.uni-lj.si, {JonesAC, TaylorIJ1}@cardiff.ac.uk, {Z.Zhao, P.W.Martin}@uva.nl

Abstract - *Many distributed time-critical applications have emerged on the Internet in recent decades, involving for example sensor-based early warning systems, online gaming and instant messaging. Such applications can be virtualised and distributed in a federated Cloud environment. Ensuring that these types of application are able to offer favourable service quality has been a challenging issue due to runtime variations in network conditions intrinsic to connections between individual application components replicated and distributed across different Cloud infrastructures. In this paper, we propose a lightweight method for performing network-level monitoring that can be used to guide the autonomous selection of optimal connections between running components, so improving distributed application performance at runtime. This solution contributes towards realising self-adaptation capabilities for time-critical applications by implementing a non-intrusive monitoring technique for key network-level parameters including round-trip time (RTT), packet loss, throughput and/or jitter. The experimental results show that the proposed framework has a low communication overhead and requires little processing power and memory capacity.*

Keywords: Monitoring System, Network QoS, Distributed Time-Critical Applications, Multi-Cloud Environment

1. Introduction

In recent years, time-critical systems such as early warning systems, multimedia applications and Cloud-based gaming have emerged as Internet services which are increasingly widely used and important, especially to organisations that want to leverage the benefits of distributed applications. Decomposing such complex applications, each application component can be distributed to a different machine such that each component interacts with other components regardless of deployment location. Accordingly, by using a multi-Cloud environment, companies can use Cloud infrastructures to run and replicate their application components in different locations.

Time-critical applications have specific network QoS (Quality of Service) requirements between their components, such as demanding minimal delay and packet loss, and require suitable support to achieve guaranteed application performance for their users. This is a challenge because the network connection quality between different components, as a key influencer of the overall application performance, is difficult to maintain when Cloud infrastructures continuously change. In particular, time-critical Cloud application providers have to dynamically adapt their services to network conditions to deliver high performance and a seamless experience. In essence, the main problem encountered by time-critical service providers is that there are limited automated and intelligent adaptation capabilities in existing Cloud infrastructures based on real-time network features that can be used to satisfy application performance requirements. Therefore, to avoid application performance issues, providers must carefully monitor the network QoS of connections within and between all of their own servers hosting application components in different Cloud infrastructures; all while being non-intrusive to the ordinary operation of the application [1].

This paper presents a lightweight monitoring approach based upon a non-intrusive design intended to enable distributed applications to autonomously reconfigure and adapt to changing network conditions at runtime. Replicating application components in different Cloud infrastructures to increase availability and reliability under various network conditions and varied amounts of traffic, and dynamically connecting each component to the best possible component in each different tier, together offering fully-qualified network performance, is often an essential requirement for providers of time-critical applications running on the Cloud. If such a network performance metric can be measured, then the system can be made automatically capable of improving the deployment of an application when performance drops. Under our proposed system, the network performance metric is a combination of measurements including network throughput, round-trip time, packet loss and/or jitter, which can be measured and responded in order to enhance application performance and hence user experience.

The rest of the paper is organised as follows. Section 2 presents summary of related work supporting network-level Cloud monitoring. Section 3 describes the use case. Section 4 discusses the architecture and implementation of our proposed approach, followed by empirical evaluation results and finally conclusion respectively in Sections 5 and 6.

2. Related Work

To achieve the objective of providing high-quality services in time-critical systems, it is essential to implement trustworthy techniques that can be responsible for maintaining QoS when considering the limitations imposed by the network. There have been many research approaches, all trying to provide QoS guarantees over Cloud networks. Anouari and Haqiq [2] analysed the performances of VoIP and Video stream traffic that is characterized by the ability to transmit real-time and interactively visual and auditory information. These types of traffic are highly delay-intolerant and need high priority transmission. Addressing this concern, their study was focused on using different service classes with respect to QoS parameters such as average delay, average jitter and throughput. Sodangi [3] designed and simulated two Cloud-based networks. The first scenario involved running multimedia applications (voice and video) and the second one involved running traditional applications (email, file transfer, web browsing). These were compared, and the main finding was that multimedia applications need appropriate throughput and are sensitive to delay, resulting in data loss, whereas traditional applications can use minimum throughput and with typical data loss levels are normally insensitive to changes in delay. In [4], the results showed that network performance varies substantially from one Cloud provider to another. Their approach can guide customers in selecting the best-performing provider for their applications. To measure the performance of internal connections between a customer's instances and to the shared services offered by a Cloud, they used throughput and latency as metrics.

With regard to network-based measurement, associated QoS attributes change constantly and so network-layer parameters need to be closely monitored. Table 1 shows the most important metrics to be analysed for Cloud network measurement: (I) Throughput, which is the average rate of successful data transfer through a network connection. (II) RTT, which is the time elapsed from the propagation of a message to a remote place to its arrival back at the source. (III) Packet loss, which occurs when one or more packets of data traveling across a network fail to reach their destination. (IV) Jitter, which is the variation in the delay of successive packets.

Lampe *et al.* [5] mostly focused on the QoS parameter of latency, since this parameter plays an important role in the overall game experience. The authors conducted their research only on network latency measurement. Their experiments could be extended through the consideration of additional metrics; for example, the effects of network disturbances, such as increased packet loss or fluctuating throughput. Samimi *et al.* [6] introduced a model including a network-based monitoring system and the enabling of dynamic instantiation, composition, configuration and reconfiguration of services on an overlay network. Mohit [7] selected throughput, RTT and data loss for Cloud network measurement. The author suggests a solution that involves use of different technologies such as high-capacity edge routers which have a high cost and cannot be afforded in all use cases. Cervino *et al.* [8] presented an experimental validation of the Cloud infrastructure's ability to distribute streaming sessions with respect to some key streaming QoS parameters. Next, the authors performed experiments to evaluate the benefits of deploying VMs in Clouds to aid P2P streaming, by measuring the QoS improvement. Chen *et al.* [9] focused on the users' perspective in Cloud gaming systems; from their point of view, the QoS metrics have an important effect on gaming experience. In other words, they proposed a suite of measurement techniques to evaluate the QoS of Cloud gaming systems.

Table 1. Relevant research on network-based measurement of Cloud environment performance

Title	Field	Measured Metrics	Results
To frag or to be fragged - an empirical assessment of latency in cloud gaming [5]	Audio/video stream	Limitations of the network infrastructure, such as high latency, potentially affect the QoS of the cloud gaming system.	While cloud gaming substantially reduces the demand of computational power on the client side, thus enabling the use of thin clients, it may also affect the QoS through the introduction of network latencies.
Service clouds: distributed infrastructure for adaptive communication services [6]	Adaptive communication services	Monitors carry out measurements on data streams. The metrics can be generic in nature (e.g., packet delay and loss rate) or domain-specific (e.g., jitter in a video stream).	Service clouds are distributed infrastructures which are designed to facilitate rapid prototyping and deployment of adaptive communication services in clouds, and they are appropriate choices when service platforms' workloads are dynamic or they need a lot of resources.
A comprehensive solution to cloud-traffic tribulations [7]	General systems	Regarding network-based measurement, the three significant parameters to be analysed are throughput, RTT and data loss.	Computation-based infrastructure measurement is insufficient for the optimal operation and future growth of the cloud. Network-based measurements of the cloud computing service are also very important.
Testing a cloud provider network for hybrid p2p and cloud streaming architectures [8]	Online real-time streaming	Authors considered four very important network parameters for video/audio streaming and for many other real-time services: bandwidth, delay, jitter and packet losses.	Using a cloud network infrastructure to cross continents has improved the majority of QoS problems. It means that using connections between distant cloud datacentres can help to improve the QoS response of streaming even in videoconferencing P2P systems.
On the quality of service of cloud gaming systems [9]	Cloud gaming systems	Authors concentrate on the metrics related to network conditions namely delay, packet loss, bandwidth and also other types of metrics which are graphic quality and frame rate.	Packet loss and bandwidth limitations impose negative impact on the frame rates and the graphic quality in the cloud gaming systems. The network delay does not predominantly affect the graphic quality of the games on the cloud gaming systems.

3. Use Case

A typical example for time-critical services considers disaster early warning systems developed for the purpose of providing proper alert before disaster occurs. Figure 1 depicts the basic framework of such a system.

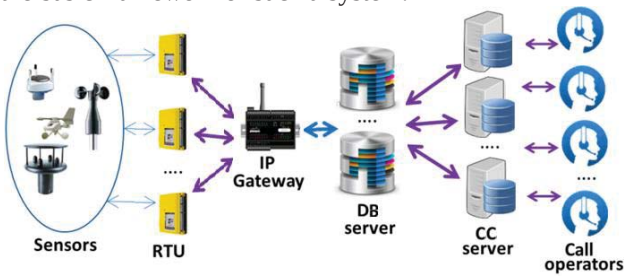


Figure 1. The basic framework of an early warning system

All application components are defined in Table 2. IP Gateway and RTU cannot be virtualized as these components have physical items like attached antennas.

Table 2. Components of a disaster early warning system

Component	Functionality	Type
Call Operator	The Call Operators decide whether or not to send an alert to emergency systems or to the public.	Dedicated and ad-hoc agents
CC Server (Contact Centre Server)	The server checks sensed data stored in DB Server and statistics in real-time and sends notifications (such as e-mail, SMS or voice call via SIP based IP telephony or ordinary PSTN) to Call Operators if values are outside predetermined thresholds for sensors.	Apache web server
DB Server (Database Server)	This is a Time Series Database which is used for storing and handling sensed values indexed by time.	Cassandra
IP Gateway	The IP Gateway is a node that allows communication between networks. It receives data over direct radio link or GSM/GPRS from sensors, aggregates the data and sends the data to the database.	E.g. TA900e or Cisco-ASA
RTU	Remote terminal units (RTUs) connect to sensors in the process and convert sensor signals to digital data.	E.g. Modbus-RTU
Sensors	Sensors can measure temperature, barometric pressure, humidity and other environmental variables.	E.g. DHT11

In this case, the overall application performance is the system's reaction time, which means the length of time taken from sensor data acquisition to when a notification is sent to the Call Operator. This application performance metric is mainly affected by the network communication quality between the DB Server and the CC Server. Due to the Cloud-based environment, several DB Servers and CC Servers can be running in various Cloud providers' infrastructures in different geographical locations, all connecting with each other. Assume that the data is replicated among DB Servers and also that each CC Server is dedicated to a certain number of Call Operators who must send warning messages through various communication channels in each region. The proposed mechanism aims at providing the ability to connect each CC Server to the best possible DB Server which has the superior network QoS in relation to the CC Server. Therefore, a Monitoring Probe is running on each CC Server's VM to measure the network performance metric (NPM) between the CC Server and every single DB Server. Our proposed approach shows how different Cloud providers can offer varying network performance in the execution of real-time applications depending on various aspects. We introduce (1) to calculate NPM including three important network parameters which are network throughput (NT), average delay (AD) and packet loss (PL).

$$NPM = \frac{\left(1 - \frac{PL}{100}\right) * NT}{AD} \qquad (1)$$

In this use case, jitter is not taken into account; since this disaster early warning system is not a real-time service involving e.g. video/audio streaming in which lower jitter is advantageous (because lower jitter means the delay times are more consistent, and therefore a connection is more stable).

4. Architecture and Implementation

Cloud-based applications can be viewed from both design-time and run-time perspectives. In the design-time view, the whole Cloud service, including application topology and application components, is shown. In the run-time view, instances of application components are examined as they are deployed and executed in VMs. Considering these two views, Figure 2 presents an overview of the proposed architecture to make an effective improvement in the performance of the aforementioned disaster early warning system. In this figure, at run-time, for example there are three running CC Servers and two running DB Servers which are dynamically connected to each other in the best possible way to maximise the overall application performance.

Network QoS between these two components (DB Server and CC Server) strongly influences the overall application performance

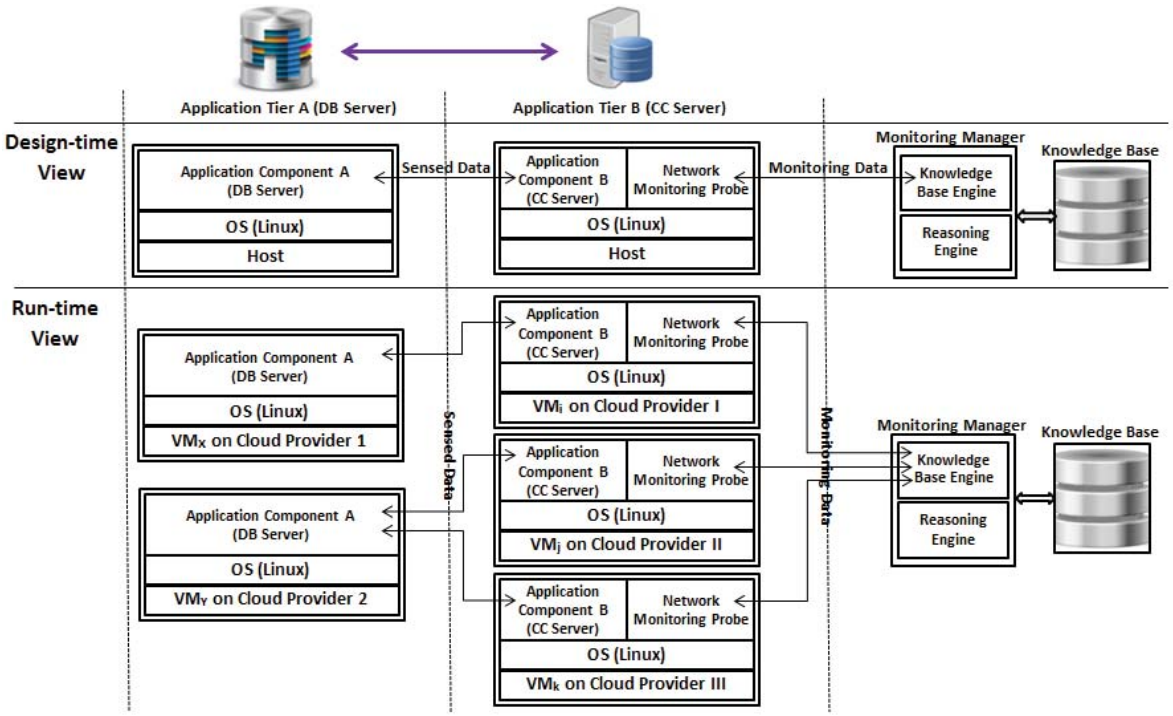


Figure 2. Overview of the proposed architecture to improve the performance of early warning system

```
1:/* Probe resides in VMi  where the CC Server is running */
2:/* Packet Loss (PL), Network Throughput (NT), Average Delay (AD) */
3:/* Network Performance Metric: NPM */
4:while(true){
5:   TS ← TimeStamp()
6:   for each DB Server running on a VMx do {
7:      PL ← Calculate_PL(VMi, VMx)
8:      NT ← Calculate_NT(VMi, VMx)
9:      AD ← Calculate_AD(VMi, VMx)
10:     NPM ← ((1 - (PL/100)) * NT)/AD
11:     Message ← Make_Message(VMi, VMx, TS, NT, AD, PL, NPM)
12:     Send_To_Knowledge_Base_Engine(Message)
13:  } // end of for
14:  wait(interval)
15:} // end of while
```

Figure 3. Pseudocode for Monitoring Probe which is deployed along with the CC Server

As depicted in Figure 2, this monitoring system employs a number of distinct components. The *Network Monitoring Probe* is responsible for monitoring network QoS parameters of links between instances of two application components (the DB Server and the CC Server). For each CC Server, the network performance metric for every connection with potential DB Servers is simultaneously evaluated periodically at regular intervals by a Network Monitoring Probe. The pseudocode of the developed algorithm for the Monitoring Probe is depicted in Figure 3.

The *Monitoring Manager* is responsible for aggregating and analysing network QoS data received from Monitoring Probes. The Monitoring Manager consists of two parts; a Knowledge Base Engine and a Reasoning Engine. The *Knowledge Base Engine* is responsible for all the work that controls the collection of network QoS values as RDF (Resource Description Framework) triples, along with actually storing and also retrieving these data on disk. This proposed monitoring system incrementally stores information about the environment in a *Knowledge Base* (KB) that will be used for interoperability, integration, analysing and optimisation purposes. Maintaining a KB enables analysis of long-term trends, supports capacity planning and allows for a variety of strategic analysis like year-over-year comparisons and usage trends. The *Reasoning Engine* is responsible for network-based QoS analysis and evaluating relevant policies such as interpreting the network performance metrics between CC Servers and DB Servers. Therefore, based on network-based analysis, the Reasoning Engine will return decisions such as which CC Server should be automatically and dynamically connected to which DB Server when current conditions do not satisfy the expected requirements. Each alternative possesses different attributes which can be compared and evaluated using network-level criteria; the proposed framework via the Reasoning Engine can then choose the best one at real-time.

For our experiments, the actual network QoS parameters for time-critical services are measured by using ICMP (Internet Control Message Protocol) requests. The "ping" tool operates by sending echo request packets to the target host and waiting for an echo reply packets. It measures the round-trip time from transmission to reception and reports errors and packet loss. We used different command options to enable the monitoring system to adjust the size of the ICMP packet, determine the number of echo requests to send, and specify wait period between pings. Moreover, we used an option to set the "Do Not Fragment" bit on the ICMP packet which does not allow fragmentation to occur in the path of the data flow by intermediate routers. We implemented the Knowledge Base Engine using a Jena Fuseki server to load an

RDF dataset and make it accessible through a REST API as a SPARQL endpoint, to expose the CRUD operations for creating, retrieving, updating and deleting records. Jena Fuseki is an open source, lightweight database server, easy to install and able to efficiently store large numbers of RDF triples on disk [10].

5. Empirical Evaluation Results

As a preliminary set of proof-of-concept results to test the design of the monitoring components, we performed an initial set of experiments to measure the network-based metrics between a particular CC Server ($Host_i$) and two replicated DB Servers ($Host_x$ and $Host_y$) at runtime. Periodically (every 10 seconds), our Network Monitoring Probe deployed on the VM, hosting also the CC Server, sends 10 ICMP packets to the both DB Servers with a 0.2 second delay between sending each packet, and then calculates the network metrics. The CC Server will then be automatically connected to the DB Server providing the highest connection quality.

The following experiment shows how this configuration allows us to check the network-based QoS features related to two different connections with the same source: the first link between $Host_i$ and $Host_x$ and the second one between $Host_i$ and $Host_y$. Table 3 shows features of these three hosts. $Host_i$ which is a CC Server, belongs to the Flexiant Cloud infrastructure in the United Kingdom. Two DB Servers—$Host_x$ and $Host_y$—are in different locations in Slovenia and belong to different Cloud infrastructure providers: the ARNES (the Academic and Research Network of Slovenia) and the FGG (the Faculty of Civil and Geodetic Engineering, University of Ljubljana).

Table 3. Features of infrastructures used in our experiment

Feature	$Host_i$	$Host_x$	$Host_y$
Type	CC Server	DB Server	DB Server
OS	Ubuntu 14.04	Debian 7.8	Ubuntu 14.04
CPU(s)	2	1	1
CPU MHz	2600.030	2666.760	2397.222
Memory	1024 MB	1024 MB	1024 MB
Speed	1000 Mbps	1000 Mbps	1000 Mbps
IP	109.231.121.55	193.2.91.109	194.249.0.142
Cloud	Flexiant	FGG	ARNES

The round-trip delay gives the total end-to-end time, and hence is an important metric in evaluating the performance of the time-critical Cloud service. A lower average delay is always preferred; because it takes less time for packets to reach and return between the servers. Therefore, Figure 4 shows that according to the average delay, the network quality of $Host_x$ is a little bit better than that of $Host_y$ for a period of time.

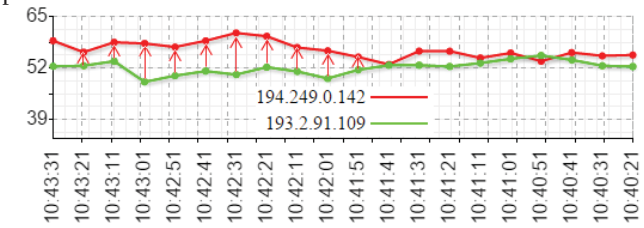


Figure 4. Average Delay (ms) for 200 second monitoring window

Time-critical Cloud applications require network services with minimal packet loss. The possibility of packet loss increases as traffic travels a longer distance and over more hops in the network. Data loss has one of the biggest impacts on time-critical applications, seriously affecting the quality of services, and this is the reason that the network should be engineered for zero percent packet loss. Our test system showed that packet loss ratio was zero, which indicates that there was no drop in either connection related to the servers deployed during the experiment.

Network throughput is the amount of data moved successfully from one place to another in a given time period. It is possible to benchmark network throughput and find bottlenecks in the network to ensure that network interfaces are fast enough to achieve desired performance. The amount of traffic in current high-speed, heavy-traffic and multi-service networks increases continuously, and traffic characteristics change heavily in time—for example network throughput fluctuates due to time of day, server backup operations, DoS (Denial of Service) attacks, scanning attacks and other anomalous network traffic. The performance of Cloud services must be independent of such states and must continue to behave reliably in all possible cases. Our proposed monitoring system sends ICMP packets, each one containing 500 bytes of data, from the first node (CC Server) to the second node (DB Server). Then it receives the results including the average delay ("*Avg*"). To make the proposed monitoring system lightweight, network throughput was estimated from the latency based on (2), which converts bytes per millisecond into kilobytes per second:

$$Network\ Throughput\ (KB/s) = \frac{500 * 10^6}{Avg * 2^{10}} \qquad (2)$$

Figure 5 shows no major variation in throughput belonging to either server; however in real-time systems, continuous fluctuation is important to be taken into account.

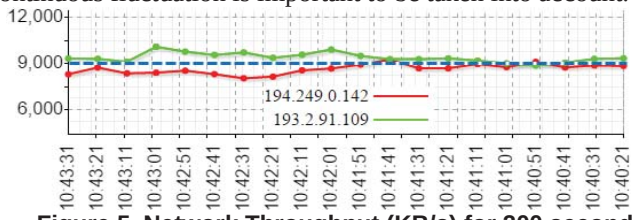


Figure 5. Network Throughput (KB/s) for 200 second monitoring window

Finally, regarding NPM, Figure 6 shows that $Host_i$ has better network performance quality with $Host_x$ compared to $Host_y$ during the last 10 intervals. Therefore, if $Host_i$ is connected to the $Host_y$, adaptation should occur and thus $Host_i$ will be connected to the $Host_x$ instead of $Host_y$.

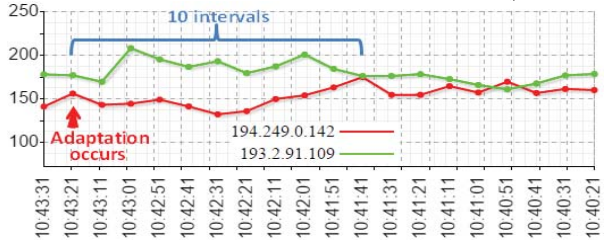


Figure 6. Network performance metric (NPM) for 200 second monitoring window

By employing only the last measurement explained above, this metric can have significant effect on the application performance and hence users' satisfaction; Cloud services for time-critical applications can automatically optimise the process of choosing the best possible application components, which are responsible for offering acceptable network QoS.

A challenge in designing a monitoring framework in the Cloud environment is ensuring that the overhead of the monitoring system is kept to the minimum [11]. The distributed nature of proposed monitoring framework quenches the runtime overhead of system to a number of Monitoring Probes running across different VMs. A detailed view on the resource consumption of the Monitoring Probe revealed that our approach is lightweight in terms of CPU and memory overhead. To confirm this, we applied the "top" tool which provides a dynamic real-time view of tasks currently being managed by the Linux kernel. Our running Monitoring Probe consumes only 0.3 percent of the whole CPU time and 3.1 percent of the whole memory usage in average.

Furthermore, comparing with the average network throughput of CC Server, the running Monitoring Probe consumes a small fraction of network bandwidth. To this end, we parsed the output of "nethogs" tool to estimate the bandwidth overhead introduced by our Monitoring Probe. We found out our Monitoring Probe transmits 1282944 bytes during 15 minutes, which means ~712 bytes per second for every DB Server in average.

Since the architecture includes a knowledge base, average "write" performance in milliseconds for the Fuseki backend implementation was calculated. The Fuseki server has one CPU 2397 MHz and 2GB total memory. During 15 minutes, 90 "write" queries were executed for each DB Server and the average query execution time was 3.93 ms.

6. Conclusion

In distributed time-critical Cloud applications, network-level features such as throughput and latency of packets travelling between application components directly affect user experience. Therefore, time-critical service providers must constantly monitor the network performance between their current servers running on different Cloud infrastructures, and other alternatives. In this way, preventing and predicting potential network performance drops related to the connections between the servers or possible overloads in the system will give more time to take action like dynamically changing connectivity topology among running components and switching from one server to another server to adjust the system in an anticipatory manner.

This research paper presented a lightweight network-based monitoring approach that is particularly suitable for autonomously adapting distributed time-critical Cloud applications. The lightweight feature for the implemented monitoring approach is a significant property in Cloud computing environments because of the necessity of being non-intrusive to the normal flows of application. The proposed solution is general and extensible, and it can be applied to any distributed Cloud application. The goal of the paper was to investigate network QoS properties that are especially important for the development of modern time-critical Cloud applications. We extend the current state-of-the-art by proposing a turnkey approach that not only monitors network QoS, but also stores the monitoring information, processes it, and integrates it with other system information for controlling the overall performance.

7. Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 643963 (SWITCH project: Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications).

8. References

[1] Ma, K., Sun, R., and Abraham, A., "Toward a lightweight framework for monitoring public clouds"; Proceedings of 4th International Conference on Computational Aspects of Social Networks (CASoN 2012), Brazil, Pp. 361—365, 2012.

[2] Anouari, T. and Haqiq, A., "Analysis of VoIP and Video Traffic over WiMAX Using Different Service Classes"; Journal of Mobile Multimedia, Vol. 9, No. 3&4, 2014.

[3] Sodangi, L. S., "Distributed Multimedia Applications in Quality of Service for Wireless Wide Area Network"; International Journal of Engineering Research and Technology (IJERT), Vol. 2, No. 10, Pp. 4088—4104, 2013.

[4] Li, A., Yang, X., Kandula, S., and Zhang, M., "Cloudcmp: comparing public cloud providers"; Proceedings of the ACM SIGCOMM conference on Internet measurement, 2010.

[5] Lampe, U., Wu, Q., Hans, R., Miede, A., and Steinmetz, R., "To Frag Or To Be Fragged - An Empirical Assessment of Latency in Cloud Gaming"; 3rd International Conference on Cloud Computing and Services Science, 2013.

[6] Samimi, A. F., McKinley, P. K., Sadjadi, S. M., Tang, C., Shapiro, J. K., and Zhou, Z., "Service Clouds: Distributed Infrastructure for Adaptive Communication Services"; IEEE Transactions on Network and Service Management, Vol. 4, No. 2, Pp. 84—95, 2007.

[7] Mohit, M., "A comprehensive solution to cloud traffic tribulations"; International Journal on Web Service Computing, Vol. 1, No. 2, Pp. 1—13, December 2010.

[8] Cervino, J., Rodriguez, P., Trajkovska, I., Mozo, A., and Salvachua, J., "Testing a Cloud Provider Network for Hybrid P2P and Cloud Streaming Architectures"; IEEE International Conference on Cloud Computing, Pp. 356—363, 2011.

[9] Chen, K. T., Chang, Y. C., Hsu, H. J., Chen, D. Y., Huang, C. Y., and Hsu, C. H., "On the quality of service of cloud gaming systems"; IEEE Transactions on Multimedia, Vol. 16, No. 2, Pp. 480—495, February 2014.

[10] Roda, C., Navarro, E., and Cuesta, C. E., "A comparative analysis of Linked Data tools to support architectural knowledge"; ISD2014 International Conference on Information Systems Development, 2014.

[11] Aceto, G., Botta, A., De Donato, W., and Pescape, A., "Cloud Monitoring: definitions, issues and future directions"; IEEE 1st International Conference on Cloud Networking (CLOUDNET), Pp. 63—67, November 2012.

OpenStackFT: Fault Tolerance in Open Source Cloud Computing Environment

H. P. Martins[1], **R. Spolon**[1], **N. G. Bachiega**[1], **R. S. Lobato, A. Manacero**[2], **M. A. Cavenaghi**[3]
[1]Departamento de Ciências da Computação, Universidade Estadual Paulista "Júlio de Mesquita Filho" Bauru, SP, Brasil
[2]Departamento de Ciências da Computação e Estatística, Universidade Estadual Paulista "Júlio de Mesquita Filho", São José do Rio Preto, SP, Brasil
[3]Humber Institute of Technology & Advanced Learning, The Business School, Toronto, ON

Abstract - *Cloud Computing is a set of features and services offered over the internet, delivered from data centers located around the world. As the Cloud Computing grows fast, the concern with the need of services offered increases, and the major challenge is to implement a fault-tolerant environment. The main issues of fault tolerance in Cloud Computing are fault detection and recovery. In order to combat such problems, many techniques are projected. Paid managers offer this kind of support, but the open source managers do not provide evidence to tolerate failures and leave users vulnerable to failures of the technology environment. This study presents the OpenStackFT, a fault-tolerant mechanism developed for the OpenStack manager. A redundancy mechanism was created in virtual machines instantiated in cloud nodes. If a node presents a transient or intermittent failure, the virtual machine will be stored on a backup node, waiting for the node to return from a failure. Experimental results show that the mechanism developed is viable and efficient because, right after a node has recovered from a failure, the virtual machine is not lost, thus becoming active again to the user.*

Keywords: *cloud computing; openstack; fault tolerance.*

1 Introduction

As there is a constant increase of computational use, problems like energetic demand and space in the data center are occurring all over the world. Many solutions are being projected to solve this kind of situation, among them is the Cloud Computing, term used initially by IBM in its white paper about technology in 2007 [1].

Cloud can be defined as a network environment based on the sharing of computing resources. Clouds are based on the internet and they try to make the complexity transparent to customers. Cloud Computing refers to applications (from hardware and software) delivered as services over the internet from data centers. Companies that provide clouds use virtualization technologies, combined with their abilities to provide computing resources through the network infrastructure [2].

In cloud environments, the concern is whether there will be high availability in the services offered by cloud managers. Many companies are migrating their services to cloud and choosing to implement their own cloud, using some open source managers such as the OpenStack.

Given the aforementioned context, there is a concern about having a fault-tolerant environment, if there is an environment failure, it means the environment is not providing the services properly to what it was designed for. If a distributed system is designed with a set of servers that communicate with each other and with customers, the inadequate supply of services means that the servers are not doing what they should. However, the fault is not always on the server that presents it, since if the server depends on other servers to provide its services, the error may have to be looked in another place [3]. This study shows the mechanism developed for the OpenStack manager.

2 Related Research

OpenStack has information about its high availability and fault tolerance on its support website, but no information on how to implement the solutions is displayed or demonstrated. The company Rackspace uses OpenStack[1] as a solution. It informs it has implemented fault tolerance and high availability, but it does not display information about how they are implemented.

Another highlighted open source is the Apache CloudStack[2], a platform that gathers computing resources for the construction of infrastructure as a service. CloudStack has some high availability characteristics such as the Management Server, which can make its implementation by itself in many nodes, where servers are balanced between data centers. The database MySQL can be set to use replication, preventing a failure situation in case of a data loss.

[1] https://www.openstack.org/
[2] http://cloudstack.apache.org/index.html

Paid clouds servers such as VMware, Amazon and Citrix have implemented fault tolerance solutions in their distributions. This research was designed from these paid managers, which are presented in this section.

The vSphere product of the VMware provides availability for a company entire virtual environment, minimizing unplanned downtime by restarting the virtual machine, providing a level of high availability [4]. A number of 5 products of high availability resources are presented such as: High Availability, Data Protection, App HÁ, Fault Tolerance and the Replication.

The AWS - Amazon Web Services (Amazon Web Services) of Amazon is one of the tools and resources that allows the creation of fault-tolerant systems that are reliable and demand little human intervention. The Amazon Services: Elastic Compute Cloud (EC2) and the Amazon Elastic Block Store (EBS) provide resources such as snapshots and availability zones, which fault-tolerant systems are highly available [5].

The Citrix XenServer is a complete virtual infrastructure solution, with a management interface, live migration resources, and tools to convert workloads from a physical environment to a virtual one. It is possible to create and manage virtual machines that can be executed from a management interface, it allows the active virtual machines to be transferred to a new physical host without the interruption of its applications, generating inactivity [6].

3 Fault tolerance Mechanism

As a distributed system environment, Cloud Computing is susceptible to faults, and fault tolerance techniques must be used to improve environment availability. A vulnerability considered critical to cloud functioning was chosen for this research. The vulnerability chosen was the hardware fault-tolerance problem in the nodes. Thus, the fault-tolerance mechanism labeled as OpenStackFT was created.

For the implementation of OpenStackFT, a node was used with the Linux OpenFiler operating system, NAS (Network Attached Storage) and SAN (Storage Area Network) for open source storage appliance, which provides data storage device access via network.

The OpenFiler3 was chosen because it has support for volume-based partitioning (iSCSI - Internet Small Computer System Interface) and management of settings via the Web. Another reason to choose the iSCSI was the test performed with other tools that did not work in OpenStack, these tools were the DRBD (Distributed Replicated Block Device), NFS (Network File System) and RSync.

The iSCSI is used to connect storage devices through a network via TCP/IP. It can be used through a local area network (LAN), a wide area network (WAN) or through the Internet. The iSCSI devices include disks, tapes, CD-ROMs and other storage devices in another computer in the network which can be connected. Sometimes, these storage devices are part of a storage area network (SAN). In the relation between the computer and the storage device, the computer is labeled as the Initiator, because it initiates the connection with the device, which is labeled as Target..

Figure 1 shows the OpenStackFT environment. In this environment, it is possible to observe the OpenFiler, responsible for the storage of the instances initiated by the OpenStack.

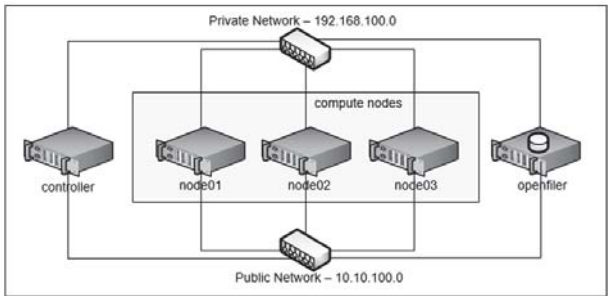


Figure 1. OpenStackTF environment.

Figure 2 shows the flow of a virtual machine when instanced by Horizon. When the virtual machine is instanced in an available node, the iSCSI protocol that is configured and active in the node replicates in the OpenFiler, creating a redundancy of the virtual machine in a directory available to the node. In the example of Figure 2, the virtual machine instanced by *node01* will be stored in the OpenFiler /dev/sdb directory. This virtual machine that was sent to OpenFiler is constantly updated by iSCSI, until a fault occurs in the node. If a fault occurs in the node, the virtual machine turns to stand by in the OpenFiler until the node recovers itself from the fault.

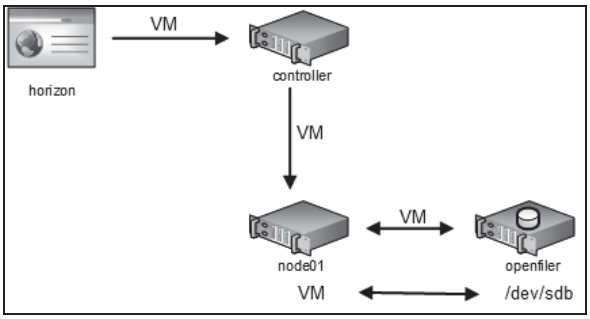


Figure 2. Virtual Machine (VM) flow in the environment.

It is emphasized that the OpenStackFT test environment has limitations, but the same environment can be reproduced in more robust hardware environments.

4 Tests and Results

The tests were carried out based on the requirements established by Tanenbaum and Steen [3], who described a fault-tolerant environment, implementing availability, reliability, safety and maintainability.

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

The tests were intensified in the nodes, since, as mentioned above, a user creates a virtual machine which will be allocated on a node, and if this node has problems, the virtual machine that is connected is lost and must be discarded by the cloud administrator. This failure makes the user lose the virtual machine, having to create a new one.

To simulate a node failure, three types of tests were carried out: unplugging the node's network cable, restarting the node, and disconnecting the node by removing it from the outlet. These failure tests are classified as transient or intermittent. For the tests, the *ping*[4] tool was used to test if the node did not respond and if it showed the expected failure. In addition to using the ping tool on two computers, the researcher personally made sure that the machine was off the network or powered off.

The failure simulation tests were timed to verify how long it takes for the virtual machine to be available for the user again. To accomplish this control of time, the controller operating system clock was used. The time control was performed as follows:

- Test 1: In the event of a network failure, the network cable was removed and a confirmation by the ping that the machine did not respond was waited. Then the network cable was plugged in and the timer was initiated. When the status of the virtual machine became "Active", the timer was stopped.
- Test 2: In the event the machine restarts, the command "shutdown -r now" was used, at the time the command was executed, the timer was initiated, and this was ceased at the time the status of the virtual machine became "Active".
- Test 3: In the event the machine abruptly turned off, the power cable was removed and plugged in soon after. At the time the machine was turned off, the timer was initiated, and it was ceased when the status of the virtual machine became "Active".

To ensure fairness and efficiency in the data collection, a number of 30 tests were performed in the environment; the results are shown in Table 1. The average time observed was 3 seconds for Test 1, 95 seconds for Test 2 and 120 seconds for Test 3, with a standard deviation of 0.00 for Test 1, 0.79 for Test 2 and 1.26 for Test 3.

In the following sections, two scenarios of tests are presented. In the first scenario, the failure tests were performed in the initial environment; to perform the same test afterwards with the active settings. The test was conducted with only one virtual machine to verify the efficiency of the solution. In the second scenario, tests were performed with the largest number of machines possible. This scenario was performed to verify if the solution would be as effective as it had been in the first scenario.

4.1 First Test Scenario

In this stage, it was taken in consideration only if the virtual machine was not lost after the recovery of a node fault.

Figure 3 shows that the virtual machine presents the Error status. Some node faults were simulated so that the Error status appeared. It is worth to highlight that after a node recovers itself from a fault, the virtual machine can no longer be re-used and it was necessary to delete the instance.

Figure 3. Simulating a node fault.

Next, the fault-tolerance configuration was performed using the OpenFiler. As shown in Figure 4, a new virtual machine was initiated, but this time, the OpenStackFT was active.

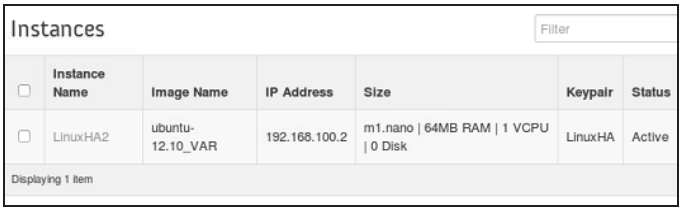


Figure 4. Initiating a virtual machine with a fault tolerance solution.

According to Figure 5, the machine status presented Error because node faults were simulated.

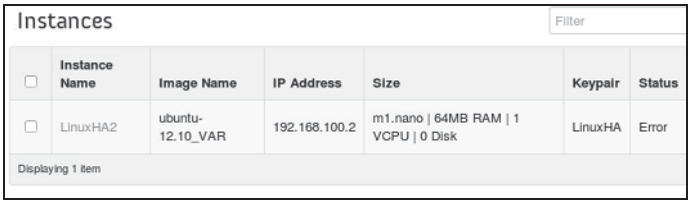


Figure 5. Simulating a node fault with a fault tolerance solution.

After replacing the network cable or reinitializing the node, the machine status became Active again. In this case, there was no need to delete the instance as shown in Figure 6.

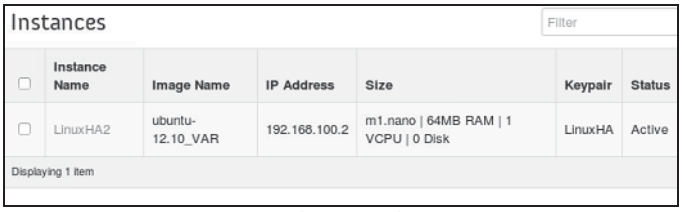


Figure 6. Virtual machine active after a node fault.

[4] Ping is a utility to test connectivity between devices. (CISCO, 2014).

In all performed tests, the behavior observed was always the same. The instance became active after the node recovered itself from the failure.

4.2 Second Test Scenario

The second test scenario was performed in a similar way as the first one, but the test was executed in the maximum of virtual machines available for the environment research. Figure 7 shows the entry of 10 virtual machines.

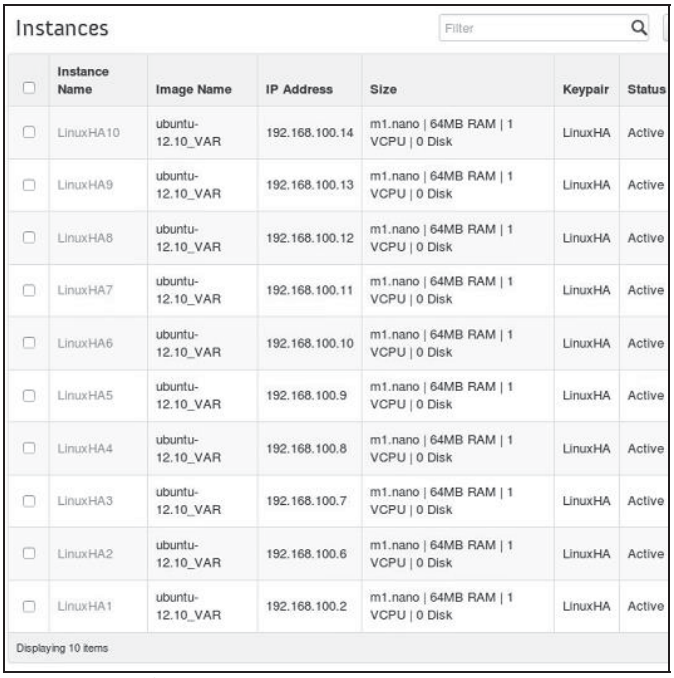


Figure 7. Entry of 10 virtual machines.

After activating the 10 virtual machines in the environment, the same procedures were performed as in the first scenario. All virtual machines obtained a successful result after the node returned from the fault. In the case of the failure test, the three types of node faults were simulated.

In all the performed tests in the second scenario, the behavior was always the same. Therefore, all virtual machines became active again after the nodes recovered from the faults.

4.3 Results Evaluation

After the implementation and the tests performed in the OpenStackFT, it can be concluded that the solution found solved a fault in the OpenStack: reactivating node instanced virtual machines. Thus, the users will not be affected if some hardware fault occurs in the node in which this virtual machine is linked to. The cloud managers that have implemented the OpenStackFT will also be benefited because they will not need to recreate new virtual machines for the users in case of faults in the nodes.

According to the collected results, the requirements of reliability shown by Tanenbaum and Steen [3] were reached: the availability requirement was reached, since the virtual machine becomes available to the user after a node fault. The reliability requirement was reached, since the virtual machine recovered itself and stood active after the node fault, offering the user the reliability of not losing the virtual machine. The security requirement was reached, since the node was inoperative for some time and, nevertheless, it started functioning normally. Besides, the virtual machines active at the time of the fault were not damaged. The maintainability requirement was reached, since the node that presented a fault was recovered.

Table 1 presents the results obtained with the tests performed in the OpenStackFT. The comparative presents the average and the time standard deviation in seconds that each scenario took to activate the virtual machines after the three failure tests. By observing this table, it is also possible to verify that the higher the number of virtual machines that need to be reactivated, the higher will be the average time that the environment will take to become fully active.

TABLE I. TIME COMPARATIVE

	Scenario 1	**± SD**	**Scenario 2**	**± SD**
Test 1	1 second	0.00	3 seconds	0.00
Test 2	35 seconds	0.83	95 seconds	0.79
Test 3	55 seconds	0.83	120 seconds	1.26

5 Considerations and Conclusions

The difficulties encountered by a user to ensure the high availability of technology services, primarily in open source environments, was an important factor for the development of the OpenStackFT mechanism.

This work is relevant to users and cloud administrators who intend to implement a open source private cloud.. In order that the administrator is able to reproduce the proposed mechanism, it is, first, necessary to have the OpenStack and subsequently implement the OpenStakTF.

6 References

[1] Z. He and Y. He, "Analysis on the security of cloud computing". Proc. Spie, Qingdao, China, n., 2011, p.7752-775204.

[2] F. SABAHI, "Cloud computing security threats and responses". Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on May 2011.

[3] A.S. TANENBAUM and M.V. STEEN, "Distributed Systems: Principles and Paradigms". 2.ed, Pearson Prentice Hall, 2007.

[4] VMWARE, Availability, Accessed on: <http://www.vmware.com/>, February 2016.

[5] AMAZON, Architecture Center of the AWS, Accessed on: <http://aws.amazon.com/pt/architecture/>, February 2016.

[6] CITRIX, Accessed on: <http://www.citrix.com>, February 2016.

ISBN: 1-60132-444-8, CSREA Press ©

SESSION

PARALLEL AND SCALABLE ALGORITHMS AND SYSTEMS, HPC, AND COMPUTATIONAL SCIENCE

Chair(s)

TBA

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

Exploration of MPI-backed Parallelization for Tableau-based Description Logic Reasoning

M. Hossain and W. MacCaull
Department of Mathematics, Statistics and Computer Science
St. Francis Xavier University, Antigonish, NS, Canada

Abstract— *Description logic (DL) reasoning systems do not scale to the requirements of the rapidly growing amount of data. Although a lot of optimization techniques have been developed over the last decades, reasoning performance is still a bottleneck for users. Moreover, most modern reasoners consist of programs that run on a single machine. When the ontology is very large and complex, the computational resources of a single machine are not enough. Therefore, in order to achieve the vision of the semantic web, developing highly scalable and efficient ontology reasoner is crucial. In this work, we have investigated the potential of improving performance of a tableau-based reasoner via parallelization. In order to achieve practical scalability via parallelization, we have developed a parallel model based on the universal manager-worker model to check the consistency of a knowledge base. We have also implemented this model in a distributed memory environment using a Java message passing library for the DL $\mathcal{ALC}$.*

Keywords: parallel reasoning; description logic; high performance computing; message passing interface.

1. Introduction

With the explosive growth of data in the semantic web (SW), large and complex knowledge bases (KBs) are emerging day by day. At present, reasoning over such KBs has become one of the most challenging problems in SW applications. Although a good number of optimization techniques have been developed in the past decades, DL reasoning systems do not scale efficiently to deal with the rapid growth of data. Most optimization techniques have been investigated for sequential reasoners, and scalability (i.e., the ability to use additional computational resources to process larger KBs) of a sequential reasoner is limited by the physical resources of a single machine. Furthermore, most state-of-the-art reasoning systems are based on tableau algorithms and the high computational complexity (e.g., KB satisfiability for $\mathcal{SHIQ}$ is ExpTime-complete) of tableau algorithms makes the process even more difficult. Therefore, in order to support the vision of the SW, developing a highly scalable and efficient ontology reasoner is crucial.

In the last decade, a few attempts have been made to parallelize DL reasoning, but most target a single machine (e.g., thread-level parallelism on multi-core systems) [1], [2], [3], [4]. The performance gain that can be achieved by this approach is limited by the number of available cores. Typically, the number of cores in such a machine is not higher than eight [5]. Moreover, for large or distributed ontologies, thread-based strategies are not suitable because they target a single machine. So, in order to reduce the processing time via parallelization, reasoning engines need to distribute their workload into different computational units. A distributed approach is potentially more scalable than a single machine approach because it can be scaled in two dimensions, namely enhancing the hardware performance of each node and increasing the number of nodes in the system.

The core function of a tableau based reasoner is checking the consistency of a KB, i.e., determining whether a given KB has a model. Therefore, we focused on parallelizing the consistency checking procedure. In this work, we developed a parallel consistency checking algorithm based on a well-known programming paradigm, the Message Passing Interface (MPI) [6]. We showed that independent tableau branches can be processed concurrently on independent processes. Our algorithm is based on a universal model namely manager-worker model. We implemented this parallel consistency checking algorithm for the DL $\mathcal{ALC}$ using MPJ Express [7], an open source Java message passing library, and discussed our initial results by executing in both multi-core processors (shared-memory) and computer clusters (distributed memory). As far as we are aware, this is the first attempt to parallelize consistency checking in a distributed memory environment using MPI. This work is also significant to the high performance computing (HPC) community because it attempts to close the gap between Java and MPI.

The rest of this paper is organized as follows: Section 2 outlines a few related works, Section 3 describes the syntax and semantics, and the tableau algorithm for the DL $\mathcal{ALC}$, Section 4 presents the parallel model to check the consistency of a KB, Section 5 describes the implementation and evaluation of this model for the DL $\mathcal{ALC}$ by means of MPI, and finally, Section 6 summarizes the paper with future work.

2. Related Work

The DL community has already made some notable efforts on adopting the HPC paradigm in DL reasoning. A few of

them are related to our work; these are reported below.

Liebig and Muller [2] reported a parallel $\mathcal{SHN}$ reasoner named UUPR (*Ulm University Parallel Reasoner*). UUPR parallelizes the tableau algorithm itself by applying concurrent computation on *disjunction* and the *at most number restriction* rules. Although various optimization techniques have been adopted in UUPR, a few significant optimizations, e.g., GCI absorption, dependency directed backtracking, are absent. Moreover, UUPR is implemented as a shared memory program using the *boost.Threads* library and hence the speed up gain is limited by the physical architecture of a single machine.

Bao et al. [8] presented a distributed tableau algorithm using MapReduce [9]. Although MapReduce has proved to be efficient for large datasets on certain kinds of distributable problems, this technique requires a considerable re-thinking of the tableau algorithms in order to conform to the Map and Reduce steps. Therefore, we did not explore this approach.

Wu and Haarslev [10] developed a parallel tableau-based DL reasoner named *Deslog* for the DL $\mathcal{ALC}$. Deslog is a shared-memory parallel reasoner for TBox classification. Several optimization techniques were incorporated into Deslog, thus leading to good efficiency for TBox classification. However, the speed up that can be gained is limited by the available cores in the shared-memory environment.

The recently developed LarKC [11] platform provides a high level of flexibility, performance, and scalability for reasoning over large-scale semantic data sets [12]. This platform addresses the limitations of current semantic reasoning engines and enables reasoning with big data by distributing computation among nodes. For the most part, there are two proposed approaches in the literature: rule partitioning and data partitioning. In [5], Cheptsov described an MPI based approach for implementing parallel semantic web applications and evaluating the performance of random indexing over large text volumes on the LarKC platform.

Faddoul and MacCaull [13] outlined a fork/join parallel framework for handling non-determinism arising from algebraic tableau reasoning. This parallel framework allows the execution of non-deterministic rules on independent cores only (not on independent processes). To work with an algebraic reasoning component, a standard tableau calculus needs to be modified and extended.

An observation is that applying thread based strategies such as multi-threading in multi-cored processor is the easiest and simplest way to achieve the high performance [10]. However, speed up gained via thread-level parallelism is limited by the currently existing computer architecture. On the other hand, the process-based strategies, such as MPI, allow one to execute applications in distributed compute architectures such as compute clusters, grid systems, etc. The MPI is a well-known programming paradigm intended for programming in distributed memory environments. It is a high-level library for sending and receiving messages that is commonly used in HPC applications to abstract the underlying networking details. In this work, we introduce and discuss solutions for the implementation of a tableau algorithm with MPI [14].

3. Preliminaries

DLs are a family of knowledge representation formalisms suitable for representing the terminological knowledge in a wide range of applications. They can be used to represent the knowledge of an application domain in a structured and formally well-defined way. A KB of a typical DL system consists of a *TBox* and an *ABox*. The TBox introduces the terminology, i.e., the vocabulary of an application domain, while the ABox contains assertions about named individuals in terms of this vocabulary [15]. A DL system sets up KBs to do reasoning and manipulation of content. In this section we introduce the syntax, semantics, and the tableau algorithm for the DL $\mathcal{ALC}$.

3.1 Syntax and Semantics of $\mathcal{ALC}$

$\mathcal{ALC}$ language is the smallest but relatively expressive propositionally closed DL. It is constructed from atomic concepts, atomic roles, $\sqcap$ (conjunction), $\sqcup$ (disjunction), $\neg$ (negation), $\forall R.C$ (value restriction), $\exists R.C$ (existential restriction).

Let N_C and N_R be non-empty and pair-wise disjoint sets of concept names and role names respectively. Let N_I be a set of all individual names. Below A is used to denote an atomic concept ($A \in N_C$), R is used to denote an atomic role ($R \in N_R$). Concept descriptions in $\mathcal{ALC}$ are formed according to the syntax rule in (1), given in BNF form; where C, D are $\mathcal{ALC}$ concepts and $\top$ (everything) and $\bot$ (nothing) are the *universal concept* and *bottom concept*, respectively.

$$C, D \rightarrow A \mid \top \mid \bot \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \forall R.C \mid \exists R.C \quad (1)$$

The formal definition of semantics of $\mathcal{ALC}$ is given by means of an interpretation $\mathcal{I}$. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a mapping function $.^{\mathcal{I}}$, called the interpretation function of $\mathcal{I}$, that maps:

- every individual name $a \in N_I$ to an element, $a^{\mathcal{I}}$, of $\Delta^{\mathcal{I}}$ $(i.e., a^{\mathcal{I}} \in \Delta^{\mathcal{I}})$
- every concept name $A \in N_C$ to a subset, $A^{\mathcal{I}}$, of $\Delta^{\mathcal{I}}$ $(i.e., A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}})$
- every role name $R \in N_R$ to a subset, $R^{\mathcal{I}}$, of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ $(i.e., R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$.

The interpretation function is extended to satisfy $\mathcal{ALC}$-concept descriptions as follows:
$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$; $\bot^{\mathcal{I}} = \emptyset$; $(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$;
$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$; $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$;
$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : \langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$;
$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : \langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$.

An $\mathcal{ALC}$ KB is a finite set of axioms formed by concepts, roles and individuals. A concept assertion is an axiom of

the form $C(a)$ (a is an instance of C) and a role assertion is an axiom of the form $R(a,b)$ (a is related to b via R), where a, b are individuals and C and R are concept and role, respectively. A concept inclusion is an axiom of the form $C \sqsubseteq D$ means that concept D is more general than concept C. An ABox is set of role assertions and concept assertions; a TBox is a set of concept inclusions.

An interpretation $\mathcal{I}$ satisfies a concept assertion $C(a)$, denoted by $\mathcal{I} \models C(a)$, if and only if (iff) $a^{\mathcal{I}} \in C^{\mathcal{I}}$; it satisfies a role assertion $R(a,b)$, denoted by $\mathcal{I} \models R(a,b)$, iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$; it satisfies an ABox, $\mathcal{A}$, (written, $\mathcal{I} \models \mathcal{A}$) iff $\mathcal{I}$ satisfies every assertion in $\mathcal{A}$. If $\mathcal{I}$ satisfies $\mathcal{A}$, then $\mathcal{I}$ is called a model of $\mathcal{A}$. An interpretation $\mathcal{I}$ satisfies a concept inclusion $C \sqsubseteq D$, denoted by $\mathcal{I} \models C \sqsubseteq D$, iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and it satisfies a TBox, $\mathcal{T}$, (written, $\mathcal{I} \models \mathcal{T}$) iff $\mathcal{I}$ satisfies every inclusion in $\mathcal{T}$. If $\mathcal{I}$ satisfies $\mathcal{T}$ then $\mathcal{I}$ is called a model of $\mathcal{T}$.

An interpretation $\mathcal{I}$ is a model of a KB, $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$, denoted by $\mathcal{I} \models \mathcal{K}$, iff $\mathcal{I}$ is a model of both $\mathcal{T}$ and $\mathcal{A}$. A concept C is satisfiable with respect to (w.r.t.) a TBox $\mathcal{T}$ iff there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. An ABox $\mathcal{A}$ is consistent w.r.t. a TBox $\mathcal{T}$, if there is an interpretation that is a model of both $\mathcal{A}$ and $\mathcal{T}$; $\mathcal{A}$ is inconsistent otherwise.

3.2 Tableau Algorithm

State of the art DL systems typically use tableau algorithms to decide satisfiability (consistency) of a KB. The standard tableau algorithm [16] generally contains the following main elements:

- A *completion forest* (called *tableau*) that represents a model of the DL language; such a completion forest typically has the tree model property.
- A set of tableau *expansion rules* to construct a completion forest.
- A set of *blocking rules* to guarantee termination.
- A set of *clash conditions* to detect logical contradictions.

For an $\mathcal{ALC}$ KB, $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, the algorithm will construct a model for both $\mathcal{T}$ and $\mathcal{A}$ to check the consistency of $\mathcal{K}$. If one such model (i.e., a completion forest) is found, $\mathcal{K}$ is consistent, otherwise $\mathcal{K}$ is inconsistent.

In order to work efficiently, it is necessary to transform a KB into the Negation Normal Form (NNF), i.e., the negation only occurs in front of concept names. A KB is in NNF if all concept descriptions in it are in NNF. Each concept description can be transformed to NNF by pushing negations inwards using the following equivalences:
$\neg\neg C \equiv C$; $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$; $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$; $\neg\exists R.C \equiv \forall R.\neg C$; $\neg\forall R.C \equiv \exists R.\neg C$.

Reasoning w.r.t. a TBox $\mathcal{T}$ can be reduced to reasoning w.r.t. an empty TBox by a process called *internalization* [15]. Given a $\mathcal{T}$, a concept $C_{\mathcal{T}}$ is defined as

$$C_{\mathcal{T}} := \prod_{C_i \sqsubseteq D_i \in \mathcal{T}} (\neg C_i \sqcup D_i).$$

Any individual x in any model of $\mathcal{T}$ will be an instance of $C_{\mathcal{T}}$ in that model [16].

Consistency checking is one of the main inference problems to which all other inferences can be reduced [15]. The main idea of tableau-based approaches for deciding the consistency of an ABox is as follows: the algorithm starts with the input ABox, $\mathcal{A}$ and applies consistency preserving expansion rules (e.g., the expansion rules for $\mathcal{ALC}$-ABoxes are presented in Table 1) until no more rules are applicable (the tableau is *complete*) or an obvious contradiction (called a *clash*) is found. If a complete and clash-free tableau is obtained, $\mathcal{A}$ is consistent; otherwise it is inconsistent. For comprehensive background on tableau calculus, the reader is referred to [15], [17], [18].

Table 1: $\mathcal{ALC}$-tableau expansion rules.

$\sqcap$-rule	**if** $C_1 \sqcap C_2(x) \in \mathcal{A}$, and $\{C_1(x), C_2(x)\} \nsubseteq \mathcal{A}$, **then** $\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.
$\sqcup$-rule	**if** $C_1 \sqcup C_2(x) \in \mathcal{A}$, and $\{C_1(x), C_2(x)\} \cap \mathcal{A} = \emptyset$, **then** $\mathcal{A}' := \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{C_2(x)\}$.
$\exists$-rule	**if** $\exists R.C(x) \in \mathcal{A}$ and there is no y such that $\{C(y), R(x,y)\} \in \mathcal{A}$, **then** $\mathcal{A}' := \mathcal{A} \cup \{C(z), R(x,z)\}$ such that z is a fresh individual.
$\forall$-rule	**if** $\{\forall R.C(x), R(x,y)\} \in \mathcal{A}$ and $C(y)\} \notin \mathcal{A}$, **then** $\mathcal{A}' := \mathcal{A} \cup \{C(y)\}$.

4. Parallelization

4.1 Parallel Consistency Checking

Tableau algorithms are amenable to parallelization due to the existence of inherently non-deterministic rules (e.g., disjunctions rule, qualified cardinality restriction rule, and choose rule for the DL $\mathcal{SHIQ}$). The application of a non-deterministic rule yields multiple alternatives, which can be treated as different possible ABoxes to continue reasoning with. Since there is no dependency between the alternatives, they can be processed concurrently. For example, if we have an *ABox*, $\mathcal{A} = \{a : C \sqcup D\}$, then the disjunction rule generates two *ABoxes* $\mathcal{A}_1 = \mathcal{A} \cup \{a : C\}$ and $\mathcal{A}_2 = \mathcal{A} \cup \{a : D\}$, which are independent, i.e., the consistency checking of one ABox does not depend on others.

In order to work efficiently, all tableau expansion rules are categorized into two main groups: *deterministic* and *non-deterministic*. If a *deterministic* rule is applicable then the original ABox, $\mathcal{A}$, is transformed into a new ABox, $\mathcal{A}_1$. Then $\mathcal{A}$ is consistent if the expanded ABox, $\mathcal{A}_1$, is. On the other hand, if a *non-deterministic* rule is applicable, then the original ABox, $\mathcal{A}$, is transformed to a set of new ABoxes, $\mathcal{S} = \{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k\}$, instead of a single ABox. Then $\mathcal{A}$

is consistent if there is some i, $1 \leq i \leq k$, such that $\mathcal{A}_i$ is consistent. Typically, consistency checking is performed on the set of ABoxes sequentially, i.e., one after another. Since the ABoxes $\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k\}$ are independent, it is possible to perform the consistency checking of these ABoxes concurrently.

4.2 Manager-Worker Model

We propose an MPI-backed parallel algorithm for checking the consistency of a KB. Our algorithm is based on one of the most universal *manager-worker* or *task parallelism* approaches. The manager-worker pattern is a variant of the *master-slave* pattern where node-0 is the manager (master) and all other nodes are workers (slave nodes). The variation is based on the fact that components of this pattern are proactive rather than reactive [19]. Each processing unit performs the same operations simultaneously and independently of the processing activity of other units. The manager and worker algorithms are presented below in Algorithm 1 and Algorithm 2, respectively.

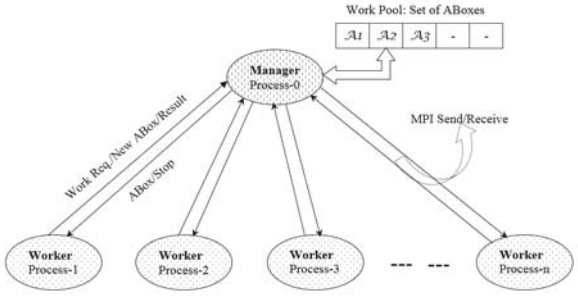


Fig. 1: Manager-worker model.

In manager-worker model, the manager maintains a work pool, a set of work items, and upon request the manager sends the next available work item to a worker (Lines 15–17, Algorithm 1). Each worker processes one work item at a time and, when finished, requests the manager for a new work item. This continues until there are no work items left. When all work items are complete, the manager tells the workers to stop (Lines 27–30, Algorithm 1). Figure 1 shows the manager-worker parallel computation model. Here, the manager executes a different algorithm from that of the worker. Though manager and worker execute different algorithms, we combine both manager and worker routines into a single program which is more convenient, efficient and also supported by all implementations of MPI.

Our algorithm follows a self-scheduling approach where the manager maintains a work pool. Each work item is an ABox, which consists of a set of concepts and expansion rules. The consistency checking algorithm for an ABox is provided in Algorithm 3. When a worker receives an ABox (Line 8, Algorithm 2), it applies consistency preserving expansion rules until no more rules are applicable or a

Algorithm 1 mpiOWL manager

Let n be the number of workers.
Let $\boldsymbol{\mathcal{W}} = \{\mathcal{W}_1, \mathcal{W}_2, ..., \mathcal{W}_n\}$ be the set of participating workers.
Let $\boldsymbol{\mathcal{A}} = \{\mathcal{A}_1, \mathcal{A}_2, ...\}$ be the finite set of ABoxes.
Let $\boldsymbol{closed\mathcal{A}} = \{\mathcal{A}_1, \mathcal{A}_2, ...\}$ be the finite set of $closed$ ABoxes.
Let $\boldsymbol{\mathcal{D}} = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_n\}$ be the set of assigned ABoxes and $\mathcal{D}_j \in \boldsymbol{\mathcal{A}}$ where $\mathcal{D}_j$ has been assigned to the worker $\mathcal{W}_j, 1 \leq j \leq n$.

Data: Input ABox, $\mathcal{A}$
Initialize:
1: $ptr \leftarrow 1$
2: $\boldsymbol{\mathcal{A}} \leftarrow \{\mathcal{A}\}$
3: $\boldsymbol{closed\mathcal{A}} \leftarrow \emptyset$
4: $isClosed \leftarrow$ **false**
5: $isComplete \leftarrow$ **false**
Require: $|\boldsymbol{\mathcal{W}}| \neq 0$
Ensure: $isComplete =$ **true** $\vee\ isClosed =$ **true**
6: **while** $\neg isComplete\ \&\ \neg isClosed$ **do**
7: Receive a $message$ from a worker $\mathcal{W}_j$
8: **if** $message$ is a state request **then**
9: **if** $ptr \leq |\boldsymbol{\mathcal{A}}|$ **then**
10: Send the state *CHECK* to the worker $\mathcal{W}_j$
11: **else**
12: Send the state *WAIT* to the worker $\mathcal{W}_j$
13: **end if**
14: **else if** $message$ is an ABox request **then**
15: Send ABox, $\mathcal{A}_{ptr}$, to the worker $\mathcal{W}_j$
16: $\mathcal{D}_j \leftarrow \mathcal{A}_{ptr}$
17: $ptr \leftarrow ptr + 1$
18: **else if** $message$ is a newly generated ABox **then**
19: $\boldsymbol{\mathcal{A}} \leftarrow \boldsymbol{\mathcal{A}} \cup \{message\}$
20: **else if** $message$ is a *CLASH* report **then**
21: $\boldsymbol{closed\mathcal{A}} \leftarrow \boldsymbol{closed\mathcal{A}} \cup \{\mathcal{D}_j\}$
22: **else if** $message$ is a *COMPLETE* status **then**
23: $isComplete \leftarrow$ **true**
24: **end if**
25: $isClosed \leftarrow (|\boldsymbol{\mathcal{A}}| = |\boldsymbol{closed\mathcal{A}}|)$
26: **end while**
27: **for all** $\mathcal{W} \in \boldsymbol{\mathcal{W}}$ **do**
28: Receive a $message$ form the worker $\mathcal{W}_j$
29: Send the state *STOP* to the worker $\mathcal{W}_j$
30: **end for**
31: **if** $isComplete =$ **true** **then**
32: $\mathcal{A}$ is *Consistent*.
33: **else**
34: $\mathcal{A}$ is *Inconsistent*.
35: **end if**

Algorithm 2 mpiOWL worker

```
1: Send a request to the manager
2: currentState ← Receive the state from the manager
3: while currentState ≠ STOP do
4:     if currentState = WAIT then
5:         Wait(timeOut)
6:         Send a request to the manager
7:     else
8:         A ← Receive an ABox from the manager
9:         if CONSISTENT(A) then
10:            Send the state COMPLETE to the manager
11:        else
12:            Send the state CLASH to the manager
13:        end if
14:    end if
15:    currentState ← Receive the state from the manager
16: end while
```

Algorithm 3 consistency checking

Data: Input ABox, $\mathcal{A}$
Result: True or False.

```
1: procedure CONSISTENT(A)
2:     if A is closed then
3:         return false
4:     end if
5:     if A is complete then
6:         return true
7:     end if
8:     Find a rule that is applicable on A.
9:     if a deterministic rule is applicable then
10:        A_1 ←Apply the expansion rule on A.
11:    else if a non-deterministic rule is applicable then
12:        {A_1, A_2, ..., A_k} ←Apply the expansion rule on A.
13:        for all i such that 2 ≤ i ≤ k do
14:            Send A_i to the manager
15:        end for
16:    end if
17:    return CONSISTENT(A_1)
18: end procedure
```

clash is found (Lines 2–7, Algorithm 3). Whenever a non-deterministic rule is applied, i.e., more than one ABox is produced (Line 12, Algorithm 3), it keeps one ABox to itself and sends the remaining ABoxes to the manager (Line 14, Algorithm 3). If the manager receives a new ABox from a worker, it adds the new ABox to the work pool (Lines 18 and 19, Algorithm 1). After performing consistency checking on a given ABox, a worker sends the result to the manager and asks for another ABox (Lines 10 and 12, Algorithm 2). If there are any available ABoxes, the manager assigns one of them to the worker, otherwise the manager tells the worker to stop. The manager stops when there is an ABox that is *complete* or all the ABoxes are *closed* (contains clash).

5. Experiments

5.1 MPI Libraries in Java

MPI is a standardized and portable message-passing system that follows a process-oriented parallel computing paradigm. It is based on a Single Program Multiple Data (SPMD) execution model. Although there is no official MPI binding for Java, there exist several projects (e.g., mpiJava, JavaMPI, MPIJ) that provide required functions with different degrees of success and compatibility [7]. Most of these projects are prototype implementations, without any maintenance. Currently, the most successful ones in terms of uptake by the HPC community are mpiJava and MPJ Express, which we now present.

mpiJava: The mpiJava[1] is an object-oriented Java interface to the standard MPI, developed as part of the HPJava project in late 1997 by Carpenter et al. [20]. The implementation of mpiJava is through Java Native Interface (JNI) wrappers to native MPI software. It takes an existing native MPI implementation and provides Java wrappers through the JNI which is a mechanism that allows the application programmer to call native subroutines and libraries (written in other languages such as C, C++) from Java and vice versa. In mpiJava, every MPI process or node corresponds to a single JVM, running on the same host computer. The mpiJava is implemented on top of a native MPI and most native implementations of MPI are not thread-safe. Therefore, it is not possible to perform MPI operations concurrently for more than one thread in a single JVM.

MPJ Express: MPJ Express[2] is a message passing library that can be used by application developers to execute their parallel Java applications on compute clusters or network of computers [7]. Although MPJ Express is designed for distributed memory environments like networks of computers or clusters, it can execute parallel programs efficiently in a multi-core processor with a shared memory environment. The MPJ Express software can be configured in two ways: (1) multi-core configuration and (2) cluster configuration. The multi-core configuration is used by the developers who want to execute their parallel Java applications on multi-core processors. The cluster configuration is used to execute

[1] http://www.hpjava.org/mpiJava.html

[2] http://mpj-express.org/

parallel Java applications on distributed memory platforms including clusters and network of computers.

Both mpiJava and MPJ Express provide message passing functionality to the Java programmers. But mpiJava can incur a noticeable overhead for large messages and also presents some portability and instability issues. There is no recent update in mpiJava and it only supports some native MPI implementations. On the other hand, MPJ Express is maintained regularly and most recent version was released on April 18, 2015. We therefore choose to use MPJ Express to implement our parallel consistency checking task.

5.2 Parallel Implementation for the DL $\mathcal{ALC}$

There is an important property in $\mathcal{ALC}$ tableau algorithm which makes the algorithm pleasingly parallel. The whole knowledge necessary for node expansion or for clash detection is contained in a given node. Therefore, there is no information exchange between nodes belonging to different branches. As a result, branches of the tableau can be constructed independently of one another. The parallelization strategy described in this paper takes advantage of this property. The standard tableau expansion rules for the DL $\mathcal{ALC}$ are presented in Table 1. Note that only the $\sqcup$-rule is non-deterministic in the $\mathcal{ALC}$ tableau algorithm.

In this work, the manager-worker model is implemented based on Pellet by means of MPI as a distributed memory program using MPJ Express library. Pellet [21] is a tableau-based OWL-DL reasoner developed by the Mind Swap group. It is an open source reasoner and developed in Java. In MPI programming, all processes execute the same program executable and each process is identified by means of a special process identifier, called rank, which is unique within a group of processes involved in the execution. The rank allows every process to identify what part of the data to be processed. In order for two processes to communicate, we use MPI blocking *send* and *receive* operations. The blocking operations do not return until the communication is finished.

There are many technical challenges in implementing this dynamic manager-worker algorithm in Java by means of MPI. One of the major challenges is passing an object (e.g., an ABox) from one process to another. As the object is not a primitive data type, to pass an object using MPI, all classes of that object must implement the *Serializable* interface. Since we are working on legacy code, it is not feasible for every class to implement the *Serializable* interface. Moreover, standard Java serialization is inefficient both in terms of speed and size. To deal with these problems, we converted an object to byte vectors using KRYO[3], a fast and efficient serialization framework for Java, and sent these byte vectors using the same method as primitive byte buffers. At the receiving end, the object is reconstructed using these byte buffers.

5.3 Evaluation

In the manager-worker model described in the previous section, the manager is dedicated to distributing work items to workers and does not itself do any computation. Consequently, if there are p processes, only $p-1$ processes are available to process the computation tasks, i.e., to perform the consistency checking. Therefore, maximum parallel efficiency can be obtained in this scheme is $[(p-1)/p] \times 100\%$. In order to remove this limitation, it is possible to ask the manager to participate in the computation as do other workers. In that case, all p processes will participate in the computation, but the manager may be less likely to be available to respond instantly to the worker's requests. If there is very large number of workers, the processing of requests for the work item on the manager may become a bottleneck. If t_{chk} is the average time required to perform consistency checking on a given ABox on a worker, and t_{req} is the time required to process a request for a work item on the manager, then the manager can process t_{chk}/t_{req} requests without keeping any worker waiting. So, the maximum number of workers that the manager can support efficiently in this scheme is t_{chk}/t_{req}. The time required to perform consistency checking on a given ABox can vary from one ABox to another. However, t_{chk} is sufficiently large compared to t_{req}. So we can expect that this scheme will perform efficiently for a large number of processes. The efficiency of this scheme also depends on the existence of non-determinism in a KB. Typically, the number of disjunctions exist in a KB is very large compared to p. For example, *Thesaurus* ontology contains 83,644 subsumption axioms and 10,242 equivalent axioms. Since each equivalence axiom can be replaced by two subsumption axioms, there are approximately 105,000 disjunctions in total. The possible advantage of parallelism increases for more expressive fragments, where there are more non-deterministic rules.

It is noted that no optimization techniques have been addressed in this manager-worker model. As it takes much time to develop a new reasoner, we implemented our manager-worker model on top of Pellet. We conducted our experiments on the ACENET[4] cluster both in shared memory (using multi-core configuration of MPJ Express) and distributed memory (using cluster configuration of MPJ Express) environments. As is observed in the Table 2, our preliminary results are not encouraging. The main reason for performance degradation is the absence of optimizations. Pellet implements most state-of-the-art optimization techniques whereas there are no optimizations in our model. Dependency-directed backtracking, also known as backjumping, is the most significant among them. Backjumping allows an algorithm to detect the source of a clash and prune the search space to avoid facing the same clash again. When we distribute the computation into different computa-

[3]https://github.com/EsotericSoftware/kryo

[4]http://www.ace-net.ca/

Table 2: Consistency test result for the Transportation ontology.

Cluster configuration		Multi-core configuration	
No of Worker	Time (seconds)	No of Worker	Time (seconds)
1	5	1	20
3	6	3	32
5	9	5	23
7	13	7	35
11	23	11	47

tional nodes, it reduces the pruning possibility significantly. Backjumping allows the search space to be dramatically pruned and plays a significant role in the performance when KB contains a lot of disjunctions. As we internalize every subsumption relation, there are a lot of disjunctions. Therefore, the absence of backjumping degrades the performance drastically. The dramatic performance of backjumping is also mentioned in [15]. The way Pellet implements backjumping is not amenable to parallelization. We leave adopting state-of-the-art optimization techniques with this model as future work.

6. Conclusions and Future Work

With the progress of semantic web technologies, knowledge bases are becoming larger and more complex. Reasoning with large and complex ontologies is one of the biggest challenges for DL reasoners. In this work, the potential for improving the scalability of a DL reasoner via parallelization was investigated. A parallel model was developed for handling non-determinism arising from tableau-based reasoning. This parallel model allows the execution of non-deterministic rules on independent processes. The parallel model was implemented to check the consistency of a KB in a distributed memory environment using MPI. Even though the model was implemented for the DL $\mathcal{ALC}$, the provided algorithm is applicable for the whole DL family including $\mathcal{SROIQ}$. For the parallel implementation, MPJ Express, a Java MPI library, was used. In this work, the process based programming model is applied to Java, which brings the parallel computation paradigm, i.e., MPI, closer to Java.

Most DL reasoners implement tableau algorithms with a set of optimization techniques. State-of-the-art optimization techniques are keys to the performance of a modern tableau-based reasoner. So, a parallel model should address the main optimizations to achieve the high performance. Currently, we are in the process of implementing this parallel model with a set of optimizations, namely dependency-directed backtracking, semantic branching, etc., in order to get better performance. We also plan to implement this model for an expressive DL, e.g., $\mathcal{SHIQ}$, in the near future.

Acknowledgments: The second author wishes to thank the Natural Sciences and Engineering Research Council of Canada for financial support.

References

[1] J. Faddoul and W. MacCaull, "Parallelizing algebraic reasoning for the description logic $\mathcal{SHOQ}$," *The 4th Canadian Semantic Web Symposium (CSWS 2013)*, pp. 20–23, 2013.

[2] T. Liebig and F. Müller, "Parallelizing tableaux-based description logic reasoning," in *Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems*, 2007, pp. 1135–1144.

[3] T. Liebig, A. Steigmiller, and O. Noppens, "Scalability via parallelization of OWL reasoning," in *Proceedings of the 4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic*, 2010, pp. 39–43.

[4] K. Wu and V. Haarslev, "Parallel OWL reasoning: Merge classification," in *Proceedings of the 3rd Joint International Semantic Technology (JIST) conference*, 2014, pp. 211–227.

[5] A. Cheptsov, "An approach for distributed parallelization of large-scale Semantic Web reasoners based on MPI," in *Web Information Systems Engineering–WISE 2011 and 2012 Workshops*, 2013, pp. 4–12.

[6] The MPI standard. [last accessed: December, 2015]. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/.

[7] A. Shafi, B. Carpenter, and M. Baker, "Nested parallelism for multi-core HPC systems using Java," *Journal of Parallel and Distributed Computing*, vol. 69, no. 6, pp. 532–545, 2009.

[8] J. Bao, D. Caragea, and V. G. Honavar, "A distributed tableau algorithm for package-based description logics," in *Proceedings of the Second International Workshop on Context Representation and Reasoning*, 2006.

[9] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal, "WebPIE: A web-scale parallel inference engine using MapReduce," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 10, pp. 59–75, 2012.

[10] K. Wu and V. Haarslev, "A Parallel Reasoner for the Description Logic $\mathcal{ALC}$," in *Proceedings of the 2012 International Workshop on Description Logics (DL-2012)*, 2012, pp. 378–388.

[11] D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. Della Valle, F. Fischer, Z. Huang, A. Kiryakov, T. Lee, L. Schooler, V. Tresp, S. Wesner, M. Witbrock, and N. Zhong, "Towards LarKC: a platform for web-scale reasoning," in *Semantic Computing, 2008 IEEE International Conference on*, 2008, pp. 524–529.

[12] M. Assel, A. Cheptsov, G. Gallizo, K. Benkert, and A. Tenschert, "Applying high performance computing techniques for advanced semantic reasoning," in *eChallenges*, 2010, pp. 1–8.

[13] J. Faddoul and W. MacCaull, "Handling non-determinism with description logics using a fork/join approach," *International Journal of Networking and Computing*, vol. 5, no. 1, pp. 61–85, 2015.

[14] M. Hossain, "Inconsistency-tolerant description logic reasoning," M.Sc. thesis, St. Francis Xavier University, Canada, 2016.

[15] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press, 2003.

[16] I. Horrocks, U. Sattler, and S. Tobies, "Reasoning with individuals for the description logic $\mathcal{SHIQ}$," in *Proceedings of the 17th International Conference on Automated Deduction*, 2000, pp. 482–496.

[17] F. Baader and U. Sattler, "An overview of tableau algorithms for description logics," *Studia Logica*, vol. 69, no. 1, pp. 5–40, 2001.

[18] R. Möller and V. Haarslev, "Tableau-based reasoning," in *Handbook on Ontologies*. Springer, 2009, pp. 509–528.

[19] K. M. Chandy and S. Taylor, *An Introduction to Parallel Programming*. USA: Jones and Bartlett Publishers, Inc., 1992.

[20] M. Baker, B. Carpenter, G. Fox, S. H. Ko, and S. Lim, "mpiJava: an object-oriented Java interface to MPI," in *Parallel and Distributed Processing*. Springer, 1999, pp. 748–762.

[21] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

Evaluating a Persistent Soft Fault Model on Preconditioned Iterative Methods

Evan Coleman[1,2], **Masha Sosonkina**[2]
[1]Naval Surface Warfare Center - Dahlgren Division, Dahlgren, VA, USA
[2]Modeling, Simulation, and Visualization Engineering Department, Old Dominion University, Norfolk, VA, USA

Abstract— *The impact of soft fault errors on the GMRES iterative method with flexible preconditioning (called FGMRES) is explored. In particular, a new method for simulating soft fault errors is implemented directly in FGMRES, and the effect of error magnitude and timing is evaluated for the FGMRES convergence in solving of an elliptical PDE problem on a regular grid. Two types of preconditioners are explored, featuring an incomplete LU factorization and an algebraic recursive multilevel solver ARMS. The experiments have confirmed an intuition that, in general, injecting perturbation-based faults at the matrix-vector operation stage had a greater impact on the convergence rate than doing so at the preconditioning application stage, resulting in more cases when the iterative solver failed. In addition, several cases of better convergence under faults in preconditioning operation were observed and analyzed.*

Keywords: iterative solvers, preconditioning, fault model, flexible GMRES, pARMS

1. Introduction

Fault tolerance methods are devised to increase both reliability and resiliency of high-performance computing (HPC) applications on exascale platforms, in which the mean time to failure (MTTF) is projected to decrease dramatically due to the sheer size of the computing platform [4]. There are many reports (e.g., [1], [4], [16]) that discuss the expected increase in the number of faults experienced by HPC environments. This is expected to be a more prevalent problem as HPC environments continue to evolve towards larger systems. As the landscape of HPC continues to grow into one where experiencing faults during computations is increasingly commonplace, the software used in HPC applications needs to continue to change alongside it in order to provide an increased measure of resilience against the increased number of faults experienced. Typically, faults are divided into two categories: hard faults and soft faults (see, e.g., [6], [10]). Hard faults come from negative effects on the physical hardware components of the system and cause program interruption. As hardware components themselves continue to evolve and grow both smaller and faster, they (generally) become more prone to error, and the algorithms and software packages that are used in HPC environments need to be able to respond to sudden and unexpected changes in both the quantity and quality of the physical resources that may be available for use. The other category of faults, soft faults, are the focus of this work. This category of failures captures all faults that a program might experience that do not immediately interrupt program execution. Most often, these faults refer to some form of data corruption that is occurring either directly inside of, or as a result of, the algorithm that is being executed. It is possible for a program to detect the presence of a soft fault while it is still executing. In order to properly investigate the impact of soft errors, one needs to select a fault model that fully encapsulates all of the potential impacts of a soft fault, implement the selected fault model into the algorithm to be investigated, and conduct the necessary experiments to determine the potential impact of a fault occurring during the selected algorithm. Typically, soft faults have been modeled by a bit flip. This study focuses on utilizing an arguably more general approach towards the modeling of soft faults, and subsequently evaluating it in the case study of the Flexible GMRES (FGMRES) [14] iterative solver. Faults were injected as small perturbations to results of certain mathematical operations using a modified version of fault injection found in [6] and [8].

The rest of the paper is organized as follows: in Section 2, a brief overview of related studies is provided, in Section 3, details concerning the fault model that is used throughout this work are given, in Section 4, experimental results are provided, and in Section 5, a quick summary is presented along with possible directions for future work.

2. Related Work

Traditionally, when performing experiments to analyze the potential impact of soft faults upon a computing environment, researchers have relied primarily upon the injection of bit flips into a particular portion of the routine [3], [9]. In contrast, in the work by Elliot, Hoemmen, and Mueller [8], [6], faults are modeled in a more general sense. These studies choose to generalize the simulation of soft faults to producing an incorrect solution to one of key computational parts, such as the application of the preconditioner inside of an iterative solver. This approach generalizes the simulation of soft faults by disregarding the actual source of the fault

and allowing the fault injector to create as large or as small a fault as necessary for the experiment. In the experiments conducted in [8], [6], [7] faults are typically defined as either a scaling of the contribution of the result of the preconditioner application for the Message Passing Interface (MPI) process in which a fault was injected, or a permutation of the components of the vector result of the preconditioner application for the MPI process in which a fault was injected.

In the taxonomy of faults given in [6], [10] soft faults are divided into the categories of transient, sticky, and persistent. Transient faults are defined as faults that occur only once, sticky faults indicate a fault that recurs for some period of time but where computation eventually returns to a fault-free state, and persistent faults arise when the fault is permanent. Whether the studies discussed above model faults using bit flips or adopt a more numerical analysis style approach, much of the previous work on the impact of silent data corruption (SDC) has to do with modeling transient errors. The goal of this effort is to present a fault model that can accurately predict the impact of persistent soft faults. Examples of scenarios that could cause a persistent fault are a stuck bit in memory, or a hardware malfunction – such as the Intel Pentium FDIV bug – or the incorrect copy of data from one location to another. [6], [5], [10] The model presented here is general enough that it can be adapted to simulate the impact of any persistent error, including those caused by hardware malfunction.

Traditional analysis of potential persistent type errors has rested more in the hardware domain than in the algorithmic domain, with analysis of both processor based faults [11], [2] and memory based faults [15]. The impact of persistent faults on iterative methods does not seem to have explored to a great extent. The work presented here follows an idea from [8], [6], [7] of disregarding the source of the error in the simulated fault. In other words, an analytical approach is taken as opposed to flipping random bits inside some pertinent data structures. On the other hand, here the simulated soft faults persist once injected as opposed to work in [8], [6] where the faults are transient in nature.

3. Fault Model

The approach chosen was to perturb the vector result of key computations for the single MPI process in which a fault was injected. This perturbation-based fault model is an adaptation of the fault model presented in [6], [8], [7]. The fault model presented in [6], [8], and [7] focuses exclusively on modeling transient faults; the fault model presented here attempts to modify that approach to extend it to persistent soft faults. In an attempt to accurately model the impact of a persistent soft fault, a small randomized perturbation is injected on each iteration after the fault is modeled to occur. Adopting a characterization from [7], faults are divided based upon their impact to the l^2-norm of the vector they are injected into. The default version of the fault model presented here relies on the generation of small random numbers that are added to each element of the data structure where the fault is injected. However, the fault model was also adapted to allow for the possibility of either moving each element of the vector to be perturbed further away from, or closer towards, zero. In this way, the fault model offers some level of control over whether the l^2-norm of the vector the fault is injected into increases or decreases. This allows observations about the effect of perturbing the l^2-norm on the general convergence of the total algorithm. Since FGMRES works towards reducing the l^2-norm of the residual vector, modifying the l^2-norm of any of the vectors used to construct the residual vector may affect its convergence rate by affecting the l^2-norm of the residual in a ripple-like effect [7], [14]. Hence, one of three outcomes may occur: The iterative solver will converge at about the same rate as without perturbation, or converge but will take significantly longer to reach convergence, or fail to converge entirely [8]. However, it is also possible that the injected fault will actually cause the iterative solver to converge in fewer iterations than without perturbations.

It is important to design the fault model in such a way that it encapsulates the worst-case behavior that one is trying to protect against. By modeling faults as a random perturbation, a controlled amount of noise is added to the result of key operation inside of the algorithm. The amount of this noise is parameterized throughout the different experiments. However, the random nature of the faults limits the knowledge of specific details regarding the fault that was injected. As persistent faults are one of three types of soft faults accounted for in the taxonomy of soft faults presented in [6], [10], designing a fault model to accurately measure the impact of these faults is an important endeavor. The nature of the fault model presented here, in that every iteration inside of the iterative solver is perturbed after the fault initially occurs, provides a way to quantify the impacts of a potential persistent soft fault.

3.1 FGMRES

The FGMRES algorithm, as described in [14], is provided in Algorithm 1. FGMRES is similar in its nature to the standard GMRES with the notable exception of allowing the preconditioner to change in each iteration by storing the result of each preconditioning operation (cf. matrix Z_m in line Line 10). FGMRES was selected in this study because it is a robust, popular iterative solver which is proven to converged under variable preconditioning, possibly resulting from a perturbation in the preconditioning operation. Here, such a perturbation is due to injected faults. In particular, faults were injected at two distinct points inside of the FGMRES algorithm; Line 1, termed here as the *outer matvec* operation, and Line 3, which the application of the preconditioner. In this study, the effect of injecting faults exclusively into one of these two locations as well as into both locations

simultaneously was considered. Also, the GMRES restart parameter (m in Algorithm 1) was taken to be 20.

Input: A linear system $Ax = b$ and an initial guess at the solution, x_0
Output: An approximate solution x_m for some $m \geq 0$
1 $r_0 = b - Ax_0,\ \beta = ||r_0||_2,\ v_1 = r_0/\beta$
2 **for** $j = 1, 2, \ldots, m$ **do**
3 $\quad z_j = M_j^{-1} v_j$
4 $\quad w = Az_j$
5 $\quad$ **for** *i = 1, 2, …, j* **do**
6 $\quad\quad h_{i,j} = w \cdot v_i$
7 $\quad\quad w = w - h_{i,j} v_i$
8 $\quad$ **end**
9 $\quad h_{j+1,j} = ||w||_2,\ v_{j+1} = w/h_{j+1,j}$
10 $\quad Z_m = [z_1, \ldots, z_m],\ \bar{H}_m = h_{i,j\,1 \leq i \leq j+1; 1 \leq j \leq m}$
11 **end**
12 $y_m = argmin_y ||\bar{H}_m y - \beta e_1||_2,\ x_m = x_0 + Z_m y_m$
13 **if** *Convergence was reached* **then return** x_m
14 **else** set $x_0 \leftarrow x_m$, **GoTo** Line 1

Algorithm 1: FGMRES as given in [14]

3.1.1 Preconditioner

A traditional linear system is given by $Ax = b$, however a transformed *preconditioned system* is given by $M^{-1}Ax = M^{-1}b$, when preconditioning is applied from the left, and $AM^{-1}y = b$ with $x = M^{-1}y$, when preconditioning is applied from the right. The matrix M is a nonsingular approximation to A, and is called the *preconditioner*. Incomplete LU factorization methods (ILUs) are an effective class of preconditioning techniques for solving linear systems. They define the preconditioner as $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are approximations of the L and U factors of the standard triangular LU decomposition of A. The incomplete factorization may be computed from the Gaussian Elimination (GE) algorithm, by discarding some entries in the L and U factors. If the m independent unknowns are numbered first, and the other $n - m$ unknowns last, the coefficient matrix of the system is permuted in the 2×2 block structure. In multi-elimination methods, a reduced system is recursively constructed from the permuted system by performing a block LU factorization of PAP^T of the form

$$PAP^T = \begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ G & I_{n-m} \end{pmatrix} \times \begin{pmatrix} U & W \\ 0 & A_1 \end{pmatrix}$$

where D is a diagonal matrix, L and U are the triangular factors of the LU factorization of D, $A_1 = C - ED^{-1}F$ is the Schur complement with respect to C, I_{n-m} is the identity matrix of dimension $n - m$, and then denote by $G = EU^{-1}$ and $W = L^{-1}F$. The reduction process can be applied another time to the reduced system with A_1, and recursively to each consecutively reduced system until the Schur complement is small enough to be solved with a standard method. The factorization of PAP^T above defines a general framework which accommodates for different methods. The ARMS preconditioner uses block independent sets to discover sets of independent unknowns and computes them by using the greedy algorithm. In the ARMS implementation used here, the incomplete triangular factors $\bar{L}$, $\bar{U}$ of D are computed by one sweep of ILUT. In the second loop, an approximation $\bar{G}$ to $E\bar{U}^{-1}$ and an approximate Schur complement matrix $\bar{A}_1$ are derived. This holds at each reduction level. At the last level, another sweep of ILUT is applied to the (last) reduced system.

3.1.2 Fault Detection and Resilience in FGMRES

Fault detection inside of FGMRES can be achieved in many different ways. Upon each restart of FGMRES, the norm of the residual is computed, and in a fault-free environment these norms should be monotonically decreasing. A cheap fault detector could be implemented to check this, and it would be an intuitive way to attempt to detect faults that occur during the outer sparse matrix-vector multiply. It will be shown experimentally that if the fault that is injected into the outer sparse matrix-vector multiply does not increase the norm of the initial residual, than it has a significantly less negative effect on the convergence of FGMRES.

One of the key observations made in [10] was that since the preconditioner is allowed to change on every iteration in the FGMRES algorithm, faults that occur during the precondioning operation (Line 3 in Algorithm 1) can be modeled as different preconditioners. As such, if a fault were to occur anywhere inside of the preconditioning operation it can be modeled by injecting a fault into the result of the preconditioning operation (z_j in Algorithm 1). The perturbation-based fault model proposed in this paper allows the size of the fault to be controlled by offering direct control on the size of the perturbation that is injected.

It will be shown experimentally that FGMRES is capable of proceeding through many faults occuring in the preconditioning operation by accepting the faulty output as a different preconditioner. This natural adaptive response in the FGMRES algorithm to faults that occur during preconditioning should also cause faults that occur during the outer sparse matrix-vector multiply to have more of an impact on the convergence of FGMRES. This was also able to be shown experimentally, and results are provided in Section 4.

4. Experimental Results

The test problem that was used comes directly from the pARMS library [12], and represents an elliptic 2D partial differential equation,

$$-\Delta u + 100\frac{\partial}{\partial x}(e^{xy}u) + 100\frac{\partial}{\partial y}(e^{-xy}u) - 10u = f$$

Parameter	Acceptable Values
Global Preconditioner	Block Jacobi
Local Preconditioner	ILUT, ARMS
Tolerance Required for Convergence	10^{-6}
Starting Iteration at which Fault Appears	≥ 5
Order of Perturbation	$10^{-6}, \ldots, 10^{-4}$
Effect on l^2-norm	Any, Decrease, Increase

Table 1: Input parameters the value of which varied in the experiments.

on a square region with Dirichlet boundary conditions, discretized with a five-point centered finite-difference scheme on a $n_x \times n_y$ grid, excluding boundary points. The mesh is mapped to a virtual $p_x \times p_y$ grid of processors, such that a subrectangle of $r_x = n_x/p_x$ points in the x direction and $r_y = n_y/p_y$ points in the y direction is mapped to a processor. The size of the problem was varied and controlled by changing the size of the mesh that was used in the creation of the domain. The mesh sizes that were considered ranged from $n_x = n_y = 100$ to $n_x = n_y = 500$, and these mesh sizes were run on numbers of processors that varied from four ($p_x = p_y = 2$) to 100 ($p_x = p_y = 10$). In order to compare experiments run with different parameters the resulting number of iterations was compared to the number of iterations in the same unperturbed run and depicted as the percentage in the plots throughout this section.

In all of the experiments that were conducted, multiple sets of runs were executed for each set of parameters (i.e. perturbation size, iteration fault was first injected) and their effect on the convergence of FGMRES was combined into an average with all other runs with the same parameters before being analyzed. The parameters that were varied in these experiments are detailed in Table 1.

Since the fault model presented here is based on a series of random perturbations, multiple runs/solves were conducted for each set of parameters, and the results were averaged and depicted in the plots of this section. In all of the experiments, a maximum number of iterations was instituted and, if a run did not converge within this preset number of iterations, then it was terminated, and determined to have failed. The focus is on investigating the effects on two of the local preconditioners available within pARMS: Incomplete LU with dual nonzero dropping strategy [14] (referred to as ILUT), and the ARMS preconditioner [13].

The experiments conducted for this study were run on two distinct hardware environments. The first test environment was a node with Intel Core i7 processor having four physical cores at 2.50 GHz each and 16 GB of main memory. The second was the Hopper supercomputer, which is a compute resource of the National Energy Research Scientific Center (NERSC). Hopper has a total of 153,216 compute cores, 212 Terabytes of memory and nodes are connected with a custom high-bandwidth, low-latency network provided by Cray. Up to five compute nodes of Hopper were utilized. The problem size was scaled appropriately for each environment, by adjusting the size of the square mesh per subdomain; namely, 200 and 500 points for the Intel Core i7 and Hopper, respectively.

The results shown in all the figures of Sections 4.1 to 4.3 come from runs on the Intel Core i7 platform using four MPI ranks, one per core. The results from the runs with larger problem sizes performed on Hopper showed similar convergence tendencies under perturbation-based faults considered here. Note that, in all the plots, the x-axis represents the fraction (as %) of the execution when a fault begins and the y-axis shows the increase (or decrease) in the number of iterations with respect to non-perturbed case. For example, a data point with x coordinate of 50% shows an effect from the fault injected *halfway through the number of iterations that would be required by a fault-free run*. This effect is quantified by the y coordinate of the point, such that, if $y = +100\%$, e.g., then the run corresponding to this data point required twice as many iterations to converge than that did in a fault free case.

4.1 Matvec Perturbations

The first set of experiments affected only the outer matvec operation in the FGMRES algorithm (see Line 1). The following results are provided the instance where the sign of the perturbation (and hence, the magnitude of the l^2-norm) was not controlled by the fault model. Figure 1 shows the effects of faults with various perturbation sizes in outer matvec when the ARMS (top) or ILUT (bottom) local preconditioner is used. Only perturbation sizes no larger than 5×10^{-5} are shown since for larger values the solver failed to converge. Comparing the results in Fig. 1 (top) and (bottom), a similar convergence behavior may be observed. However, the faults corresponding to smaller perturbations ($10^{-6}, \ldots, 5 \times 10^{-5}$) have a slightly greater negative effect on the runs with the ILUT preconditioner than on those with ARMS. When examining effects of very small perturbations (on the order of convergence tolerance, which is 10^{-6} here), it was found that they had either no effect at all on the convergence rate or slightly decreased the total number of iterations. This beneficial effect was noted regardless of when during the run the fault has started, and it appears more often with the ARMS preconditioner than with ILUT.

Next, results for the case where the sign of the perturbation was matched with the sign of the existing vector component in order to ensure that the l^2-norm of the perturbed operation result decreased (Fig. 2). In order to match the sign appropriately, the fault model checks the sign of the original vector component before applying the fault.

It is interesting to observe in Fig. 2 the increased rate of successful convergence for a much larger range of fault magnitudes. A larger spectrum of perturbation sizes resulted in successful convergence and, hence, is represented in Fig. 2. Comparing the effects of injecting faults that vary the l^2-

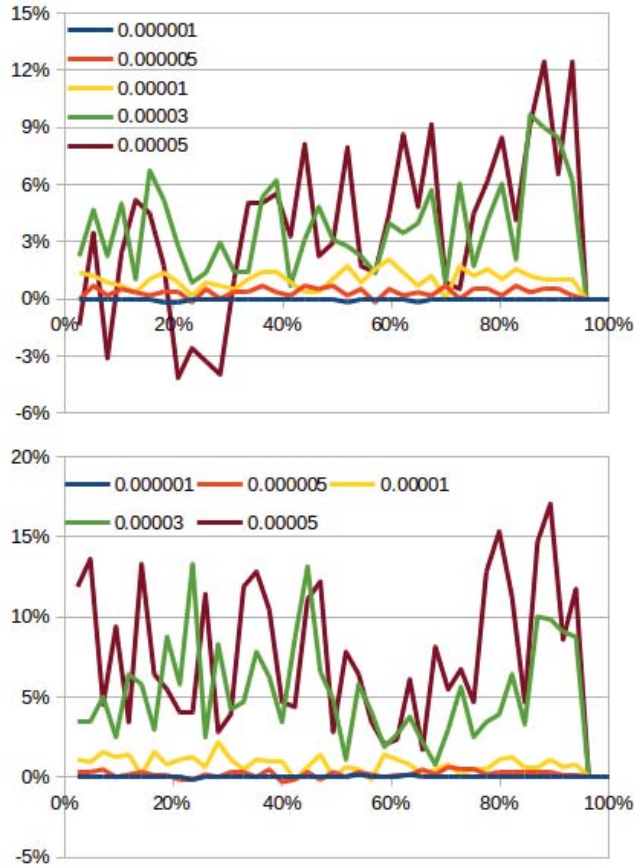


Fig. 1: Outer matvec perturbation faults with varied l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

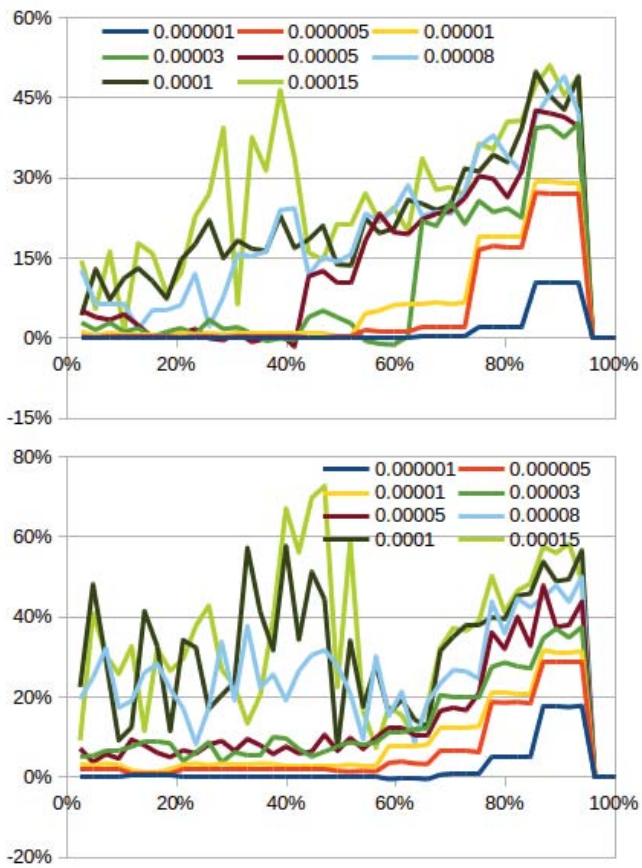


Fig. 2: Outer matvec perturbation faults with decreasing l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

norm (Fig. 1) to those that shrink the l^2-norm (Fig. 2), there is also a decrease in the negative effect that a fault of the same magnitude has upon the FGMRES algorithm. In general, the performance of the two preconditioners is fairly similar in the case when faults are incurred in the outer matvec operation. For instance, as expected, there is a tendency for the fault to have more of an impact on the convergence if the fault commences later in the execution; smaller perturbations show little effect while larger perturbations produce a much higher variations of convergence results. Perturbed executions resulting in fewer iterations than non-perturbed ones appear to arise with about equal frequency between scenarios using either the ARMS or the ILUT preconditioner. These results are seen for all the fault sizes—although much more commonly for faults of size $\leq 10^{-4}$—and are observed most when faults occur before the run reaches approximately 60% of completion of a fault-free run.

4.2 Preconditioner Perturbations

Results (Fig. 3) are presented for each of the two preconditioners, ARMS and ILUT, and exclusively for the version of the perturbation-based soft fault model that decreases the l^2-norm of the vector that it is applied to. Comparing the results with the ILUT preconditioner to those with ARMS, it again appears that the runs with the latter suffer less of a negative effect than those with the former for the faults of an equivalent size. Next, when examining results with the ARMS preconditioner in Fig. 3(top), it is clear that injecting a perturbation-based fault into the result of the application of the preconditioner (from Line 3 in Algorithm 1) has less of an effect on a FGMRES solve using the ARMS preconditioner than that from injecting a similar fault into the result of the outer matvec iteration (Fig. 2(top)). Even a magnitude of fault (e.g., 10^{-4}) that may cause stagnation when injected into the outer matvec operation, causes only a 40–50% increase in the total number of iterations here and only has a large impact if injected throughout the

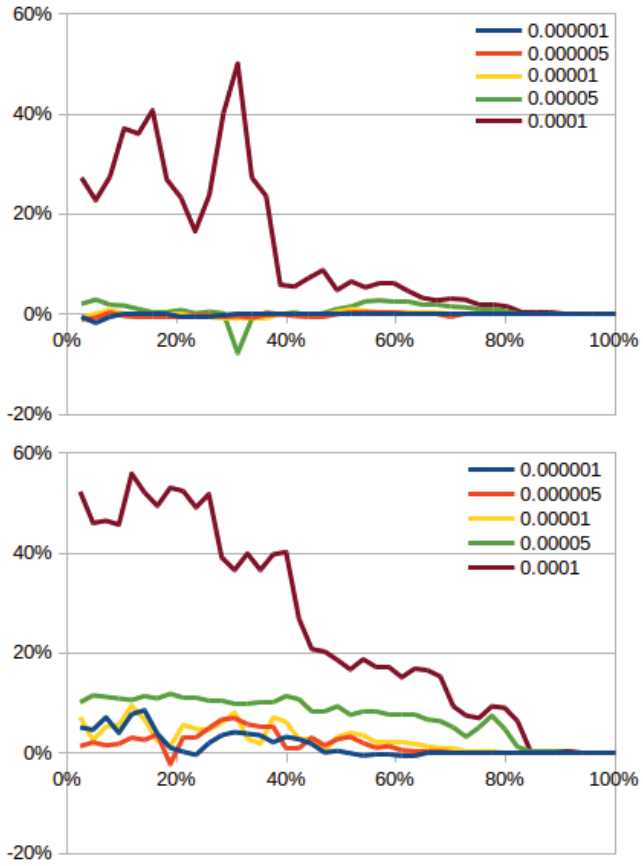


Fig. 3: Preconditioner perturbation faults with decreasing l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

Fig. 4: Outer matvec and preconditioner application faults with decreasing l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

majority of the run. Similar observations may be made for the ILUT preconditioning in Fig. 3(bottom): less of a negative effect is evident when perturbation-based faults appear in this preconditioning operation than in the outer matvec (cf. Fig. 2(bottom)). In general, FGMRES, being able to converge with a preconditioner that changes at each iteration, does not negatively react to perconditioner changes due to faults in the course of linear system solution.

4.3 Matvec and Preconditioner Perturbations

The graphs in Fig. 4 show the effect of injecting a fault into the two fault sites considered simultaneously (i.e., at the same FGMRES iteration), the outer matvec *and* preconditioner application, such that the l^2-norm decreases. In Fig. 4, notice that, for large faults (starting at 10^{-4}), the increase in the number of iterations required to converge was very high—between 400-600% at times. This increase is also much higher than that for either matvec- or preconditioner-only incurred faults producing the highest increases of $\sim$60% and $\sim$55%, respectively, for the same perturbation value of 10^{-4}. Conversely, for perturbation sizes of 10^{-5} and smaller, the effect on convergence appears similar to that of either matvec faults. This suggests that the ability of FGMRES to accept faulty preconditioners is inhibited by the coexistence of a matvec fault. Also, there were fewer cases where the number of iterations to converge decreased due to faults.

All of the experiments were also performed with a variant of this perturbation-based fault model that increased the l^2-norm of the operation result. In all instances, smaller fault sizes caused FGMRES to fail to converge compared with the other l^2-norm variants of the fault model, and, for the cases in which the iterative solver converged, many more iterations were required. Due to space considerations, the experiments with the increasing l^2-norm are not detailed in this paper.

Fault Size	Fault Location	Starting Its	l^2-norm Effect	PC	Improvement
5×10^{-5}	matvec	0% - 30%	Varied	ARMS	2% - 4%
10^{-6}	matvec	(anywhere)	Varied	ARMS	0% - 1%
$10^{-6} - 10^{-5}$	matvec	(anywhere)	Varied	ILUT	0% - 1%
(any size)	matvec	0% - 60%	Decreasing	ARMS, ILUT	0% - 1%
$\leq 5 \times 10^{-5}$	PC	(anywhere)	Decreasing	ARMS	0% - 5%
$\leq 5 \times 10^{-6}$	PC	(anywhere)	Decreasing	ILUT	0% - 2%

Table 2: Summary of Beneficial Results- Note: Its (Iteration), PC (Preconditioner)

5. Summary and Future Work

This paper showcased experiments designed to exhibit a persistent fault model with faults affecting bounds within an iterative solver, which may be monitored and play a role in the solver reaction to faults. Specifically, effects on the l^2-norm of the fault-perturbed vector were explored and it was found that persistent faults may be treated similarly to episodic faults in quantifying their effects except that the application possibly needs to adjust to continuing operation "under failure". An investigation of such adaptations is left as a future work. In particular, persistent faults that shrink the l^2-norm have less of a negative effect upon the convergence of the iterative solver. It was also found that injecting faults into the outer matvec operation, in general, had a greater impact upon the FGMRES convergence than doing so for the preconditioner application—including causing more cases in which the iterative solver failed—which was observed for both the ARMS and ILUT preconditioners. It appears that runs using ARMS preconditioner are more naturally resilient to the injection of persistent perturbation-based faults than runs using the ILUT preconditioner; regardless of which of the two fault sites is chosen. In addition, a small fault injection resulted in several runs that converged in up to 5% fewer iterations than would be typically required. Table 2 summarizes beneficial outcomes from the results presented in this paper. In the future, it is planned to use this summary as a guide in devising algorithm based fault tolerance procedures for the flexible GMRES.

5.1 Acknowledgments

This work was supported in part by the Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476, by the National Science Foundation grant 1516096, by the U.S. Department of Energy (DOE), Office of Advanced Scientific Computing Research, through the Ames Laboratory, operated by Iowa State University under contract No. DE-AC02-07CH11358, and by the U.S. Department of Defense High Performance Computing Modernization Program, through a HASI grant. This work used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] K Asanovic, R Bodik, BC Catanzaro, JJ Gebis, P Husbands, K Keutzer, DA Patterson, WL Plishker, J Shalf, SW Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[2] FA Bower, DJ Sorin, and S Ozev. Online diagnosis of hard faults in microprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(2):8, 2007.

[3] G Bronevetsky and B de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 155–164. ACM, 2008.

[4] F Cappello, A Geist, W Gropp, S Kale, B Kramer, and M Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.

[5] A Edelman. The mathematics of the Pentium division bug. *SIAM review*, 39(1):54–67, 1997.

[6] J Elliott, M Hoemmen, and F Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1193–1202. IEEE, 2014.

[7] J Elliott, M Hoemmen, and F Mueller. Tolerating Silent Data Corruption in Opaque Preconditioners. *arXiv preprint arXiv:1404.5552*, 2014.

[8] J Elliott, M Hoemmen, and F Mueller. A numerical soft fault model for iterative linear solvers. In *Proceedings of the 24nd International Symposium on High-Performance Parallel and Distributed Computing*, 2015.

[9] J Elliott, F Mueller, M Stoyanov, and C Webster. Quantifying the impact of single bit flips on floating point arithmetic. *preprint*, 2013.

[10] M Hoemmen and MA Heroux. Fault-tolerant iterative methods via selective reliability. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society*, volume 3, page 9. Citeseer, 2011.

[11] ML Li, P Ramachandran, SK Sahoo, SV Adve, VS Adve, and Y Zhou. Trace-based microarchitecture-level diagnosis of permanent hardware faults. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 22–31. IEEE, 2008.

[12] Z Li, Y Saad, and M Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical linear algebra with applications*, 10(5-6):485–509, 2003.

[13] Y Saad and B Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. *Numerical linear algebra with applications*, 9(5):359–378, 2002.

[14] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[15] B Schroeder, E Pinheiro, and WD Weber. DRAM errors in the wild: a large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 193–204. ACM, 2009.

[16] M Snir, RW Wisniewski, JA Abraham, SV Adve, S Bagchi, P Balaji, J Belak, P Bose, F Cappello, B Carlson, et al. Addressing failures in exascale computing. *International Journal of High Performance Computing Applications*, page 1094342014522573, 2014.

Performance Evaluation of Parallel Algorithms in R Statistical Package on Multicore Parallel Architectures

A. F. Kummer Neto[1] and A. Charão[2] and P. P. Barcelos[2] and B. O. Stein[2]
[1]Post-Graduate Program in Computer Science
[2]Department of Languages and Computing Systems
Federal University of Santa Maria, Brazil

Abstract— *This paper presents an empirical performance evaluation of some parallel algorithms implemented in R statistical software. We focus on multicore parallel architectures, as other related works are more concerned with parallel distributed architectures. Such an evaluation is important, given that we live in an era in which the amount of data streams is very large, thus requiring high performance techniques and tools. The results show that the parallel algorithms can be effective but not so efficient.*

Keywords: r statistical software; parallel algorithms; performance evaluation; multicore architecture

1. Introduction

R is a prominent free development environment used for statistical data analysis [1]. It comprises a domain specific programming language and a number of extensions and function libraries that implement algorithms for data mining such as sorting, clustering, association, among many others.

In statistical data analysis, there are many algorithms that can be significantly time-consuming, depending on the size of the input dataset. This becomes more of a concern in the current scenario of big, voluminous amount of data that is generated at speeding rates and has the potential to be analyzed for extracting useful information.

Given R's popularity in this domain, and also given the advances on parallel computer architectures, some authors and developers proposed solutions for parallel data analysis using R [2], [3], [4]. This gave rise to packages such as `multicore` [5], `snow` [6] and, finally, `parallel` [7], which is included in the core distribution of R. The functions provided by these libraries can also be combined with other packages to speedup time-consuming data analysis, as with, for example, Caret (short for Classification and Regression Training) [8] and Boruta [9] packages.

The packages we mentioned above cover distributed and shared memory parallel architectures but, to our knowledge, there is no thorough study of their parallel efficiency on such architectures. On the other hand, there is a widespread availability of multicore parallel architectures which support computations as provided by these packages. In this context, the objective of this work is to evaluate the parallel performance of some benchmarks on multicore platforms, using `multicore` and `parallel` packages. Throughout the article, we present the R environment, its extensions and support for parallel computations, the benchmarks we used and our computational experiments and, finally, we discuss the results.

2. R and Parallel Computing

R environment provide resources for handling, analysis and visualization of data. In addition to the native capabilities, R is designed to extensible through libraries for a wide variety of tasks, including big data analysis procedures and data mining tasks.

Concerning to data visualization and graphical user interfaces, we can highlight Rattle [10] and RStudio [11] distributions which bring a number of embedded data mining algorithms and a graphical user interface that further facilitates using R.

Among some extensions that lend themselves to data mining, we can cite Caret (short for Classification and Regression Training) [8] and Boruta [9] packages. The first is a compilation of functions that facilitates the creation of data models. The second is a framework that implements algorithms for feature selection and machine learning tasks.

Back to Caret package, it is used to simplify complex regression and classification problems. One of its features is the capability of concurrent execution of R code snippets via `multicore` package. As a performance concern, the Caret package does not load its entire function collection at once. These are loaded as needed to prevent memory waste [8].

Concerning to parallel processing, we can explore parallelism in `distributed` and `shared memory` systems through `snow` [6] and `multicore` [5] R extensions, respectively. These packages manages parallel jobs, dividing the computation between the available cores/hosts available. Note that these extensions are available for in Unix-based systems only.

Since R 2.14.0, `parallel` package [12] offers a drop-in replacement for `snow` and `multicore` functionalities–some of them depicted in Table 1–and brings support for dynamic work scheduling between workers.

Function	Memory model	Task scheduling
`clusterApplyLB`	distributed	dynamic
`parLapply`	distributed	static
`mclapply with mc.preschedule = FALSE`	shared	dynamic
`mclapply with mc.preschedule = TRUE`	shared	static

Table 1
GENERAL PURPOSE FUNCTIONS OFFERED BY `PARALLEL` PACKAGE TO PERFORM PARALLEL COMPUTATION ON SHARED AND DISTRIBUTED MEMORY SYSTEMS.

All functions listed in Table 1 have an interface similar to native `lapply` R function. `clusterApplyLB` and `parLapply` were designed to reduce communication between workers due to high overhead inherent to network communication. Communication is made via via sockets.

To exploit parallel processing in *shared memory* architectures, we can use `mclapply` function to spawn workers on a multicore system. In current version of R, the workers are implemented via processes and piped communication since R interpreter is not reentrant.

3. Benchmarks and Performance Metrics

To evaluate performance gains archieved with parallelism facilities available in R, we build a set of four benchmark applications. The first is application aim to take a user perspective of parallel computing. The remaining benchmark applications are elaborated to stress specific parts of interpreter as an attempt to reach a more robust analysis of software infrastructure. A complete list of benchmark applications is available in Table 2.

Benchmark application	Test case
`bench-caret`	User point of view; automatic parallelism over Caret native support for multicore architectures
`primes`	Unbalanced, independent tasks with simple reduction procedure; low memory and communication requirements
`primes-repeat`	Repetition of homogeneous tasks; CPU-bound application
`firesim`	Monte Carlo simulation for fire spreading on forest

Table 2
BENCHMARK APPLICATIONS USED IN THIS WORK.

To proceed with discussion of results, we use two metrics proposed by [13] to compute the gains of parallel approach over sequential one. The first metric is called *Speedup* and is calculated as

$$S_n = \frac{T_{seq}}{T_n} \tag{1}$$

where n indicate the number of concurrent tasks, T_{seq} and T_n is how long the sequential and concurrent procedures take to finish, respectively. The second metric measures how scalable the parallelism approach is through *Efficiency* indicator

$$E_n = \frac{S_n}{n}. \tag{2}$$

Ideally, we want a linear speedup, which incurs to efficiency of 1. In most cases, this is infeasible since hardware and software bound the gains of parallel approaches [14].

4. Case study: `bench-caret.R`

Elimination of meaningless information of a dataset is a common steps in data mining technique.

Elimination of meaningless information of a dataset is a common steps in data mining scenarios. An analysis with all the variables of a particular object of study is unviable, since the vast amount of data degrades the performance of algorithms and overload computational resources–sometimes worsening end result of analysis. To overcome this situations is usual to employ a characteristics selection filtering to eliminates any variable that is irrelevant for mining.

To elucidate the practice of data mining, we used a benchmark written in R called `bench-caret.R` [15]. It is structured as follows: initially, the benchmark loads the Caret and RandomForest packages (set of algorithms used for sorting); random samples of data with increasing size are generated; the generated data are analyzed by the code snippet responsible for processing. Lastly, the result is printed, stating the size of the sample tested and the time spent on execution.

To evaluate the performance of `bench-caret.R` benchmark, a computer with two Intel Xeon 2.00 GHz 4-core processors (8 cores the total) was used, 64-bit operating system Linux (Kernel 2.6.20), the statistical environment R with Multicore and Caret packages.

The tests were performed with 1 to 8 cores, with in each case three rounds were carried out the same test. This allowed a check fluctuations in time for the same number of cores. After collecting performance data, one can calculate the mean, standard deviation and the coefficient of variation for each core, encompassing three rounds each. There were more executions performed because the coefficient of variation was low (from 0 to 0.03) for all cases. Furthermore, a round was performed without multicore package to ascertain the effect on function of time.

As shown in Figure 1, the test took longer was the use of only one core. There was a big difference in time using

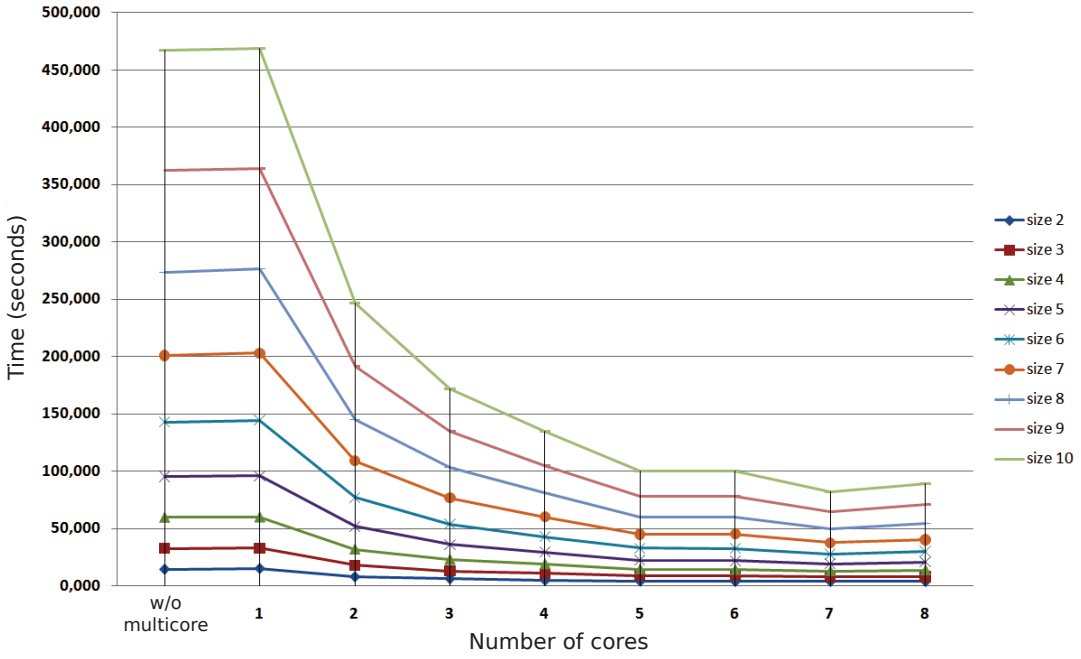


Fig. 1
EXECUTION TIMES FOR VARYING NUMBER OF CORES AND DATA SIZES

1, 2, 3, 4 and 5 cores. From there the differences became smaller. It should be noted that with 5 and 6 cores, times were nearly identical. Intuitively, one might think that with 8 test centers would have been done faster. However, it was not what happened. The test performed faster with 7 cores. For the sample sizes of 4, 5, 6, 7, 8, 9 and 10, the 8 cores were slower than 7. By rotating the test without Multicore, there was a decrease in execution time regarding the test 1 with core and with the function. This is because this type of function generates overhead.

Figure 2 brings the result of acceleration (speedup). It has a representation of what would be the ideal performance, but as you can see, is not what happens. Note, however, that the higher the data size, the greater the acceleration.

5. Case study: `primes` application

The `primes` application is used to calculate how many primes exists from 1 to a upper limit value ub, as illustrated in Routine (1). On the parallel version, we perform computations of lines (4 – 8) concurrently through functions of Table (1). As output, the application count how many primes exists in range $(1, ub]$.

For the following results, we conducted our computational experiments on a Dell PowerEdge R720 machine equiped with two Intel Xeon E5-2697 processors (each one working at 2.7 GHz) and 64 GiB of main memory (DDR3 1600 MHz SDRAM). As the version of R available on repositories of distribution used by this machine is old (R 3.1.1 in Debian 8 official repositories), we opted to a manual install of official R distribution 3.2.4 with standard library bundle. The machine uses a 64-bit Linux operating system 3.16.0.

Routine 1 Brute-force prime counting routine

Input: Upper limit ub for greatest number of testing sequence.
Output: $count$, Number of primes found in interval $(1, ub]$.

```
 1: count ← 0
 2: for v ∈ (1, ub] do
 3:     flag ← true
 4:     for i ∈ [2, v − 1] do
 5:         if v mod i ≠ 0 then
 6:             flag ← false
 7:         end if
 8:     end for
 9:     if flag = true then
10:         count ← count + 1
11:     end if
12: end for
```

The first Figure (3) show speedup reached up to 24 workers. The results show that `primes` application does not scales well on any of test cases, mainly due to unbalanced nature of tasks. Contrary to what we expect, dynamic assignment of task does not help to improve speedup of any test cases.

On package documentation, the authors pointed communication overhead could make dynamic load balancing prohibitive. That does not justificates poor performance of `primes` application since each task accomplished need to communicate only one byte to job scheduler process.

Low speedup rates result in poor parallelism efficiency. In

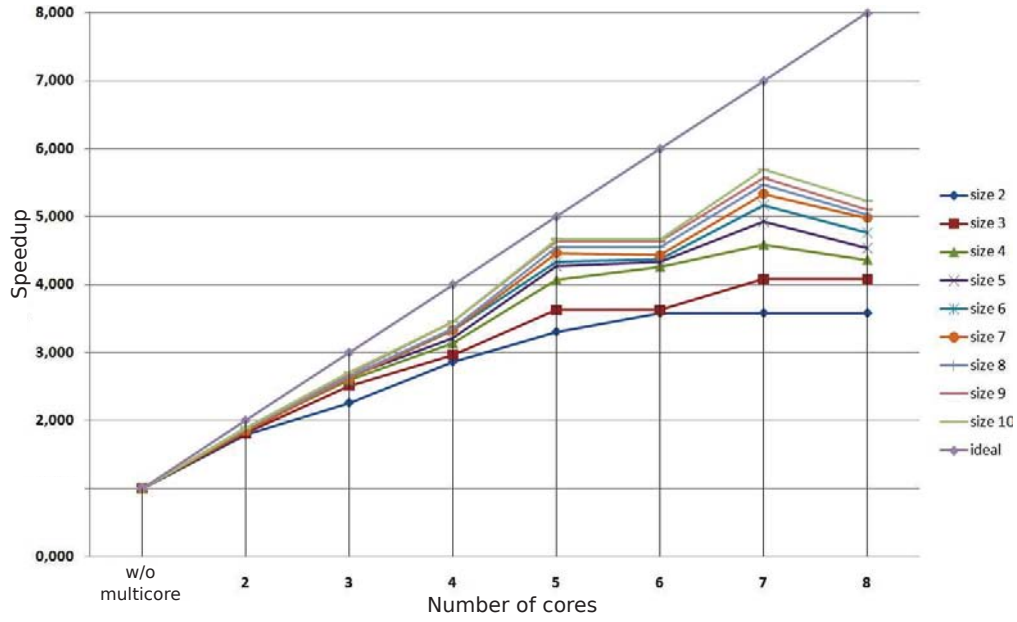


Fig. 2
SPEEDUP FOR BENCH-CARET.R

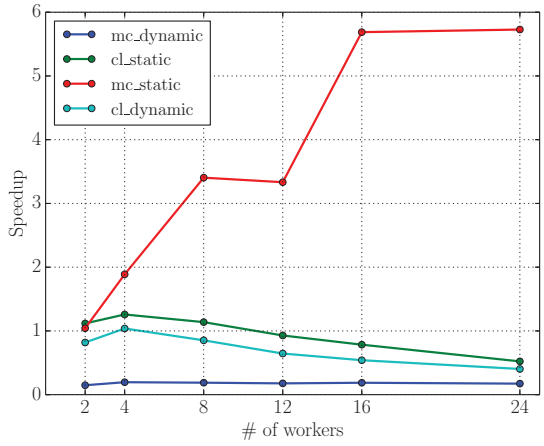


Fig. 3
SPEEDUP VERIFIED FOR PRIMES BENCHMARK

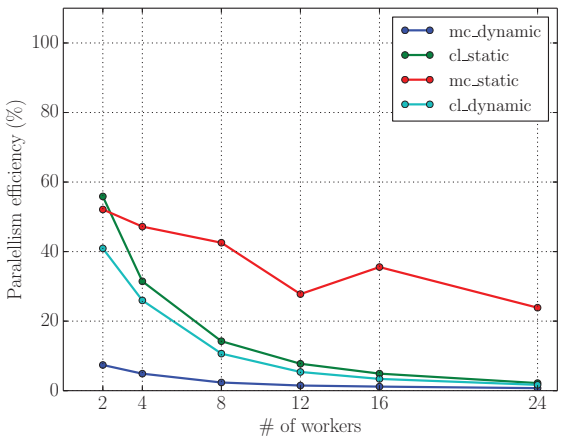


Fig. 4
PARALLELISM EFFICIENCY FOR PRIMES BENCHMARK

the best case (speedup of ≈ 6 for $n = 12, 24$), parallelism efficiency is always below 60%–value which decrease as number of workers grows up.

6. Case study: `primes-repeat` application

Similar to previous benchmark, `primes-repeat` application test for primes repeatedly over a small range of values. The choice of elements is such that each number takes almost the same time to be tested. For this, we substitute the range presented at line (2) of Routine (1) by a list of numbers to test.

We use R bundle function `rep.int(x, times)` to build a list with *times* repetitions of x value and use as input of benchmark. With a list build by `rep.int(`$5{\times}10^5$`,` $5{\times}10^3$`)` we verified almost linear speedups as shown in Figure (5).

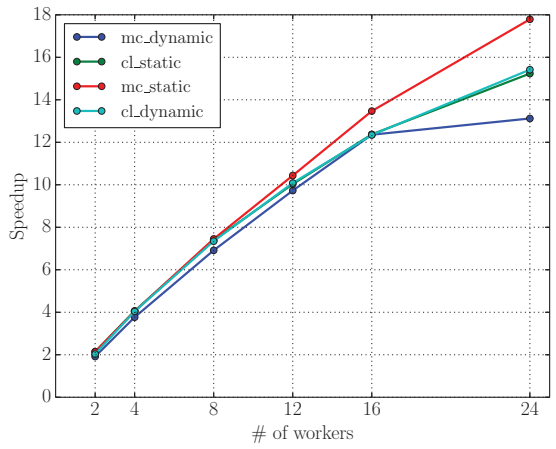


Fig. 5
SPEEDUP VERIFIED FOR PRIMES-REPEAT BENCHMARK

As expected, we observed great efficiency rates for all test cases with static task scheduling, as show in Figure (6). The best results are obtained with `mclapply` with an average 89% of efficiency. Besides, worst results are found with `mclapply` too, but with dynamic task schedule, due to repetitive creation and destruction of worker processes [12]. The remaining results pointed that `clusterApplyLB` and `parLapply` obtained 87% of average efficiency for both static and dynamic task scheduling (2% worse than best results) and up to 16 cores.

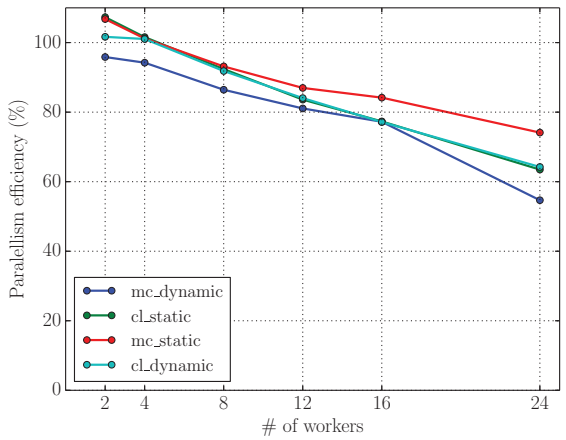


Fig. 6
EFFICIENCY VERIFIED FOR PRIMES-REPEAT BENCHMARK

Increase from 16 to 24 cores do not promote any improvement in computation times. Besides, we verified a efficiency drop of 29% of efficiency for `mclapply` and static work schedule, the test case of best results for `primes-repeat` application.

7. Case study: `firesim` application

`firesim` is a benchmark application based on a Monte Carlo method to simulate fire spreading on a forest. The application perform several trials to estimate the percentage of forest burnt with distinct fire spread probabilities.

The application uses a matricial representation of forest, in which each position of matrix is a tree. Each tree can be in one of the following states: *unburnt*, *smoldering*, *burning* and *burnt*. A tree in *burning* state can spread fire to its neighbors following a 4-connected model.

Each trial starts with one *smoldering* tree. The trial ends after several iterations of 4-connected fire spreading simulation, when there is no trees in *smoldering* and *burning* state.

In our computational experiments we used a 30×30 forest (900 trees), spreading probabilities from 0% to 100%–a total of 10 spreading scenarios, each one repeated 5000 times.

Similar to previous results, we observed low speedup rates–up to 14 times faster than serial implementation–as pointed in Figure 7. The result is shown in Figure 8 indicates less than 60% of parallelism efficiency when all cores of machine be used.

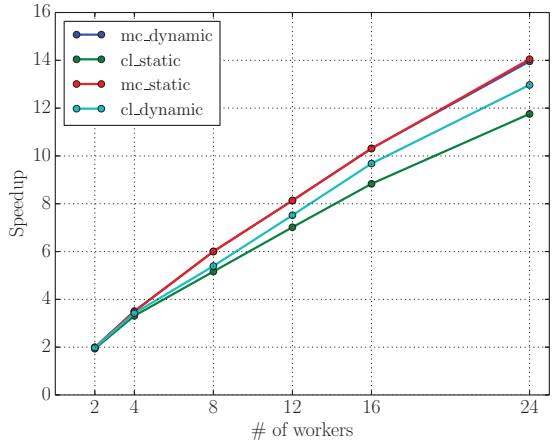


Fig. 7
SPEEDUP VERIFIED FOR FIRESIM BENCHMARK

8. Conclusions

The combination of Caret and multicore packages was effective and achieved satisfactory results with relatively large data samples. The [15] benchmark revealed that the parallel processing feature improves the performance of the tests in most cases. The acceleration, however, does not hold up to eight available cores.

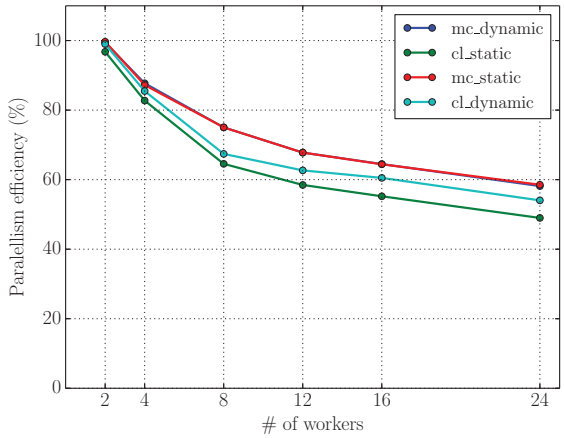


Fig. 8
EFFICIENCY VERIFIED FOR FIRESIM BENCHMARK

The other tests shown that great speedup and parallelism efficiency are achieved with coarse-grained tasks, as well as dynamic job scheduling proved to be ineffective to surpass unbalanced tasks.

It is intended to continue this research, primarily to ascertain the cause of the poor performance with 8 cores. This will only be possible with a deeper analysis of the Caret packages, Multicore and own mining algorithm.

References

[1] W. N. Venables and D. M. Smith, "An introduction to R," 2010, available: <http://cran.r-project.org/doc/manuals/R-intro.pdf>. Accessed April 2016.

[2] M. Schmidberger, M. Morgan, D. Eddelbuettel, H. Yu, L. Tierney, and U. Mansmann, "State of the art in parallel computing with r," *Journal of Statistical Software*, vol. 31, no. 1, pp. 1–27, 8 2009. [Online]. Available: http://www.jstatsoft.org/v31/i01

[3] Q. E. McCallum and S. Weston, *Parallel R*. O'Reilly Media, Inc., 2011.

[4] E. Mahdi, "A survey of r software for parallel computing," *American Journal of Applied Mathematics and Statistics*, vol. 2, no. 4, pp. 224–230, 2014. [Online]. Available: http://pubs.sciepub.com/ajams/2/4/9

[5] S. Urbanek, "multicore: Parallel processing of r code on machines with multiple cores or CPUs," 2009, available: <https://cran.r-project.org/src/contrib/Archive/multicore/>. Accessed April 2016.

[6] L. Tierney, A. J. Rossini, N. Li, and H. Sevcikova, "Simple network of workstations for r," 2003, available: <http://homepage.stat.uiowa.edu/~luke/R/cluster/cluster.html>. Accessed April 2016.

[7] R-core, "Package parallel," 2015, available: <http://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf>. Accessed April 2016.

[8] M. Kuhn, "The Caret package," 2010, available: <http://cran.r-project.org/web/packages/caret/vignettes/caretTrain.pdf>. Accessed April 2016.

[9] B. M. Kursa and W. R. Rudnicki, "Feature selection with the boruta package," *Journal of Statistical Software*, vol. 36, 2010, available: <http://www.jstatsoft.org/v36/i11/paper>. Accessed April 2016.

[10] Togaware, "Rattle: Gnome cross platform gui for data mining using R," 2010, available: <http://rattle.togaware.com/>. Accessed April 2016.

[11] J. S. Racine, "Rstudio: A platform-independent ide for r and sweave," *Journal of Applied Econometrics*, vol. 27, no. 1, pp. 167–172, 2012.

[12] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org

[13] D. L. Eager, J. Zahorjan, and D. Lazowska, "Speedup versus efficiency in parallel systems," *Computers, IEEE Transactions on*, vol. 38, no. 3, pp. 408–423, 1989.

[14] V. P. Kumar and A. Gupta, "Analyzing scalability of parallel algorithms and architectures," *Journal of parallel and distributed computing*, vol. 22, no. 3, pp. 379–391, 1994.

[15] A. Engelhardt, "Benchmarking feature selection with Boruta and Caret," 2010, available: <http://www.r-bloggers.com/benchmarking-feature-selection-with-boruta-and-caret/>. Accessed April 2016.

Scalability of OpenFOAM for Simulations of a Novel Electromagnetic Stirrer for Steel Casting

Isabella Mazza[1] 1*, **Ahmet Duran**[2] 2&, **Yakup Hundur**[3] 3#, **Cristiano Persi**[1], **Andrea Santoro**[1], **Mehmet Tuncel**[2,4]

[1]Ergolines Lab s.r.l., Area Science Park, Padriciano 99, 34149, Trieste, Italy
[2]Mathematical Engineering, Istanbul Technical University (ITU), 34469 Sariyer, Istanbul, Turkey
[3]Physical Engineering, Istanbul Technical University (ITU), 34469 Sariyer, Istanbul,Turkey
[4]Informatics Institute, Istanbul Technical University (ITU), 34469 Sariyer, Istanbul, Turkey

Abstract - *In this work, custom codes were developed for HPC-based magnetohydrodynamics (MHD) simulations, enabling the design of a dedicated electromagnetic stirrer (EMS) for the electric arc furnaces (EAF). The fluid-dynamics of liquid steel within the EAF under the effect of electromagnetic stirring has been studied under different simulation parameters. We performed parallel simulations using an OpenFOAM solver and other related programs on IBM-FERMI (a PRACE Tier-0 system) at CINECA, Italy. We realized performance analysis for the current sequential version and updated parallel versions of the code via extensive simulations. We present and discuss the results of the scalability analysis of the specific codes using two different domain decomposition methods including simple and hierarchic*.

Keywords: HPC, scalability, OpenFOAM, steel casting, magnetohydrodynamics simulations

1 Introduction

The use of state-of-the-art electromagnetic stirrers (EMSs) for steel quality improvement represents a well-established practice in the steelmaking industry ([1], [2]). Besides their employment in steel continuous casting, dedicated EMS can be designed to improve the performance of the electric arc furnace (EAF), where metal scrap is melted at the very first stage of the steel casting process. A state-of-the-art overview of the applications of electromagnetic machines in the steelmaking industry is given in [2]. Numerical simulations of the effects of electromagnetic fields on liquid metals can be performed by various Computational Fluid Dynamics (CFD) codes, including OpenFOAM [3], which solve the equations of Magneto-Hydro-Dynamics (MHD) models. A general review of possible solutions is given by Murawski [4]. More recently, swirl flow velocities are compared in the presence and absence of solidification for the mould EMS system simulations (see Ren et al. [5]). Moreover, a cellular-automaton-finite-element method was used to simulate the solidification structure of a continuous casting large round billet to examine the effect of mold electromagnetic stirring (see Tao et al. [6]). Furthermore, a variational multiscale algorithm was employed to simulate the liquid steel flow in a non-industrial EMS application in order to study the small scale turbulences in [7]. In order to design highly customized EMSs, dedicated codes for MHD simulations need to be implemented.

It is important to conduct research including HPC-based MHD simulations to design a new EMS dedicated to the casting of large blooms. The very first stage of the casting process is the melting of metal scrap into the EAF. Optimal steel melting has a critical impact on the efficiency of the overall casting process both in terms of energy efficiency, costs optimization and productivity. Employment of electromagnetic stirring in the EAF significantly improves the EAF performance by providing several benefits, including improved homogenization of the liquid bath and reduced furnace wear.

* Corresponding author. *E-mail address*: isabella.mazza@ergolines.it
& Corresponding author. *E-mail address*: aduran@itu.edu.tr
# Corresponding author and speaker. *E-mail address*: hundur@itu.edu.tr

The goal of this paper is to conduct HPC-based simulations of the fluid dynamics of liquid steel in the EAF under the effect of electromagnetic stirring. Due to the complexity of the multi-physical system under study, very fine discretization in terms of geometry will be required. The use of HPC and the possibility to take advantage of specialized expertise is therefore key to meet this industrial challenge.

The remainder of this paper is organised as follows: Section 2 presents methodology and results. Section 3 concludes this work.

2 Methodology and results

EMS design has been performed following an iterative, multiple-simulation process including: 1) analysis of the geometrical constraints, 2) calculation of the EM performance, 3) fluid dynamic simulation 4) parameter calibration, 5) iteration of steps 2 to 4 until the required EM performance is achieved.

The project partners at ITU (Duran, Hundur and Tuncel) have prepared sequential job submit scripts and parallel job submit scripts to compile and run OpenFOAM with mathematical operators such as turbulence models and various mesh operators and the solver, and also to execute other related programs on IBM-FERMI at CINECA, Italy. They guided Ergolines for performance and scalability of the codes on HPC system.

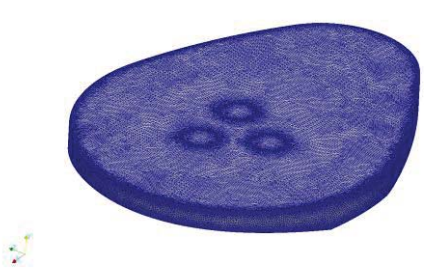


Fig. 1a: EAF geometry and mesh (top view).

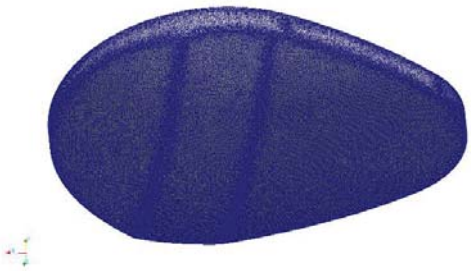


Fig. 1b: EAF geometry and mesh (bottom view).

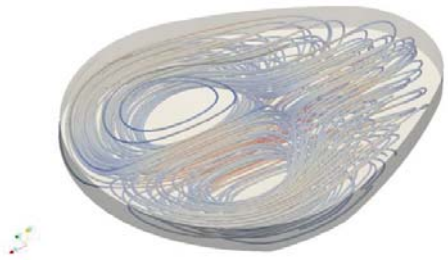


Fig. 1c: Fluid-dynamic simulation: velocity field displayed as flux lines (top view).

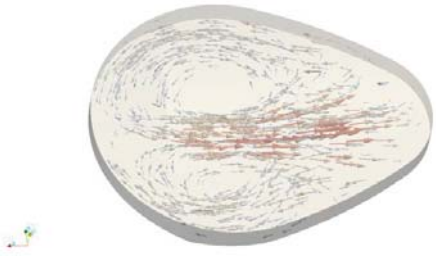


Fig. 1d: Fluid-dynamic simulation: velocity field displayed as vector field (bottom view).

The fluid-dynamics of liquid steel in an electric arc furnace under the effect of electromagnetic stirring has been studied by means of HPC-based numerical simulations. The geometry, mesh and fluid dynamics of the system under study are represented in Figures 1a-1d. The velocity field generated by the EMS, which is located under the EAF, is also shown.

The magnetic field produced by the stirrer and the force field induced into the steel has been computed by means of Comsol Multiphysics in order to calculate the initial conditions for the fluid-dynamic simulations. The stationary magnetic and force fields have then been used as initial conditions for the fluid dynamic simulations, carried out in OpenFOAM code (C++, MPI) where OpenFOAM (see [3]) is an open source CFD toolbox. A series of fluid dynamic simulations has been performed by considering a stationary magnetic field. A mesh of 3 million elements was used. Ergolines' proprietary solver, implemented in OpenFOAM, has been compiled on the CINECA FERMI supercomputer. GNU 4.4.6 C++ compiler has been used because the related libraries were compiled via GNU by CINECA. Specifically, in order to simulate the effects of electromagnetic stirring on liquid steel, a dedicated customization of Ergolines' current OpenFOAM code has been implemented so as to couple Electromagnetism with Fluid Dynamics. The simulations in this work are obtained using the OpenFOAM 2.1.1.

In order to better assess how parallelisation improves computational performance, the simulations have been carried out by considering an increasing number of processors. All the simulations were run over 200 iterations: in fact, this figure represents a good trade-off between computational times and statistics, since it produces enough data to carry out a sound statistical analysis while maintaining at the same time an acceptable computational time.

In addition, two different domain decomposition methods have been compared:

- The "simple" method, which generates a mesh where the number of elements per unit volume is in general not the same for all the elements;
- The "hierarchical" method, which generates a mesh where the number of elements per unit volume is constant in the whole domain. This approach enables to efficiently distribute the computational load between the cores.

While the first approach is best suited for simple geometries, the latter one is more convenient when dealing with complex domains. We observe that the simple decomposition is not a suitable strategy for the case using 256 cores possibly due to the unbalanced computational load distribution in Table 1.

The results of the simulations are reported in Table 1, where performance is quantified in terms of speed-up and computational time. The BlueGene/Q (FERMI) configuration (see [8]) is made of 10 racks such as 2 racks having 16 I/O nodes per rack, implying a minimum job allocation of 64 nodes (1024 cores) and 8 racks having 8 I/O nodes per rack, implying a minimum job allocation of 128 nodes (2048 cores). In other words, each node contains 16 cores.

Figure 2.a shows the measured speed-up as a function of the number of cores used. The ideal trend is linear and it is displayed as a green line for comparison. The blue and red lines represent the measured trends based on the hierarchic and simple methods, respectively.

Table 1. Data displayed in Figures 2 and 3. Legend: *# of nodes*: number of nodes allocated on CINECA FERMI (minimum 64). Each node has 16 cores at most. *# of cores*: number of cores used in the simulation. *# of iterations*: number of iterations. *Decomposition*: domain decomposition methos and strategy (different strategies were used for the same method). *Computation time*: time of the simulation (seconds). *Speed-up*: speed-up as a function of the number of cores, calculated as the ratio of the time with "n" cores over the time with 20 cores (20 cores has been considered as the normalization factor).

# of nodes	# of cores	# of iteration	Decomposition	Computational time (s)	Speed-up
64	256	200	not suitable with simple decomposition strategy		
64	128	200	simple 8x4x4	9520	3.04
64	64	200	simple 4x4x4	15907	1.82
64	**20**	**200**	**simple 2x2x5**	**28988**	**1.00**
256	1024	200	hierarc. 32x4x8 xzy	1241	25.96
256	1024	200	hierarc. 16x8x8 xzy	1277	25.23
128	1024	200	hierarc. 16x8x8 xzy	1304	24.71
128	1000	200	hierarc. 10x10x10 xzy	1247	25.84
128	900	200	hierarc. 10x10x9 xzy	1231	26.18
128	800	200	hierarc. 10x10x8 xzy	1254	25.70
128	600	200	hierarc. 10x10x6 xzy	1412	22.82
64	512	200	hierarc. 8x8x8 xzy	1477	21.82
64	512	200	hierarc. 16x4x8 xzy	1544	20.87
64	400	200	hierarc. 10x10x4 xzy	2005	16.07
64	256	200	hierarc. 8x4x8 xzy	2830	11.39
64	128	200	hierarc. 8x4x4 xzy	5479	5.88
64	64	200	hierarc. 8x2x4 xzy	10613	3.04
64	**20**	**200**	**hierarc. 5x2x2 xzy**	**32222**	**1.00**

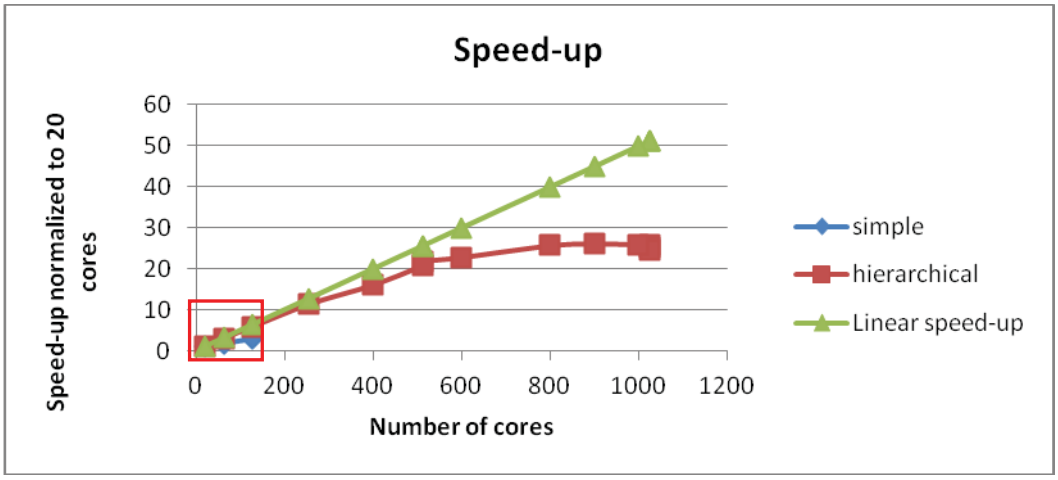


Fig. 2a: Speed-up as a function of the number of cores.

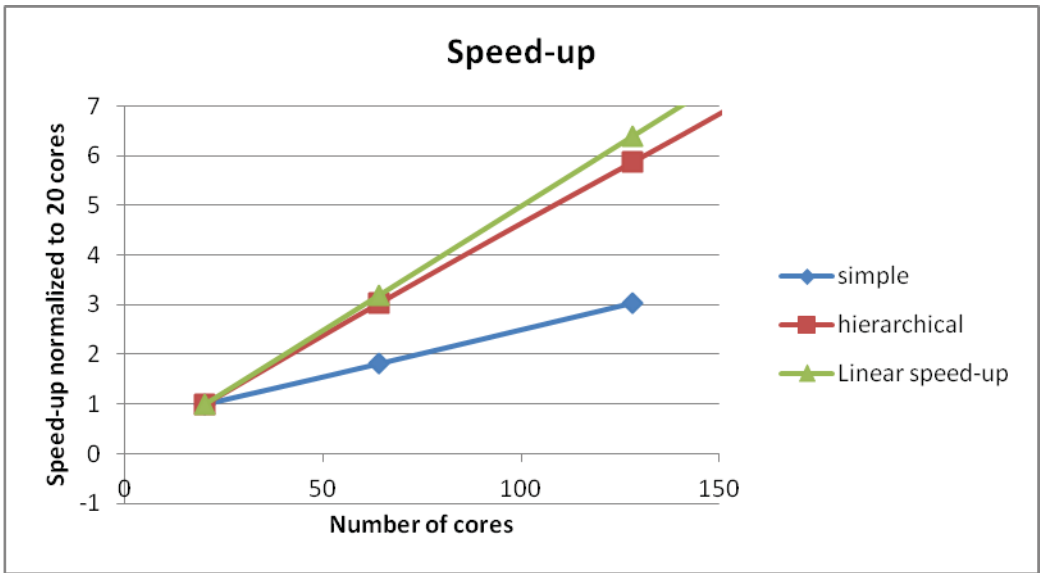


Fig. 2b: Zoom on the region of the graph in Fig. 2a enclosed in a red rectangle: linear regime.

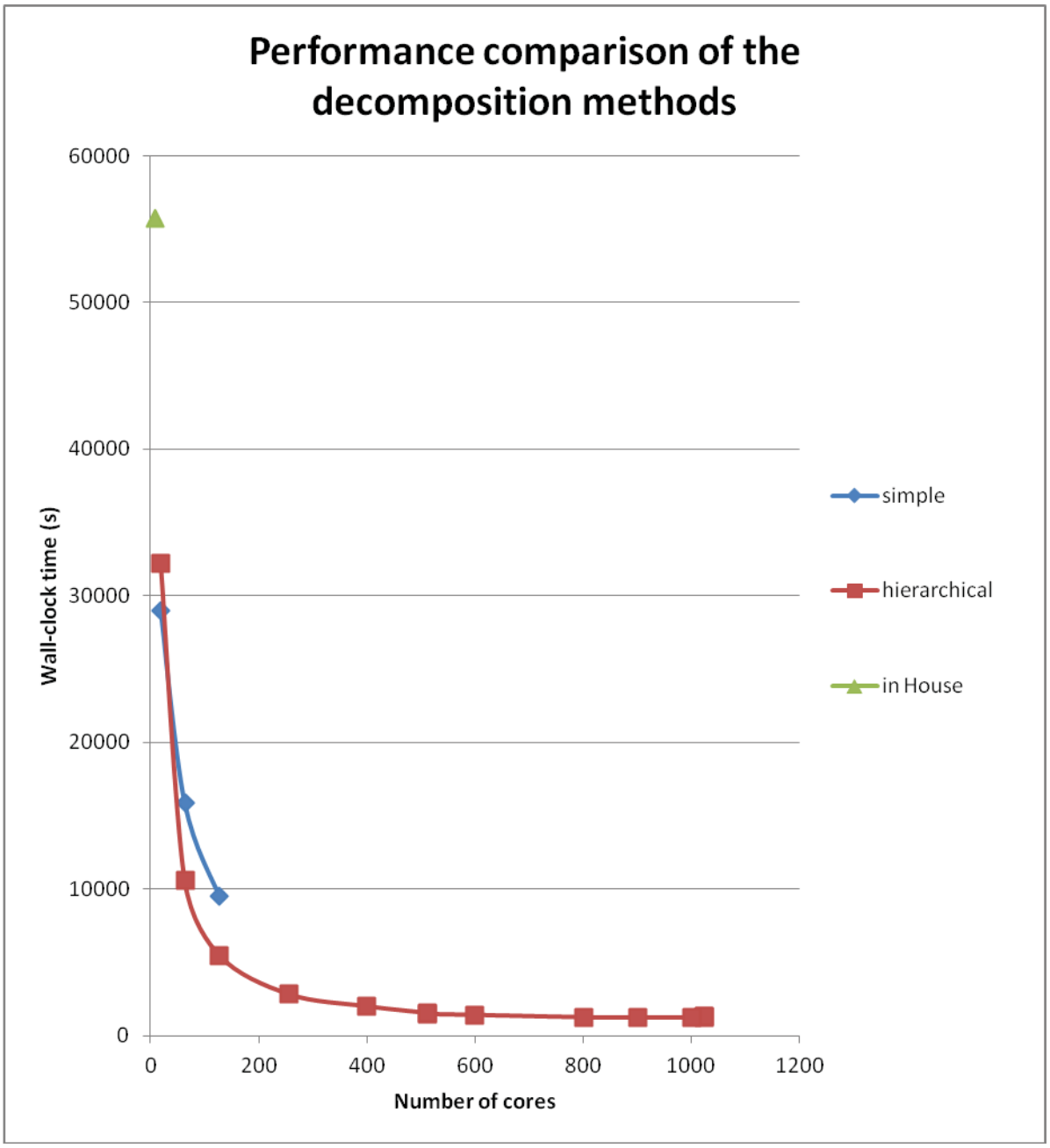


Fig. 3: Wall-clock time as a function of the number of cores.

Figure 2.a displays almost a linear speed-up till 512 cores. Part of the linear region is enlarged in Figure 2.b, which shows the superior performance of the hierarchic decomposition, close to the ideal trend, with respect to the simple method. In fact, since the shape of the domain representing the furnace is quite complex, the hierarchic decomposition enables to distribute the mesh elements and computational load much more efficiently than the simple method. In addition, the simple method fails to decompose the domain if more than 128 cores are used. Based on this result, in all the successive simulations only the hierarchical method was used.

After 512 cores, the speed-up growth-rate gradually decreases and saturation is reached. When a large number of cores are used, the number of elements per core decreases, reducing the computational time, but the communication overhead dominates because more cores need to communicate with each other. Since this figure is characteristic of the system, it does not improve so much by increasing the number of cores, thus causing the speed-up to saturate. Moreover, we observe a gradual speed-up up to 1024 cores with oscillations depending on the decomposition and the memory usage of each core in Table 1. For example, the simulation took 1241 seconds with the hierarchical decomposition mesh of 32x4x8 and 1024 cores as having advantage where 4 cores are used per node instead of 8 cores per node so that a larger memory can be provided.

3 Conclusions

We conducted research and prepared codes/scripts for HPC-based magnetohydrodynamics simulations for designing an electromagnetic stirrer. We performed parallel simulations using the OpenFOAM, solver and other related programs on IBM-FERMI at CINECA. We obtained that the solver with hierarchical decomposition method scales for a mesh domain having 3000 K elements, on FERMI, CINECA. We observed almost a linear speed-up up to 512 cores and then a gradual speed-up up to 1024 cores. Moreover, the matrices having larger order coming from the finer meshes may require a higher saturation point for the optimal minimum number of cores (see [9]). Thus, the code is suitable for the BlueGene/Q (FERMI) system.

The fluid-dynamics of liquid steel in an electric arc furnace under the effect of electromagnetic stirring has been studied by means of HPC-based numerical simulations. The velocity field was generated by the EMS.

As a conclusion, the use of HPC for steel casting provided a dramatic advantage and enabled to carry out an extensive analysis of the fluid-dynamic of the liquid steel in the furnace under the influence of electromagnetic stirring, providing key information for EMS design and industrialization.

Acknowledgements

This work was supported by the PRACE project funded in part by the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement RI-312763, by the EU's Horizon 2020 research and innovation programme (2014-2020) under grant agreement 653838, and by the Project 2010PA3012 awarded to access to IBM-FERMI at CINECA, Italy under the 21st Call for PRACE Preparatory Access.

4 References

[1] Norbert Vogl, Hans-Jürgen Odenthall, and Markus Reifferscheid, Fluid flow in continuous casting affected by electromagnetic fields, 6th International Congress on Science and Technology of Steelmaking, Beijing, May 2015.

[2] Brian G. Thomas and Rajneesh Chaudhary, State of the art in electromagnetic flow control in continuous casting of steel slabs: Modelling and plant validation, 6th Int. Conference on Electromagnetic Processing of Materials EPM 2009, Oct. 2009.

[3] OpenFOAM main site, http://www.openfoam.com

[4] Kris Murawski, Numerical solutions of magnetohydrodynamics equations, Bulletin of the Polish Academy of Sciences, Technical Sciences, 59(2), 2011.

[5] B-Z. Ren, D-F. Chen, H-D. Wang, M-J. Long, and Z-W. Han, Numerical simulation of fluid flow and solidification in bloom continuous casting mould with electromagnetic stirring, Ironmaking & Steelmaking, 42(6), 401−408, 2015.

[6] Tao Sun, Feng Yue, Hua-jie Wu, Chun Guo, and Ying Li, and Zhong-cun Ma, Solidification structure of continuous casting large round billets under mold electromagnetic stirring, Journal of Iron and Steel Research, International, 23(4), 329−337, 2016.

[7] Marioni Luca, Jose Alves, François Bay, and Elie Hachem, Effect of m-ems on in-mould transient flow during continuous casting, the Proceedings of Int. Conference on Heating by Electromagnetic Sources, Italy, May 2016.

[8] http://www.cineca.it/en/content/fermi-bgq

[9] Ahmet Duran, M. Serdar Celebi, Senol Piskin, and Mehmet Tuncel, Scalability of OpenFOAM for bio-medical flow simulations, Journal of Supercomputing, 71(3), 938−951, 2015.

Parallel Kernel K-Means on the CPU and the GPU

Mohammed Baydoun[1], **Mohammad Dawi**[1], **and Hassan Ghaziri**[1]
[1]Beirut Research and Innovation Center, Beirut, Lebanon

Abstract – *K-Means is probably the leading clustering algorithm with several applications in varying fields such as image processing and patterns analysis. K-Means has been the basis for several clustering algorithms including Kernel K-Means. In machine intelligence and related domains Kernelization transforms the data into a higher dimensional feature space by calculating the inner products between the different pairs of data in that space. This work targets Kernel K-Means and presents parallel implementations of the clustering algorithm using CUDA on the GPU and OpenMP, Cilk-Plus and BLAS on the CPU. The implementations are tested on different datasets leading to different speedups with CUDA achieving the faster runtimes.*

Keywords: Kernel K-Means; clustering; CUDA; OpenMP; BLAS

1 Introduction

Clustering consists of finding partitions of a data set such that similar data points are grouped together. This definition is vague on purpose. In fact, there is no standard definition to clustering and it remains at the end a subjective procedure allowing the end user to detect some hidden structures or regularities inside the data set. Clustering has a wide variety of applications such as data mining, pattern recognition, knowledge discovery, text mining, and many others [1].

Definitely, there are several clustering algorithms and perhaps the most famous one is K-Means due to its simplicity and behavior. K-Means has had several modifications and amongst the notable ones are those that utilize Kernelization which are generally termed Kernel K-Means. These methods are used to bypass some of the limitations of K-Means related to data representations. Kernel Methods are mainly used since the data in its raw representation often needs to be explicitly transformed into feature vector representation via a user-specified feature map. Kernel methods are relatively computationally cheap methods for achieving this objective using a selected kernel, i.e., a similarity function over pairs of data points in raw representation. Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional implicit feature space [2].

High Performance computing is a major area of research. In particular, parallel programming leads to implementing a parallel version of the required algorithm while targeting a suitable hardware. The aim is essentially speeding up the performance of the algorithm in comparison with other hardware. While a sequential or traditional program runs serially, a parallel program utilizes the independences or parallelizable parts of the algorithm to speed-up the performance taking into consideration the used hardware and its limitations. It is worth mentioning that the speedup of any program is limited by its sequential part according to Amdahl's law.

Applications to parallel computing and programming are limitless since there are limitless algorithms and various parallel architectures or platforms. These include Field Programmable Gate Arrays (FPGA), Graphics Processing Units (GPU), Multi-Core Central Processing Units (CPU), Networks or Clusters, ASICs, and others developed for the main purpose of enhancing performance of various algorithms.

In regards to CPUs, several tools exist and perhaps the most common one is Open Multi Processor (OpenMP). Others include PThreads and Cilk Plus. Concerning GPUs, the main available tools are General Purpose Programming on the GPU (GPGPU), Open Computing Language (OpenCL), and Compute Unified Device Architecture (CUDA).

This work addresses using the CPU and implements Kernel K-Means using OpenMP and Cilk Plus. In addition, we target the GPU and provide the Kernel K-Means implementation on an Nvidia CUDA capable GPU.

Also, this work considers various artificial and real datasets having different numbers of patterns and features without accounting for big data cases which is definitely an important issue but is a problem on its own and it can be targeted in future works. It is important to note that in order to obtain accurate comparisons; this work optimizes the serial and the parallel versions of the algorithm on the handled architectures.

The remainder of the paper discusses the different ideas regarding the proposed Kernel K-Means implementations. In section II, we provide a brief review of previous literature on the related subjects. Afterwards, section III explains the Kernel K-Means clustering algorithm. Section IV details the serial and parallel CPU related implementations and section V dwells on the CUDA GPU implementation. After that,

section VI presents the main results and the last section provides the conclusion.

2 Literature Review

K-Means have been the subject of much research. Kernel K-Means is part of this research, where works concentrate on the accuracy of the clustering algorithm. The main ideas related to Kernel K-Means were discussed in [2]. Other ideas were discussed in different research. The work in [3] added a kernel step to the global K-Means algorithm discussed in [4] to deal with nonlinearly separable clusters. In [4] the work proposed a way to eliminate high sensitivity of K-Means to the initial guess. In [3], the authors also presented a fast version to get comparable running times. In [5], the authors addressed implementing the Kernel K-Means algorithm on large data. They discussed changing the clustering order from the sequence of the sample to the sequence of the Kernel, which enabled an efficient way of handling the Kernel Matrix along with using all the available disk space. The work in [5] aimed at improving the speed of the kernel k-means by using a new kernel function termed conditionally positive definite kernel (CPD). They mentioned getting running times superior to that of the K-means clustering algorithm on artificial and real data. In regards to parallelizing the Kernel K-Means algorithm, previous work is lacking and therefore, we consider the literature related to parallel implementations of K-Means.

The work in [6] implemented several algorithms on the GPU and K-Means was one of them. Moreover, the work in [7] discussed the first implementation on a CUDA device by mainly parallelizing the part accounting for the minimum distance calculation. In [8], the work provided a detailed CUDA implementation of K-Means and discovered that a speedup of 14 times is possible in comparison to a single threaded CPU implementation. Also, MapReduce was used in several works in order to present a parallel K-Means algorithm such as the algorithm in [9]. Additionally, the work in [10] considered the parallel implementation of K-Means on MPI, OpenMP and CUDA and concluded that for relatively small data, OpenMP performs the best, while for larger data, CUDA obtains the best speedup.

In [11], the objective was to utilize the advantages of both MPI and OpenMP to parallelize the K-Means algorithm, namely parallel processing inside the node and distribution of tasks inside the cluster obtained through MPI. The latter work showed good speedups especially on large datasets. Moreover, [12] focused on distributed ways to parallelize K-Means and proved scalability over large datasets.

Thus, and despite the lack of any parallel implementation of Kernel K-Means, there are lots of relevant works on the parallel implementation of the K-Means algorithm. So, the main contribution of this work lies in being the first parallel implementation, as far as we know, of Kernel K-Means using the targeted architectures or tools.

3 Kernel K-Means

Kernel K-Means utilizes the Kernelization approach to divide given data into a set of clusters using an approach that is mainly based on K-Means.

First, it is important to provide a basic idea about Kernelization. This is a common approach that is used with various algorithms including PCA, SOM, etc… It can be explained in the following.

Given a set S in the input space, its image is $\phi(S)$ in the feature space. The center of mass of set $\phi(S)$ is the vector given by:

$$\phi_{mean}(S) = \frac{1}{l}\sum_{i=1}^{l}\phi(x_i) \tag{1}$$

As with all points in the feature space we will not have an explicit vector representation of this point. However, a point x_{mean} whose image under ϕ is $\phi_{mean}(S)$ may not exist. In other words, we are now considering points that potentially lie outside $\phi(X)$ that is the image of the input space X under the feature map ϕ.

Despite this apparent inaccessibility of the point $\phi_{mean}(S)$, we can compute its norm using only evaluations of the kernel function on the inputs:

$$\begin{aligned}\left\|\phi_{mean}(S)\right\|^2 = \left\langle \phi_{mean}(S), \phi_{mean}(S)\right\rangle &= \left\langle \frac{1}{l}\sum_{i=1}^{l}\phi(x_i), \frac{1}{l}\sum_{j=1}^{l}\phi(x_j)\right\rangle \\ &= \frac{1}{l^2}\sum_{i,j=1}^{l}\left\langle \phi(x_i),\phi(x_j)\right\rangle \\ &= \frac{1}{l^2}\sum\nolimits_{i,j=1}^{l} k(x_i,x_j)\end{aligned} \tag{2}$$

Where l is the number of samples in the set $\phi(S)$, or in other words it is the number of samples per class or cluster.

Hence, the square of the norm of the center of mass is equal to the average of the entries in the kernel matrix.

Incidentally this implies that this sum is greater than or equal to zero, with equality if the center of mass is at the origin of the coordinate system. Similarly, we can compute the distance of the image of a point x from the center of mass $\phi_{mean}(S)$.

$$\begin{aligned}\left\|\phi(x)-\phi_{mean}(S)\right\|^2 &= \left\langle \phi(x)-\phi_{mean}(S), \phi(x)-\phi_{mean}(S)\right\rangle \\ &= \left\langle \phi(x),\phi(x)\right\rangle - 2\left\langle \phi(x),\phi_{mean}(S)\right\rangle + \left\langle \phi_{mean}(S),\phi_{mean}(S)\right\rangle \\ &= k(x,x) - \frac{2}{l}\sum\nolimits_{i=1}^{l} k(x,x_i) + \frac{1}{l^2}\sum\nolimits_{i,j=1}^{l} k(x_i,x_j)\end{aligned} \tag{3}$$

In regards to clustering in general and K-Means in particular, the main aim is to determine the clusters of a certain input set X = {x1,...xN}.

We first provide a basic explanation of K-Means. K-Means starts with the data randomly labeled according to a selected number of clusters. Then, and on an iterative basis, the label of each pattern is chosen based on the distance from the center of the cluster which is constantly updated until convergence.

For Kernel K-Means, the data is initially transformed into the Kernel feature space using a predefined kernel function such as the ones mentioned in the following equations:

$$k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \text{ (Gaussian)} \quad (4)$$

$$k(x,y) = \exp\left(-\frac{\|x-y\|}{2\sigma^2}\right) \text{ (Radial Basis)} \quad (5)$$

$$k(x,y) = \tanh(\alpha x^T y + \beta) \text{ (Sigmoid)} \quad (6)$$

$$k(x,y) = 1 - \sum_{n=1}^{N} \frac{(x_i - y_i)^2}{\frac{1}{2}(x_i + y_i)} \text{ (Chi-Square)} \quad (7)$$

In this work, we only utilize the Gaussian Kernel noting that other kernels can be performed in a similar manner without greatly affecting the performance of the algorithm especially that this only affects the first phase of the algorithm (Phase I), where the Kernel Matrix needs to be computed. It is worth emphasizing that this phase can be used in several other Kernel methods such as Kernel SOM [13] and Kernel PCA, so its implementation is generally useful.

In order to provide a complete explanation of the Kernel K-Means algorithm, the pseudo code of the algorithm is provided noting that we divide the algorithm into two phases.

Given a training set of *N* samples $\leftarrow X_1,.....,X_N$

(Phase I) $K_{NxN} \leftarrow$ Kernel Matrix:

$K(i,j) \leftarrow \langle \phi(X_i), \phi(X_j) \rangle = \exp\left(-\frac{\|X_i - X_j\|^2}{2\sigma^2}\right)$ (If the Gaussian Kernel is used)

Number of clusters $\leftarrow k$, $cluster\ centers (C_1,.....,C_k)$

Distance matrix: D (*N* rows, *k* columns),

$D(i,j) = \left\|\phi(X_i) - \phi(C_j)\right\|^2$

Randomly assign each pattern to a cluster: Labels matrix A (*N* rows, *k* columns), label rule:

$A_{ij} = \begin{cases} 1 & if\ X_i\ in\ cluster\ j; \\ 0 & otherwise \end{cases}$

change $\leftarrow$ 1
while change ≠0 do
(Phase II) Iterative
for j=1:*k* do
calculate the cluster size *l*
for i=1:*N* do
calculate distance of each sample from the mean of the cluster in feature space using:

$$\phi(C_j) = \frac{1}{l}\sum_{s=1}^{l} A_{ij}\phi(X_i)$$

$$D(i,j) = K(X_i,X_i) - \frac{2}{l}\sum_{m=1}^{N} K(X_i,X_m) + \frac{1}{l^2}\sum_{m=1}^{N}\sum_{n=1}^{N} A_{mj}A_{nj}K(X_m,X_n)$$

end for
end for
old A = A
update labels of all samples according to minimum distance
A =indices(minimum of D along the columns)
if old A = A then
change=0
end if
end while

4 CPU Implementation

This work mainly tested the algorithm using C on different artificial and real datasets with varying number of patterns and features without accounting for big data cases since this is a domain on its own and should be the subject of future work.

We optimized both the serial and the parallel implementations on a multicore CPU to attain the highest possible speedups whilst ensuring the correctness of the results.

Concerning the serial version, the implementation relies on using a Matrix-like form that utilizes functions similar to matrix and vector operations which is generally optimal in Matrix Based Software such as Matlab. This matrix like form is possible due to the nature of the problem as can be observed in the pseudo code. For example, phase I is relatively similar to the matrix multiplication, but by replacing the multiplication with a subtraction, squaring and division followed by the exponential calculation. Also, the distance calculation involves a matrix multiplication for calculating the second term in a single kernel for all the values followed by multiplying by (2/*l*). So, these can be directly optimized.

Thus, in the C sequential version, we initially implemented a direct serial version and afterwards used and modified the Open-BLAS library [14], where BLAS stands for Basic Linear Algebra Subprograms, in order to optimize the implementation using matrix-like operations. Open-BLAS includes one of the fastest sequential and parallel matrix multiplication algorithms, so we implemented similar approaches that suit the Kernel-K Means algorithm which helped achieve the faster serial solution.

Concerning the parallel multicore CPU implementation, we implemented the Kernel K-Means algorithm in C using two very common libraries. These are the OpenMP and Cilk-Plus parallel libraries. Also, and due to the matrix properties of the problem, we used Open-BLAS which allows for selecting the number of threads and is therefore parallel. The OpenMP and Cilk Plus implementations are relatively similar since we basically parallelized the sequential version by dividing the threads according to the number of patterns, which allows for the highest level of parallelism in this algorithm. In regards to the BLAS related versions, we based on the parallel BLAS version in addition to using OpenMP when BLAS was not applicable or even when it achieved lower times to obtain the fastest possible times.

5 CUDA Implementation

The CUDA implementation is rather similar to the parallel CPU one but with important differences that require explanation.

Initially, CUDA requires transferring the required data to the GPU, which is in this case the dataset itself including the patterns and the features in addition to the initial random labels, although these can be initiated on the GPU. Definitely, these variables and others as mentioned in the pseudo code require to be allocated in the GPU memory, so this can be termed (Phase 0) and is relatively time consuming depending on the size of the data. In general, memory transfers can be a bottleneck in GPU implementations, except when used with multi-streams, but this is not applicable here, since the complete data is required from the start of the Kernel K-Means algorithm.

After transferring the data, the implementation requires computing the Kernel Matrix, which is similar to matrix multiplication as already mentioned in the previous section. Thus, this can be performed using a kernel similar to the (cublas) kernels that are already available with CUDA, or to the kernels provided in the CUDA sample codes, which is adopted here using a modified version of the matrix multiplication kernel. This is probably the best possible implementation according to the CUDA guidelines.

Afterwards, the iterative phase needs to be performed. The first part accounts for counting the size of each cluster, which is simply performed using the atomicAdd function. Then, one needs to compute the distance value for each pattern with each cluster. This is composed of three terms. The first term is from the Kernel Matrix, which is already computed. The second term can be computed for all the patterns and the clusters in a similar operation to a single matrix multiplication operation as already noted in the serial section. The last term is common for each cluster meaning that there can be only a number of clusters' terms. Thus, it needs to be computed only once for all the patterns and used according to each cluster. This term is composed of two summation steps and is computed using a reduction operation. Reduction is usually used for summations and in this case we have summation of several variables that are a collection of multiplied values, which means reduction can be used. Like other matrix operations, reduction is provided in the CUDA sample codes, so these were modified to suit the calculation of this term.

After obtaining the three terms, the distance matrix can be computed, where each value is computed by a single CUDA thread while ensuring that the accesses to the variables in the memory are coalesced to achieve better speedups.

Afterwards, the minimum distance and the corresponding cluster for each pattern need to be determined. Like the distance matrix computation, this is done for each pattern in a single CUDA thread and this leads to the new labels matrix whilst checking if convergence was achieved through the Boolean "change" variable. This completes the second phase, which is repeated for a certain number of iterations until convergence is achieved.

After the completion of the algorithm, the final labels need to be transferred back to the host from the GPU, which completes the implementation.

Thus, the proposed CUDA implementation mainly relies on modifying tasks that are generally provided with the CUDA sample codes in order to achieve the best possible speedups.

6 Results

The Kernel K-Means algorithm, and as already noted in the pseudo code can be divided into two distinct phases. The first phase (PI) calculates the Kernel matrix (K), and the second phase (PII) is part of (the while loop) and is thus an iterative procedure that is required to update the distances, and obtain the clusters and the labels of the patterns. (PII) is performed for a number of times until the convergence of the algorithm.

Moreover, we can consider a phase zero (P0), which is necessary for allocating the data and copying the required values only in the CUDA case. P0 accounts for copying the data to and from the GPU.

Thus, we need to report on the results of these different phases by considering different datasets. So, we considered artificial and real datasets as noted in Table 1. Several of these datasets are shown in Figure 1. Besides the ones in the figure, we generated a Gaussian random dataset of four clusters where each pattern has three features. Also, for the MNIST dataset [15], we utilized the test data for the images of “0” and “1” which means that there are only two clusters.

Table 1. Timing Results for the different implementations

Dataset	Patterns	Features	Clusters	C (ms)		C+BLAS (ms)		OpenMp (ms)		Cilk (ms)		MP+BLAS (ms)		CUDA		
				PI	PII	PI	PII	PI	PII	PI	PII	PI	PII	P0	PI	PII
Circular	1012	2	2	13.2	3.8	4.3	3.3	5.4	1.9	4.2	1.3	1	1.9	1.1	0.17	0.6
Full Moon	1000	2	2	11.8	4.1	2.5	3.3	5.8	1.8	4.9	1.2	1	1.8	1.1	0.17	0.6
Atom	800	2	2	9.2	2.5	1.9	2.2	3	1.5	3.4	1.3	0.5	1.5	1	0.12	0.5
Chainlink	1000	3	2	12.9	3.8	2.2	3.4	5.1	2	3.4	1.6	0.9	2	1.1	0.19	0.6
Spiral	2000	2	2	60	44	18	43	15	11	15	14	9.3	25	2.9	0.65	0.5
Gaussian	2000	3	4	50	32	12.2	21	14	10	13	8.2	8.8	14	7.7	1.7	0.9
EngyTime	4096	2	2	198	65.5	69.3	59.1	54.9	22	56.4	21.4	36.9	38	10	2.7	1.1
Mnist (0,1)	2115	400	2	1877	18	91.1	15.7	356.6	6.6	359.5	6.3	37	6.6	5.4	0.92	0.6

The results are presented in Table 1 for the various implementations using the different datasets. The table shows the time in milliseconds. All the CPU related tests were performed using a 2.3GHz i7 Intel CPU with 4 cores. The GPU implementation was performed on a Tesla C2050 device.

It is worth comparing the obtained times of Kernel K-Means with that of a serial C K-Means implementation and although the timing is not mentioned, all the times for K-Means are less than 0.5 milliseconds, which indicates that Kernel K-Means is much more time consuming.

Besides, it is important to note that the CPU implementation varied according to the used CPU and the number of available cores whose increase directly meant higher speedups although this is not detailed here. Moreover, the CUDA implementation directly depends on the used GPU where a newer GPU should yield better results. The discussion related to such ideas should be provided in future works.

In regards to the errors and the accuracy of K-Means versus Kernel K-Means, there is a drastic difference with Kernel K-Means being much more accurate and having near 100% accuracy in all the cases except the Gaussian and Spiral case while the K-Means accuracy is generally much lower except when the data is linearly separable.

The results indicate that computing phase I and phase II is clearly faster on the CUDA device and even if the memory timings or phase zero is accounted for, the CUDA device still proves to be the faster albeit not by much when compared with the OpenMP+BLAS version except in relatively larger datasets such as MNIST and Energy Time.

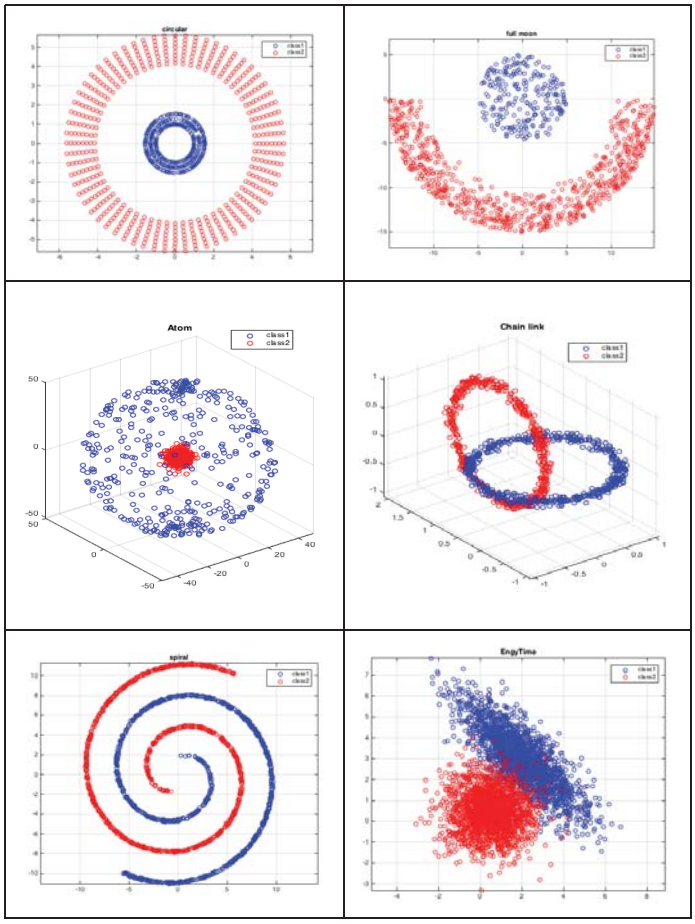


Figure 1. Circular, Moon, Atom, Chainlink, Spiral and Energy-Time Datasets

7 Conclusion

This work presented several parallel implementations of the Kernel K-Means algorithm using the CPU and the GPU. The CPU implementations involved using Cilk-Plus and OpenMP in addition to OpenBLAS due to the matrix-like nature of the problem, while the GPU used CUDA and the available sample codes. The CUDA implementation proved to yield the faster

results despite the relatively high time consumption caused by memory allocation and data transfers between the CPU and the GPU. The work also notes that there is a large room for further improving the parallel implementations with emphasis on large data.

8 References

[1] C. C. Aggarwal and C. K. Reddy, Data Clustering: Algorithms and Applications. CRC Press, 2013.

[2] J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis. Cambridge university press, 2004.

[3] G. F. Tzortzis and A. C. Likas, "The global kernel-means algorithm for clustering in feature space," Neural Networks, IEEE Transactions on, vol. 20, pp. 1181-1194, 2009.

[4] G. Tzortzis and A. Likas, "The global kernel k-means clustering algorithm," in Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on, 2008, pp. 1977-1984.

[5] R. Zhang and A. I. Rudnicky, "A large scale clustering scheme for kernel k-means," in Pattern Recognition, 2002. Proceedings. 16th International Conference on, 2002, pp. 289-292.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," Journal of Parallel and Distributed Computing, vol. 68, pp. 1370-1380, 2008.

[7] R. Farivar, D. Rebolledo, E. Chan and R. H. Campbell, "A parallel implementation of K-Means clustering on GPUs." in Pdpta, 2008, pp. 212-312.

[8] M. Zechner and M. Granitzer, "Accelerating K-Means on the graphics processor via cuda," in Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on, 2009, pp. 7-15.

[9] W. Zhao, H. Ma and Q. He, "Parallel K-Means clustering based on mapreduce," in Cloud ComputingAnonymous Springer, 2009, pp. 674-679.

[10] J. Bhimani, M. Leeser and N. Mi, "Accelerating K-Means clustering with parallel implementations and GPU computing," in High Performance Extreme Computing Conference (HPEC), 2015 IEEE, 2015, pp. 1-6.

[11] L. M. Rodrigues, L. E. Zárate, C. N. Nobre and H. C. Freitas, "Parallel and distributed kmeans to identify the translation initiation site of proteins," in Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on, 2012, pp. 1639-1645.

[12] K. Stoffel and A. Belkoniene, "Parallel k/h-means clustering for large data sets," in Euro-Par'99 Parallel ProcessingAnonymous Springer, 1999, pp. 1451-1454.

[13] K. W. Lau, H. Yin and S. Hubbard, "Kernel self-organizing maps for classification," Neurocomputing, vol. 69, pp. 2033-2040, 2006.

[14] http://www.openblas.net/

[15] Y. LeCun, C. Cortes and C. J. Burges, The MNIST Database of Handwritten Digits, 1998.

On Optimization of Parallel Communication-Avoiding Codes for Multicore Architectures

Emna Hammami, Yosr Slama
University of Tunis El Manar, Faculty of Sciences of Tunis,
University Campus - 2092 Manar II, Tunis, Tunisia

Abstract - *Parallelizing polyhedron programs (PP) i.e. programs structured in nested loops with affine bounds is still a matter of focus for a lot of research works due to its practical interest in scientific applications. Indeed several efficient code generation tools (CGT) from PP's have been proposed in the literature such as Pluto. Despite its performance, these CGT's rarely take into consideration the target architecture (TA) which may be a source of performance improvement. We precisely address this aspect i.e. adapting code generation from a PP to a given TA, namely a multicore machine in order to enhance the efficiency of the generated parallel communication-avoiding codes. For this purpose, we propose a two-phase approach. The first consists, for given TA and PP, in fairly distributing the parallel tasks involved by the PP onto the TA cores. The second improves data locality through cache memory optimization. In order to evaluate the interest of our contribution, we carried out a series of experiments targeting a quadcore bi-processor machine and choosing three PP's often encountered in scientific applications, namely matrix addition, matrix-vector product and matrix-matrix product.*

Keywords: Allocation, cache memory, code optimization, multicore architecture, polytope model, Prefetching.

1 Introduction

Parallel computing may be performed on different parallel platforms. Recently, many studies have focused on multicore machines which are nowadays the most used components of supercomputers, clusters and grids. Following the recent architectures development, it becomes necessary to adapt existing parallelization methods and tools to the new architectures by taking into account the target machine (TA) characteristics, e.g. cache memory organization and size, cores' speed, etc. It is in this general context that our work focuses especially on polyhedron programs (PP) i.e. programs structured in nested loops with affine bounds, which are the most used codes in scientific computing. Since code generation (CG) is the last phase of automatic parallelization based on the polytope model, we are interested in parallel generated codes from PP's. In spite of the performance of the most known CG tools (CGT) such as Pluto, these latter don't take into account all the TA intrinsic characteristics which may be a rich source of performance enhancement. Thus, we take a special interest in the generated code optimization based on some hardware characteristics of the TA in order to improve the completion time. In particular, we aim at optimizing parallel generated communication-avoiding codes. The remainder of the paper involves four sections organized as follows. In section 2, we present a brief state-of-the art on the most known optimization tools for parallel codes incorporated in Pluto. Section 3 is devoted to the description of our general two-phase approach, where we detail an experimental study on a multicore parallel platform that allows our theoretical contribution validation and evaluation. We finally conclude and present some perspectives in section 4.

2 Brief state-of-the art

In this section, we present a summary on existing code optimization tools related to the Pluto compiler [1].

In fact, we find in the literature many code optimization methods, particularly for parallel processing. This optimization may be seen as aiming at reducing either the code size or the run time in order to increase the overall performance. In our work, we mainly address the second point i.e. improving run time. Clearly, such optimization is quite important for codes automatically generated by CGT's. Indeed, these latter, though powerful in parallel code generation, are sometimes inefficient from an optimization point of view because of their generality and the overheads they induce.

In this paper, a special interest is attached to the Pluto compiler [1] because of the high-level optimizations it achieves by means of the code optimization tool Cloog [2] and the tiling technique. More precisely, Pluto transforms C programs from source to source for coarse-grained parallelism and data locality simultaneously for generating parallel C OpenMP codes [1]. It uses a scheduling algorithm which tries to find affine transformations allowing the most efficient tiling.

Cloog [2] is a free software which generates code for scanning Z-polyhedra. It is designed to build control automata for high-level synthesis and find the best polynomial approximation of a given function. Based on Quilleré's method [3], Cloog helps to solve scanning Z-polyhedra matters [2]. It was originally written to solve code generation

problems for optimizing compilers based on the polytope model. It allows avoiding control overhead and producing effective codes in a reasonable amount of time when there is full control on generated code quality. Besides, it performs dependency analysis and provides scattering functions which remove redundant constraints and specify better transformations reordering. These latter transforms the original polyhedron, by applying a new lexicographic order, into an equivalent target polyhedron [4].

Tiling which is a loop transformation that decomposes the whole computation set into subsets of smaller computation blocks, is especially used in vectorization, coarse-grained parallelism, and also in many high-level program optimizations such as data locality. Following old tiled loop generation methods with fixed tile sizes e.g. the approaches of Goumas and al [5], Kelly and al. [6], the SUIF [7] tool and the method of Ahmed and al. [8], new tiling techniques have been developed. These latter use parameterized and multi-level tiled loop generation methods where the block size is not fixed at compile time, but remains a symbolic constant. Hence, it may be changed even at runtime. Among such methods, we can mention the PrimeTile tool of Hartono and al. [9], the techniques of Baskaran and al. [10] and those of Kim [11]. Let us notice that the tiled code generation scheme of Pluto, which initially used fixed tile sizes, was extended to use parametric tiling by incorporating the tiled code generation with parametric tile sizes within the polyhedral model addressed by Renganarayanan and al. [12] in order to make it convenient for iterative and empirical optimization.

To conclude, we notice that most of these optimization techniques, though effective in parallel code generation, don't care of the most intrinsic characteristics of the TA. Subsequently, through this paper, we aim to present our proposal for optimizing the quality of the code generated from Pluto by taking into account some physical characteristics of the TA. For this purpose, based on a selective study of hardware architecture detection tools, we have chosen the Hwloc package [13]. Indeed, it is able to supply generic and complete vision of hardware architecture. The provided data are more detailed relatively to the other methods. It generates a portable abstraction of the hierarchical topology of modern architectures with gathering information about NUMA memory nodes, sockets, shared caches, cores and simultaneous multi-threading.

3 Proposed approach

In this section, we're going to present the general principle of our contribution as well as the following two steps: (i) distribution of the parallel tasks onto the cores of a target architecture (TA) and (ii) cache use optimization. A series of experiments were achieved on the chosen TA in order to evaluate the interest of our contribution.

3.1 General principle

Our aim is to link the software topology of the parallel code generated by Pluto [1] to the target hardware topology in order to optimize completion time. Thus, we have taken into consideration some hardware features e.g. cache memory size, cache sharing, cores organization on nodes, etc. In this work, as it is shown in Fig. 1, given a parallel program (not adapted to any architecture and generated by Pluto) and a target multicore architecture, we propose the following approach: After detecting the architecture hardware characteristics using the Hwloc framework [13] and the Cpuinfo library, we had developed a software component allowing code adaptation to the TA. So, we propose two steps towards this adaptation consisting, first, in distributing the different parallel tasks on the various available cores, and second in optimizing the cache memory use.

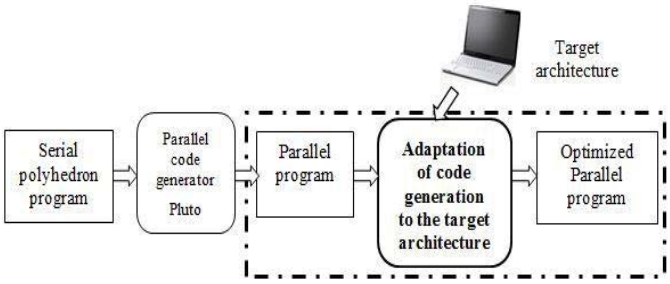


Fig. 1. *Schematic description of the proposal*

In the following subsections, we'll first study various ways for distributing parallel tasks on the cores and we'll choose the best allocation through an experimental study. Secondly, we'll propose a method to improve the quality of a parallel generated code by taking into account some cache memory characteristics of the TA. This improvement will be evaluated through a series of experiments.

Let us mention that our multicore TA is an Intel ® Xeon dual quadcore processor CPU E5420 @ 2.50GHz. Its eight cores have dedicated L1 cache with 32 KB size, and share in pairs the L2 cache level whose size is 6MB. The RAM size is 4 GB.

Our target parallel programs don't require any core communication since the iterations (tasks) are independent. This feature may be called communication-free (or more precisely communication-avoiding). We chose three benchmarks codes: (i) matrix addition (MA), (ii) matrix-vector product (MVP) and (iii) matrix-matrix product (MMP) where N denotes the size of the processed matrices.

The two proposed improvement phases were implemented in an automatic tool which takes as input a parallel communication-avoiding code generated by Pluto (CGP) and then generates an equivalent code undergoing the optimization that will be more detailed in the remainder.

3.2 Distributing the parallel tasks on cores

Taking into account the target architecture (TA), there are several factors which have a direct impact on run time. Particularly, changing the way to distribute threads on the different cores may reduce data exchanges and thus improve data locality. For this reason, it is important to appropriately choose the allocation of parallel tasks on the available cores. We therefore examined some parallel code allocation methods expressed by the OpenMP library. In fact, the distribution of parallel iterations (tasks) on physical cores may be automatized by specifying the chunk size (CHS) i.e. the number of iterations constituting a thread. The chunks may be computed then allocated to threads in compile time. In this case, the OpenMP task allocation may be performed either statically or dynamically. A static allocation means that iteration blocks divided into chunks are statically mapped to the execution threads in a round-robin manner. However, for the dynamic allocation, once a thread is available, it requests a chunk of iterations to execute.

Another manner in task distributing on cores is the OpenMP sections directive. Each section is assigned to one thread and each thread (TH) may be explicitly executed by one core. To illustrate this distribution manner, let us consider the following example consisting of a single parallel loop (the loop body may be a perfect nest not only a single iteration):

```
FORALL (i = 1, N)
{ S(i)}   /*loop body*/
ENDFORALL
```

In our case, we run as many sections as the cores number (i.e. 8) as follows:

```
Section 1: { TH 0: (i= 1, N/8 )          Allocate (TH 0; Core0)}
Section 2: { TH 1: (i= (N/8)+1, N/4)     Allocate (TH 1; Core1)}
Section 3: { TH 2: (i= (N/4)+1, 3N/8)    Allocate (TH 2; Core2)}
Section 4: { TH 3: (i= (3N/8)+1, N/2)    Allocate (TH 3; Core3)}
Section 5: { TH 4: (i= (N/2)+1, 5N/8)    Allocate (TH 4; Core4)}
Section 6: { TH 5: (i= (5N/8)+1, 6N/8)   Allocate (TH 5; Core5)}
Section 7: { TH 6: (i= (6N/8)+1, 7N/8)   Allocate (TH 6; Core6)}
Section 8: { TH 7: (i= (7N/8)+1, N)      Allocate (TH 7; Core7)}
```

An experimental study covering on the one hand static and dynamic automatic allocation, and on the other hand manual allocation (sections) was carried out in order to compare them and choose the most efficient.

Three chunk types in the automatic case were studied by varying the sizes i.e. :

- X1 = cache line size
- X2 = cache size
- X3 = cache size / 2
- X4 = problem size (iterations number) / the cores number

X is considered ST (resp. DY) if the automatic allocation is static (resp. dynamic).

We detail experimental comparisons carried out on our TA and illustrated through some excerpts of our performed tests in Tables I, II and III, and Fig. 2 for the three benchmarks MA, MPV and MMP. We mention that the following notations are adopted in these tables in order to indicate the execution time of the studied codes:

- PLUTO : code generated by Pluto (CGP)
- STi : CGP with a CHS equal to Xi (i=1…4, see above) when the allocation is static
- DYi : CGP with a CHS equal to Xi (i=1…4) when the allocation is dynamic
- TILE : CGP undergoing the tiling technique adopted by Pluto
- Section : CGP with manual allocation (fair sections)
- Execution times : T1 for MA (ms), T2 for MVP (ms), T3 for MMP (s)
- Reducing time ratio r (%):
  rj(Version)=(Tj(PLUTO)-Tj(Version)) / Tj(PLUTO)
  where j=1 for benchmark MA, 2 for benchmark MVP, 3 for benchmark MMP and
  Version ϵ {ST1,.., ST4, DY1,.., DY4, TILE, Section}

The various algorithm versions were coded in C under Linux. For the parallel experiments, we used the shared memory OpenMP environment. We point out that for each benchmark; we chose 10 values of N in the range [500, 5000] with a step equal to 500. For each N, the execution time is the mean of five runs. We therefore achieved 330 tests in total (110 for each benchmark). Excerpts of the results we obtained are depicted below.

TABLE I. EXECUTION TIME (MS) OF MA FOR DIFFERENT ALLOCATIONS

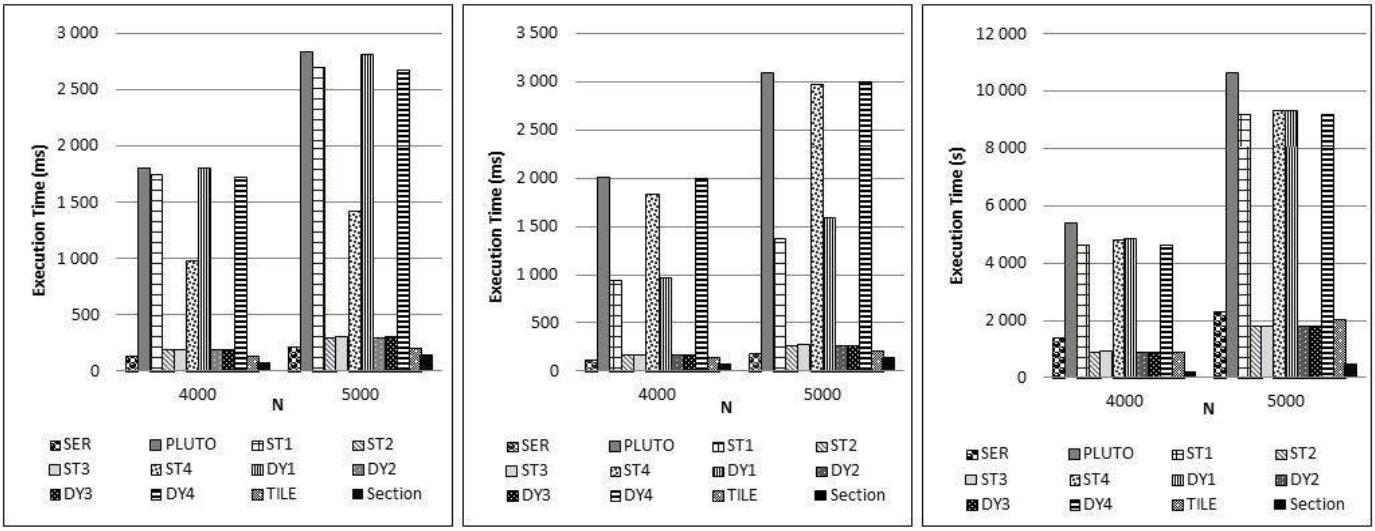
N	PLUTO	ST1	ST2	ST3	ST4	DY1	DY2	DY3	DY4	TILE	Section
1000	116.783	118.135	14.743	15.660	48.610	113.249	15.325	15.783	111.800	8.349	**7.779**
2000	457.104	444.688	48.108	48.917	212.243	461.003	49.125	48.897	432.978	33.110	**17.815**
3000	983.732	954.530	108.772	108.735	480.052	1049.271	108.273	109.783	973.274	75.970	**45.061**
4000	1799.962	1750.715	194.027	192.424	982.619	1809.196	192.318	193.677	1730.625	142.906	**74.757**
5000	2832.404	2704.928	304.036	302.762	1431.34	2825.001	303.613	305.762	2682.211	209.901	**138.349**

TABLE II. EXECUTION TIME (MS) OF MVP FOR DIFFERENT ALLOCATIONS

N	PLUTO	ST1	ST2	ST3	ST4	DY1	DY2	DY3	DY4	TILE	Section
1000	121.466	66.508	16.209	15.900	123.169	63.398	17.988	14.497	128.805	7.2170	**3.838**
2000	501.556	256.033	42.869	42.876	484.484	247.971	43.953	42.898	498.401	29.646	**6.571**
3000	1159.214	565.273	100.598	96.345	1058.303	540.607	96.287	96.318	1097.031	66.422	**22.584**
4000	2005.986	939.893	175.306	171.199	1842.777	975.638	171.115	171.033	2008.391	119.317	**34.703**
5000	3097.488	1375.272	269.459	272.383	2977.059	1592.616	268.522	268.374	3009.494	183.488	**50.349**

TABLE III. EXECUTION TIME (S) OF MMP FOR THE DIFFERENT ALLOCATIONS

N	PLUTO	ST1	ST2	ST3	ST4	DY1	DY2	DY3	DY4	TILE	Section
1000	89.934	76.675	14.166	14.173	80.813	79.644	14.174	14.178	76.412	8.941	**1.892**
2000	669.358	599.686	114.846	114.851	616.960	611.035	114.940	114.955	579.875	80.489	**19.097**
3000	2262.674	1962.621	386.629	386.618	2054.293	2035.842	386.889	386.897	1930.912	322.032	**102.357**
4000	5421.078	4636.604	915.902	916.462	4837.712	4853.935	916.4089	916.567	4614.829	895.139	**212.622**
5000	10629.747	9198.934	1798.107	1800.011	9320.983	9353.020	1798.316	1798.423	9190.790	2015.090	**475.367**



(a). MA (b). MVP (c). MMP

Fig. 2. *Experimental comparison of the different allocations of : (a) MA , (b) MVP, (c) MMP*

From Tables I, II and III, we noticed that automatic allocation of OpenMP tasks performed either statically or dynamically is less efficient than the manual one. Indeed, it's due to the generation of some overheads. Fig. 2 shows that the best inherent allocation is expressed in "Section" version for all the studied benchmarks. This improvement is more clarified in Table IV where we notice that the reducing time ratios (r) are high and vary in the range 93.33 (N=1000) - 95.84 % (N=4000) for MA, 96.84 (N=1000) - 98.68 % (N=2000) for MVP, and 95.48 (N=3000) - 97.90 % (N=1000) for MMP. Remark that the variations of r are not regular in terms of N.

We expect that the improvement would continue for larger N. Let us add that with ST2, ST3, DY2, DY3 and TILE, we also obtain an improvement but less than with Section. Notice finally that a negative r corresponds to an increase in execution time. According to the comparative study below (see Fig. 2 and Table IV), we hence choose the manual allocation with fair sections. Subsequently, we decided in the tool that we have implemented to produce in the generated code as many sections as available cores (manual allocation). We present in the following section the major optimization phase which improves data locality through better cache memory use.

TABLE IV. REDUCING TIME RATIOS R(%)

Benchmark	N	r(ST1)	r(ST2)	r(ST3)	r(ST4)	r(DY1)	r(DY2)	r(DY3)	r(DY4)	r(TILE)	r(Section)
MA	1000	-1.15	87.37	86.59	58.37	3.02	86.87	86.48	4.26	92.85	**93.33**
	2000	2.71	89.47	89.29	53.56	-0.85	89.25	89.30	5.27	92.75	**96.10**
	3000	2.96	88.94	88.94	51.20	-6.66	88.99	88.84	1.06	92.27	**95.41**
	4000	2.73	89.22	89.30	45.40	-0.51	89.31	89.23	3.85	92.06	**95.84**
	5000	4.50	89.26	89.31	49.46	0.26	89.28	89.20	5.30	92.58	**95.11**
MVP	1000	45.24	86.65	86.90	-1.40	47.80	85.19	88.06	-6.04	94.05	**96.84**
	2000	48.95	91.45	91.45	3.40	50.55	91.23	91.44	0.62	94.08	**98.68**
	3000	51.23	91.32	91.68	8.70	53.36	91.69	91.69	5.36	94.27	**98.05**
	4000	53.14	91.26	91.46	8.13	51.36	91.46	91.47	-0.11	94.05	**98.27**
	5000	55.60	91.30	91.20	3.88	48.58	91.33	91.33	2.84	94.07	**98.37**
MMP	1000	1.30	7.06	84.24	10.14	11.44	84.24	84.24	15.04	90.06	**97.90**
	2000	0.17	0.87	82.84	7.83	8.71	82.83	82.83	13.37	87.98	**97.15**
	3000	0.05	0.26	82.91	9.21	10.02	82.90	82.90	14.66	85.77	**95.48**
	4000	0.02	0.11	83.09	10.76	10.46	83.10	83.09	14.87	83.49	**96.08**
	5000	0.01	0.06	83.07	12.31	12.01	83.08	83.08	13.54	81.04	**95.53**

Remark: A negative ratio corresponds to an increasing time (e.g. r(ST1)= -1.15 for MA when N=1000).

3.3 Optimizing the cache memory use

Typically, when the core-processor needs to read or to write a data, it fetches it first in the cache memory which avoids access to the main memory. If the cache memory has a copy of the data required by the processor, this is called a cache hit, otherwise this is called a cache miss. Cache memory is in fact categorized in levels that describe its closeness and accessibility to the processor. Level 1 (L1), which is extremely fast but of relatively small size, is located close to the processor and used for temporary instructions and data storage. Level 2 (L2) cache, located half-way between the processor and the system bus, is fairly fast and medium-sized. Level 3 (L3) cache is relatively large and close to the RAM.

Some cache memory levels can be shared between cores. This sharing allows a better cache coherence as well as a considerable decreasing of cache misses. Cores communication becomes faster since it depends on the shared cache memory speed which is faster than the main memory.

There are two basic types of reference locality. Temporal locality refers to the re-use of specific data, and/or resources, within relatively small time duration. Spatial locality refers to the use of data elements within relatively close storage locations.

To take advantage of spatial locality, we suggest loading in advance data that we shall probably need in the near future in order to enable the processor to find the requested data it usually seeks in the cache memory and to avoid the overhead due to main memory access. This phenomenon is called the Prefetching, and may be performed either by the programmer or directly by the processor which is called the hardware prefetcher. This latter can operate transparently, without any programmer intervention, to fetch data and instructions from memory into the unified second-level cache. The hardware prefetcher can be trusted to prefetch highly regular accesses, while software Prefetching can be used for irregular accesses that the hardware one can't handle.

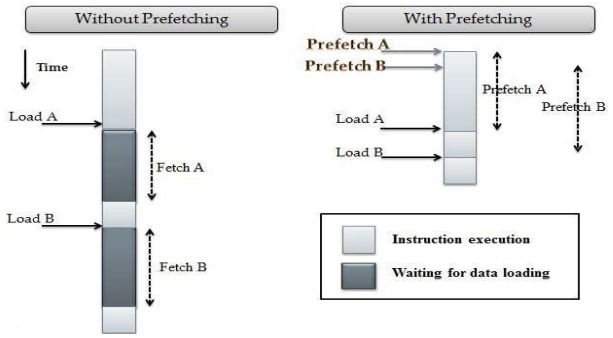


Fig. 3. *Prefetching principle*

In this work, we supposed that our TA doesn't rely on a hardware prefetcher. It's in this context that the second phase focuses on the software Prefetching technique (PT) to take advantage of data locality in order to optimize cache memory use.

Let us adopt the following notations:

- CL : Size of cache memory line
- DS : Data size
- CQ = CL/DS : Cache Line-data ratio

The Prefetching technique can be explained through the following MMP code generated by Pluto (see Nest 1.).

Nest1. Matrix- Matrix Product MMP (A,B,C, N)

```
lbp=0;
ubp=N-1;
#pragma omp parallel for
for (t2=lbp;t2<=ubp;t2++) {
    for (t3=0;t3<=N-1;t3++) {
        for (t4=0;t4<=N-1;t4++) {
            C[t2][t4]=C[t2][t4]+A[t2][t3]*B[t3][t4];
        }
} }
```

- Starting from the innermost for loop (t4), we detect the cache memory size line (assuming that CL= 64B) by calling a function from the Cpuinfo library.
- Since we know the data size to prefetch (e.g. DS(C[i,j]) = sizeof(int) = 4 B), we compute the cache line-data ratio corresponding to the data to prefetch (in this case CQ = CL/DS = 64/4 = 16). Since all transfers between the main memory and the cache memory are done cache line-wise, when loading C[t2, t4], we'll load CQ data in the same line starting by C[t2, t4] until C[t2, t4+15].
- We detect the access type for each data structure, then we call the necessary Prefetching instructions. In our case, C[t2, t4] would be prefetched when writing and B[t3,t4] when reading.
- We apply the Prefetching mechanism on the innermost loop by duplicating the instruction CQ times and by Prefetching at each iteration, the requested data.

This finally leads to the illustrated code in Nest 2. An experimental study targeting the same TA was achieved on the three benchmarks MA, MVP and MMP with the different allocations previously tested after undergoing the Prefetching technique. This could lead to interesting improvements. Tables V, VI and VII show a comparison emphasizing that the best result is provided by the program working with the manual allocation using equal sized sections and undergoing the Prefetching technique (PT).

We detail in Tables V, VI and VII respectively for MA, MPV and MMP, excerpts of experimental comparisons carried out on our TA.

Nest2. Matrix- Matrix product MMP(A,B,C, N)

```
for (t2 = lbp ; t2 < ubp; t2++) {
    for (t3 = 0; t3 <= N - 1; t3++) {
        __builtin_prefetch(&C[t2][0], 1, 1);
        __builtin_prefetch(&B[t3][0], 0, 1);
        for (t4 = 0; t4 <= (N – 1) - 16; t4 += 16) {
    __builtin_prefetch(&C[t2][t4 + 16], 1, 1);
    __builtin_prefetch(&B[t3][t4 + 16], 0, 1);
C[t2][t4 + 0] = C[t2][t4 + 0] + A[t2][t3] * B[t3][t4 + 0];
C[t2][t4 + 1] = C[t2][t4 + 1] + A[t2][t3] * B[t3][t4 + 1];
C[t2][t4 + 2] = C[t2][t4 + 2] + A[t2][t3] * B[t3][t4 + 2];
C[t2][t4 + 3] = C[t2][t4 + 3] + A[t2][t3] * B[t3][t4 + 3];
C[t2][t4 + 4] = C[t2][t4 + 4] + A[t2][t3] * B[t3][t4 + 4];
C[t2][t4 + 5] = C[t2][t4 + 5] + A[t2][t3] * B[t3][t4 + 5];
C[t2][t4 + 6] = C[t2][t4 + 6] + A[t2][t3] * B[t3][t4 + 6];
C[t2][t4 + 7] = C[t2][t4 + 7] + A[t2][t3] * B[t3][t4 + 7];
C[t2][t4 + 8] = C[t2][t4 + 8] + A[t2][t3] * B[t3][t4 + 8];
C[t2][t4 + 9] = C[t2][t4 + 9] + A[t2][t3] * B[t3][t4 + 9];
C[t2][t4 + 10] = C[t2][t4 + 10] + A[t2][t3] * B[t3][t4 + 10];
C[t2][t4 + 11] = C[t2][t4 + 11] + A[t2][t3] * B[t3][t4 + 11];
C[t2][t4 + 12] = C[t2][t4 + 12] + A[t2][t3] * B[t3][t4 + 12];
C[t2][t4 + 13] = C[t2][t4 + 13] + A[t2][t3] * B[t3][t4 + 13];
C[t2][t4 + 14] = C[t2][t4 + 14] + A[t2][t3] * B[t3][t4 + 14];
C[t2][t4 + 15] = C[t2][t4 + 15] + A[t2][t3] * B[t3][t4 + 15];}
        for (t4 = (N – 1) - 16 + 1; t4 <= N - 1; t4++) {
            C[t2][t4] = C[t2][t4] + A[t2][t3] * B[t3][t4]; }}}
```

We mention that the following notations are adopted in order to indicate the execution time of the studied codes:

- STi_Pre: code generated by Pluto (CGP) with a chunk size (CHS) equal to Xi (i=1..4, see section 3.2) when the allocation is static, and undergoing the PT.
- DYi_Pre: CGP with a CHS equal to Xi (i=1..4) when the allocation is dynamic, and undergoing the PT.
- Section_Pre: CGP with manual allocation (fair sections) after undergoing the PT.
- Execution times: T'1 for MA (ms), T'2 for MVP (ms), T'3 for MMP (s).
- Reducing time ratio r' (%):
  r'j(Version)=(T'j(PLUTO)-T'j(Version))/T'j(PLUTO)
  where j=1 for MA, 2 for MVP and 3 for MMP
  and Version ϵ { STi_Pre, DYi_Pre, Section_Pre }

For these parallel experiments, we used the "GNU_SOURCE" package. We remind that, for each benchmark algorithm, we chose 10 values of N in the range [500, 5000]. For each N, the execution time is the mean of five runs. So, we achieved 90 tests for each benchmark. Excerpts of the results we obtained are depicted below.

The experimental study of the MA benchmark (respectively MVP and MMP) was carried out with both static and dynamic automatic allocation by varying the CHS as indicated in section 3.2. It was then processed with manual allocation. All these experimental versions have undergone the PT as explained in section 3.3. Table VIII shows our excerpts of experimental comparisons of ST1, ST4, DY1 and Section versions performed on our TA. Several improvements are obtained through the reduction of the execution time.

Similar remarks as done in section 3.2 may be detailed here. The best version is still the manual allocation expressed in "Section_Pre" version for the three studied benchmarks. The reducing time ratios (r') are also high and vary in the range 93.38 (N=1000) - 96.13 % (N=2000) for MA, 92.53 (N=1000) - 98.93 % (N=5000) for MVP, and 96.58 (N=5000) - 98.16 % (N=1000) for MMP. Furthermore, the variations of r' are irregular in terms of N. We have to note that with the others versions, we also obtain an improvement relative to the code generated by Pluto (CGP) but less than with Section undergoing the Prefetching technique (PT).

TABLE V. EXECUTION TIME (MS) OF MA FOR DIFFERENT ALLOCATIONS WITH PREFETCHING

N	ST1_Pre	ST2_Pre	ST3_Pre	ST4_Pre	DY1_Pre	DY2_Pre	DY3_Pre	DY4_Pre	Section_Pre
1000	27.953	13.206	12.527	25.352	25.612	11.023	12.553	23.741	**7.734**
2000	113.844	40.82	40.923	94.616	104.572	40.116	40.483	99.527	**17.697**
3000	248.827	91.688	92.648	224.009	240.715	90.356	89.922	223.656	**42.593**
4000	411.637	161.35	161.921	399.57	410.759	160.156	160.054	379.512	**76.386**
5000	643.714	257.076	255.802	594.37	634.819	251.689	253.035	592.206	**112.569**

TABLE VI. EXECUTION TIME (MS) OF MVP FOR DIFFERENT ALLOCATIONS WITH PREFETCHING

N	ST1_Pre	ST2_Pre	ST3_Pre	ST4_Pre	DY1_Pre	DY2_Pre	DY3_Pre	DY4_Pre	Section_Pre
1000	30.424	15.067	11.1	27.859	33.475	15.188	9.536	29.702	**9.077**
2000	127.251	37.267	33.586	109.82	130.924	34.127	34.106	106.012	**14.698**
3000	272.951	77.719	76.621	245.621	293.985	78.194	78.116	244.15	**24.048**
4000	486.27	133.826	133.928	412.417	527.677	135.708	135.798	439.297	**32.441**
5000	743.987	215.561	213.861	613.375	808.491	218.966	218.684	667.755	**33.017**

TABLE VII. EXECUTION TIME (S) OF MMP FOR DIFFERENT ALLOCATIONS WITH PREFETCHING

N	ST1_Pre	ST2_Pre	ST3_Pre	ST4_Pre	DY1_Pre	DY2_Pre	DY3_Pre	DY4_Pre	Section_Pre
1000	26.545	12.198	12.194	23.069	21.905	12.220	12.220	21.499	**1.658**
2000	220.829	97.846	97.845	201.861	170.788	97.792	97.869	169.298	**19.742**
3000	762.466	327.575	327.865	727.719	588.535	328.164	328.430	569.553	**66.423**
4000	1816.399	775.579	775.503	1688.525	1407.161	776.375	776.273	1348.133	**184.934**
5000	3541.314	1543.684	1543.122	3326.861	2667.724	1545.763	1545.772	2619.664	**363.515**

TABLE VIII. REDUCING TIME RATIOS R' (%)

Benchmark	N	r'(ST1_Pre)	r'(ST2_Pre)	r'(ST3_Pre)	r'(ST4_Pre)	r'(DY1_Pre)	r'(DY2_Pre)	r'(DY3_Pre)	r'(DY4_Pre)	r'(Section_Pre)
MA	1000	76.06	88.69	89.27	78.29	78.07	90.56	89.25	79.67	**93.38**
	2000	75.09	91.07	91.05	79.30	77.12	91.22	91.14	78.23	**96.13**
	3000	74.71	90.68	90.58	77.23	75.53	90.81	90.86	77.26	**95.67**
	4000	77.13	91.04	91.00	77.80	77.18	91.10	91.11	78.92	**95.76**
	5000	77.27	90.92	90.97	79.02	77.59	91.11	91.07	79.09	**96.03**
MVP	1000	74.95	87.60	90.86	77.06	72.44	87.50	92.15	75.55	**92.53**
	2000	74.63	92.57	93.30	78.10	73.90	93.20	93.20	78.86	**97.07**
	3000	76.45	93.30	93.39	78.81	74.64	93.25	93.26	78,94	**97.93**
	4000	75.76	93.33	93.32	79.44	73.69	93.23	93.23	78.10	**98.38**
	5000	75.98	93.04	93.10	80.20	73.90	92.93	92.94	78.44	**98.93**
MMP	1000	70.48	86.44	86.44	74.35	75.64	86.41	86.41	76.10	**98.16**
	2000	67.01	85.38	85.38	69.84	74.48	85.39	85.38	74.71	**97.05**
	3000	66.30	85.52	85.51	67.84	73.99	85.50	85.48	74.83	**97.06**
	4000	66.49	85.69	85.69	68.85	74.04	85.68	85.68	75.13	**96.59**
	5000	66.68	85.48	85.48	68.70	74.90	85.46	85.46	75.36	**96.58**

4 Conclusions

Addressing polyhedron program automatic parallelization, we first studied the most known code optimization tools related to Pluto. Then, we proposed a two-phase approach leading to parallel code optimization. Targeting a multicore machine, our proposal takes into consideration the number of cores and the cache line size, and aims to especially optimize the cache memory use. In fact, starting with a program generated by Pluto, we proposed to automatically generate a parallel avoiding-communication code with explicit task allocation on the available cores. This code then automatically underwent the Prefetching technique. It therefore took advantage of the spatial locality which minimized the main memory access. Ultimately, our proposal was translated by a software component that was integrated with Pluto. A series of experimentations could validate our contribution and specify its practical interest.

However, several interesting points remain to be seen, particularly: (i) extension of the experimental study to other benchmark matrix algorithms, (ii) optimization of parallel programs with core communications by taking into consideration the cache memory sharing, (iii) adaptation of the code generation to other parallel architectures e.g. GPUs.

Acknowledgment

We'd like to thank Pr. Zaher Mahjoub for his valuable help.

5 References

[1] U. Bondhugula, A. Hartono, and J. Ramanujan, "A Practical Automatic Polyhedral Parallelizer and Locality Optimizer", ACM SIGPLAN Conference on Programming Languages Design and Implementation (PLDI), Tucson, Arizona, USA, vol. 43(6), pp. 101-113, June 2008.

[2] C. Bastoul, and P. Feautrier, "Improving data locality by chunking", CC'12 International Conference on Compiler Construction, Poland, Vol. 2622, pp. 320-335, April 2003.

[3] F. Quilleré, S. Rajopadhy, and D. Wilde, "Generation of Efficient Nested Loops from Polyhedra", International Journal of Parallel Programming, vol. 28(5), pp. 469-498, October 2000.

[4] C. Bastoul, "Improving Data Locality in Static Control Programs", PhD thesis, University Paris 6, Pierre et Marie Curie, France, December 2004.

[5] G. Goumas, M. Athanasaki, and N. Koziris, "An efficient code generation technique for tiled iteration spaces", IEEE Transactions on Parallel and Distributed Systems, vol. 14(10), pp. 1021-1034, October 2003.

[6] W. Kelly, W. Pugh, and E. Rosser, "Code generation for multiple mappings", Frontiers'95: The 5th Symposium on the Frontiers of Massively Parallel Computation, McLean, VA, pp. 332, February 1995.

[7] R. P. Wilson, Robert S. French, C. S. Wilson, S. P. Amarasinghe, J. M. Anderson, S. W. K. Tjiang, S-W. Liao, C-W. Tseng, M. W. Hall, M. S. Lam, and J. L. Hennessy, "SUIF: An infrastructure for research on parallelizing and optimizing compilers", J. ACM SIGPLAN Notices, vol. 29(12), pp. 31-37, December 1994.

[8] N. Ahmed, N. Mateev, and K. Pingali, "Synthesizing transformations for locality enhancement of imperfectly-nested loop nests", Int. J. of Parallel Programming, vol. 29(5), pp. 493–544, October 2001.

[9] A. Hartono, M. M. Baskaran, J. Ramanujam, and P. Sadayappan, "Dyntile: Parametric tiled loop generation for effective parallel execution on multicore processors", IEEE International Parallel and Distributed Processing Symposium (IPDPS), Atlanta, USA, pp.1-12, April 2010.

[10] M. M. Baskaran, A. Hartono, S. Tavarageri, T. Henretty, J. Ramanujam, and P. Sadayappan, "Parameterized tiling revisited", IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Toronto, Ontario, Canada, pp. 200-209, April 2010.

[11] D. Kim, "Parameterized and Multi-level Tiled Loop Generation", PhD thesis, Colorado State University, Fort Collins, CO, USA, April 2010.

[12] L. Renganarayanan, D. Kim, S. Rajopadhye, and M. M. Strout, "Parameterized tiled loops for free", The 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '07), San Diego, California, USA, vol. 42(6), pp. 405-414, June 2007.

[13] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and N. Namyst, "hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications", Proceedings of the 18th Euromicro International Conference PDP 2010: Parallel, Distributed and Network-Based Processing, IEEE Computer Society Press, Pisa, Italia, pp 180-186, February 2010.

Parallel Edge Detection using Sobel Algorithm with Contract-time Anytime Algorithm in CUDA

Md Kamal Hossain, Md Assaduzzaman Ashique, Md Asif Ibtehaz, and Jia Uddin
Computer Science and Engineering Department, BRAC University
Dhaka 1212, Bangladesh.
E-mail: shajal16@gmail.com, ashique12301017@gmail.com, ibtehaz.shawon@gmail.com, jia.uddin@bracu.ac.bd

Abstract - *Edge detection is a considerably important factor in image or video processing. Detection of edges plays a significant role in image segmentation, data compression, well matching, and image reconstruction. Among several edge detection approaches we focus on Sobel edge detection using contract-time anytime algorithm in CUDA. To reduce the computational complexity we implemented our proposed edge detection method using CUDA. In the experimental setup we have used NVIDIA GTX 550Ti GPU along with AMD FX8150 Processor and 8 GB RAM. Finally, we measure speedup using 3 steps of contract-time anytime of our proposed parallel implementation model. Comparing with conventional serial CPU based edge detection we have experienced maximum 4X speedup of proposed implementation for 16 block dimension.*

Keywords: Edge detection; CUDA; Anytime algorithm; Parallel Computing; Sobel; NVIDIA

1 Introduction

Edge detection from a color image is a very important and basically critical area in low level image processing. For performing high speed industrialized application based on image processing, edge detection is a mandatory thing to enhance work rate as well as accuracy. A number of researchers works on several edge detection algorithms and they give different responses and details to the different input images [1-7]. Edge detection quality has a great impact on realization of complex automated computer/machine vision systems [1]. Among them, the Sobel edge detection algorithm is much more popular than simple gradient operators due to its property to counteract the noise sensitivity and easier implementation process [2]. While using Sobel operator for GPU takes much less time than CPU. Again, the use of Interruption-Algorithm for image processing much less time efficient [3]. Moreover, in case of canny edge detection in GPU time process seems efficient but not enough for real time [4]. The use of anytime algorithm for GPU architecture makes it run faster in association with Dijkstra's algorithm [5, 8]. In addition, anytime algorithm seems much efficient when it is used for observing different tasks [6]. Interruptible Anytime Algorithm for image processing is much faster than normal image processing algorithms and also gives the privilege of getting output in different stage of time [7]. That is why, we are choosing contract-time anytime algorithm in co-ordination with Sobel operator for proposed parallel implementation.

The rest of this paper is organized as follows: Section II describes the detailed of Anytime Algorithm. Section III presents details about the proposed Sobel edge detection using Contract-time anytime algorithm in an NVIDIA GPU in CUDA, and Section IV contains experimental results of analysis and comparison. Finally, Section V concludes this paper.

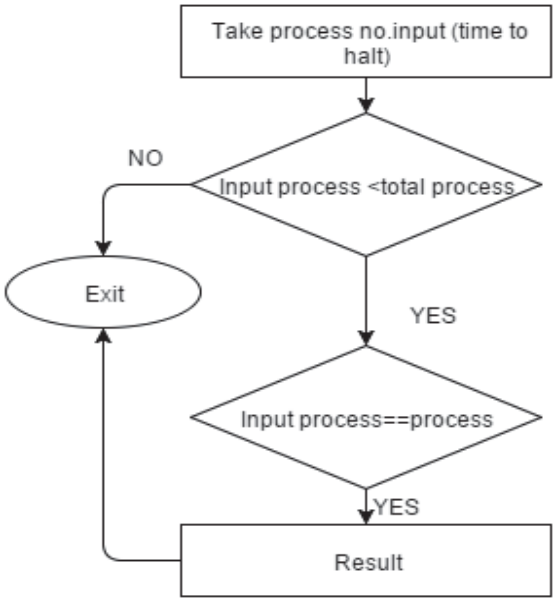


Figure 1. Model of contract-time Anytime Algorithm.

2 Contract-time anytime algorithm

Anytime algorithm is a kind of algorithm that searches for better result instead of calculating the final result [3, 5, and 7]. There are mainly two types of anytime algorithm: Interruption and Contract-time. Interruption anytime is the algorithm that continues running and can be stopped anytime to get the final result. For contract-time anytime it is a little different, as final result is generated based on user input of

processes or time. Figure 1 is the presentation of contract-time anytime algorithm, which we used in our proposed edge detection algorithm.

3 Proposed model

Figure 2 shows the block diagram of proposed parallel implementation of CPU-GPU based edge detection method. To evaluate our proposed model have utilized different test images. First of all, we have taken the images as input. As the images are color images, we converted it into gray scale images. The process ran in GPU and we used interruptive anytime algorithm to make the conversion process faster, as depicted Figure 2.

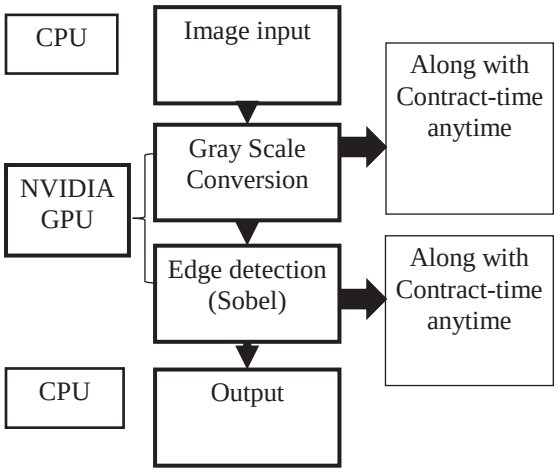


Figure 2. Proposed model for Sobel and contract-time anytime based edge detection using NVIDIA GPU.

After that, the edge detection process runs in GPU. Again, used interruptive anytime algorithm to detect the edges. We have calculated the time for Sobel operator in CUDA environment and took the time for processing and compared results. Our all the outputs shows in CPU that were calculated in GPU that is the primary aspiration of parallel implementation of Sobel operator along with any time algorithm.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Figure 3. Sobel operator Convolution Kernel/Mask.

Figure 3 is a Sobel operator matrix that we used to calculate value for detecting edges in our CUDA environment. Here we used the general Sobel operator gradient matrix with CUDA. That detects edges and the time taken here is less than normal Sobel operator in conventional CPU programming.

$$S = \sqrt{S_x^{\,2} + S_y^{\,2}} \quad (1)$$

$$|S| = |S_x| + |S_y| \quad (2)$$

Where S_x represents horizontal convolution mask and followed by S_y represents vertical convolution mask. These convolution mask is being used for calculating the gradient.

Sobel operator 2D gradient based measurement is performed on an image. High spatial frequency which correspond to edges is mainly used to perform the measurement. For measurement we use Equation 1 which is the equation for gradient magnitude. In addition, Equation 2 can give approximate magnitude for the computation, which is much faster to compute the gradient.

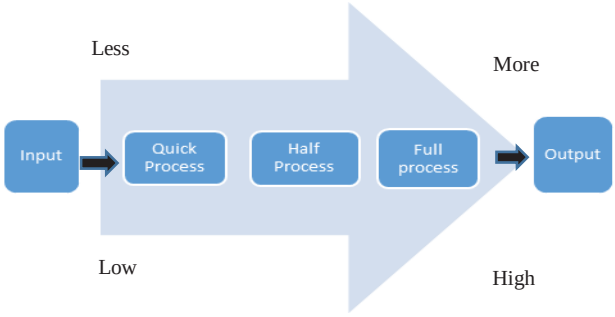


Figure 4. Contract-time Anytime Algorithm task processing.

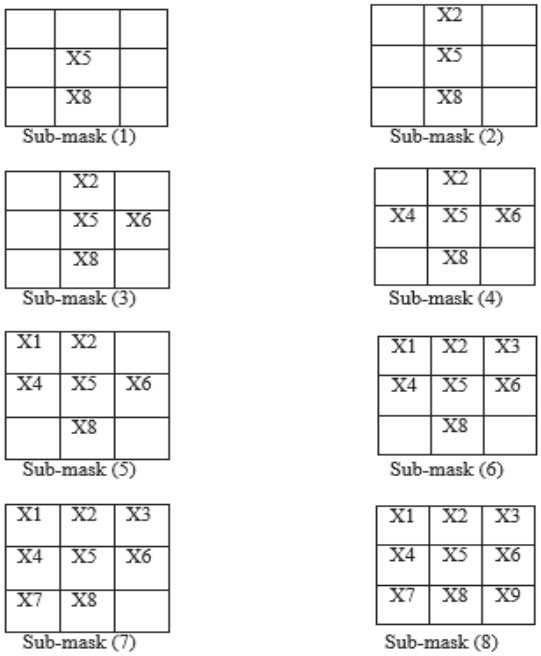


Figure 5. 3x3 sub-mask filters (1-8).

Figure 4 shows the task processing structure of any image input. From starting point it takes less time to compute but quality of processing is low. That is the quick process of contract-time anytime algorithm. Gradually for half process and complete process of the program gives better output.

Figure 5 depicted how we divide every image into eight 3x3 sub-masks to use Sobel operator and use contract-time in different time period. We initially experiment only 3 contracts using these sub-masks and calculate process time in GPU and CPU system

4 Experimental results and analysis

In the experimental setup, we have used AMD FX 8150 CPU, 8 GB RAM with a GTX 550ti GPU, which specification is given in the Table I.

TABLE I. GTX 550TI GPU ENGINE SPECS.

Parameters	Value
CUDA Cores	192
Graphics Clock (MHz)	900
Processor Clock (MHz)	1800
Texture Fill Rate (billion/sec)	28.8
Processor Clock (MHz)	1800
Total amount of shared memory per block	49152 bytes
Maximum number of threads per block	1024
CUDA Driver Version / Runtime Version	7.5 / 7.5

4.1 Experimental Results of Parallel Sobel Detection

To evaluate our proposed edge detection model, we have used a 3840x2400 image [Image 1] and a 1920x1080 image [Image2]. Figure 6 is our sample image [Image 1] for this experiments. Figure 7 and Figure 8 are the two outputs of input we present in Figure 6.

Figure 6. Sample image 1.

Figure 7. Output of [Image 1] for block dimension 16.

Figure 8. Output of [Image 1] for block dimension 32.

Figure 9 and Figure 10 are two outputs for our experiment image, Image I. For large pixel images we capture the real image in 4096x4096 texture and then we send it to the GPU for gray scale conversion and edge detection. CPU execution time is the total time to read the image and printing the output. GPU time is the time for kernel that is calculating the total time to execute the kernel in GPU. We have taken block dimension 16 and 32 to calculate threads. Maximum amount of threads for 16 block dimension is 256 and for 32 block dimension is 1024 threads. For our experiment, we have calculated the blocks using Equation 3. .

$$\text{Block Size} = (X_1, Y_1) \tag{3}$$

$$X_1 = W/B_d$$

$$Y_1 = H/B_d$$

Where width and height of image are from input image and block dimension is our default value. Width of image = *W*, Height of image = *H*, and Block dimension = B_d.

```
Image loaded as an OpenGL texture ,L²
Texture size 4096 x 4096
threads in block = 256,blocks = 65536
Time for the kernel: 209.877853 ms
 Completed host processing
CPU execution time = 486.000000 ms
```

Figure 9. Console Result for 16 block dimension.

```
Image loaded as an OpenGL texture ,T·
Texture size 4096 x 4096
threads in block = 1024,blocks = 16384
Time for the kernel: 230.908798 ms
 Completed host processing
CPU execution time = 508.000000 ms
```

Figure 10. Console Result for 32 block dimension.

A conventional CPU based Sobel edge detection is able to compute the edge detection for our sample Image,1920 x 1080 pixel image. However, it is unable to compute a 3840 x 2400 pixel image. Figure 11 depict the information related to conventional CPU based implementation.

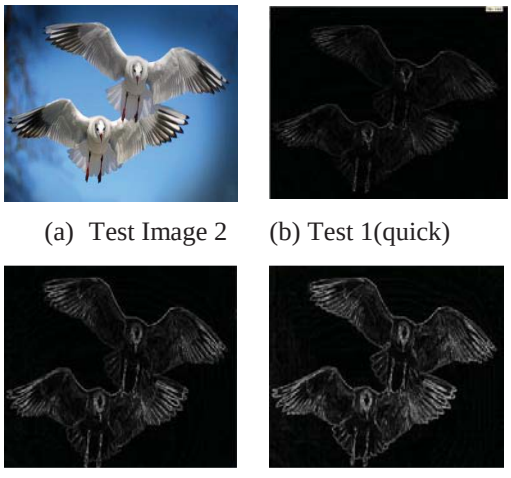
```
Sobel Enhancement
Exception in thread "main" java.lang.IllegalArgumentException: Pixel s
        at java.awt.image.PixelInterleavedSampleModel.<init>(PixelInte
        at Main.getImage(Main.java:145)
        at Main.main(Main.java:71)
```

Figure 11. Conventional CPU Error Output.

4.2 Experiment with Parallel Contract-time Anytime and Sobel Detection

For experimenting our contract time algorithm with Sobel, we have used a 1920x1080 image [Image 2], which is presented in Figure 12, in two different 16 and 32 block dimensions.

Figure 12. Sample image 2.

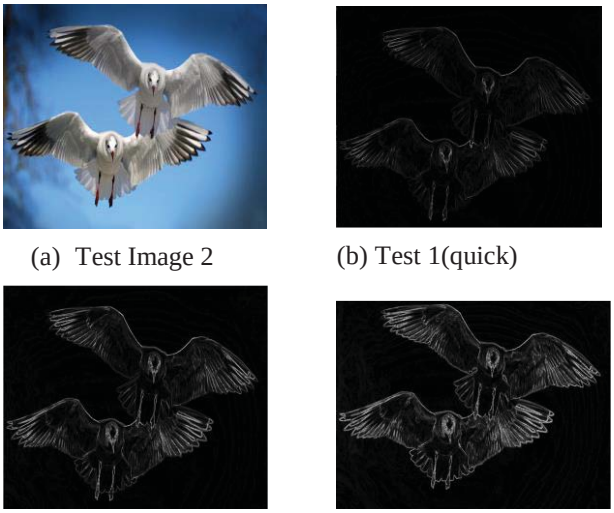


(a) Test Image 2 (b) Test 1(quick)

(c) Test 2(half) (d) Test 3(full)

Figure 13. Three contract-time process test for 32 block .dimension Test (b, c, and d).

Figure 13 shows that we have used 32 block dimension for test contract-time anytime algorithm. Where we have got different output and execution time from 3 contract of our program. Test 1 is the result of quick process. Test 2 is for half process and Test 3 is for Full process. Test 1 takes comparatively less time than test 3. Same goes for the image tested in 16 block dimension that's showed in Figure 14. Here, we have changed our block dimension to 16 to measure the computation time when we are using less amount of



(a) Test Image 2 (b) Test 1(quick)

(c) Test 2(half) (d) Test 3(full)

Figure 14. Three contract-time process test for 16 block dimension test (b, c and d).

blocks.

```
Insert
2 for quick process
4 for half process
6 for full preocess
4
Image loaded as an OpenGL texture ,
Texture size 2048 x 1024
threads in block = 1024,blocks = 2048
Time for the kernel: 27.027777 ms
 Completed host processing
CPU execution time = 108.000000 ms
```

[Output 1] Anytime algorithm output

```
Sobel Enhancement
Output file name: >> output1920_output
Output file extension : >> jpg
Done > Sobel
480 ms
```

[Output 2] Conventional CPU output

Figure 15. Sample output of parallel anytime and conventional CPU

Figure 15 illustrates the sample outputs of our process. Where process of our algorithm is anytime algorithm output [Output 1], here user defines which process will be calculated. This output is for 16 block dimension half process. Output 2 is a sample output for conventional CPU based program.

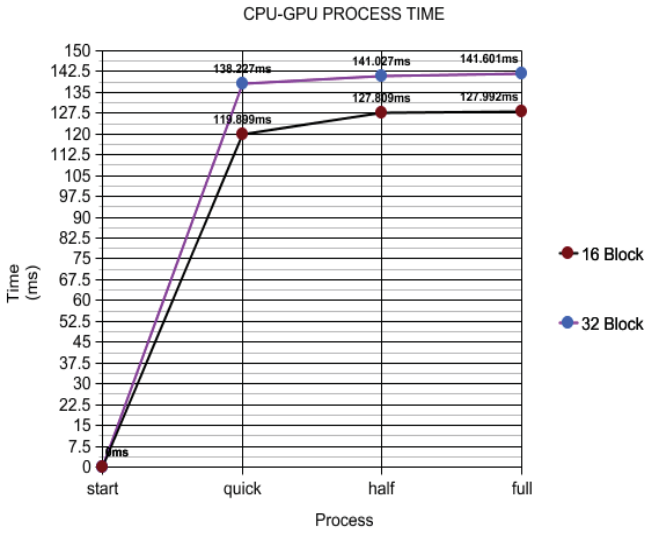


Figure 16. Process versus Time for 16 and 32 block dimension.

Figure 16 graph represents the detailed comparison between 16 and 32 block dimension with respect to execution time.

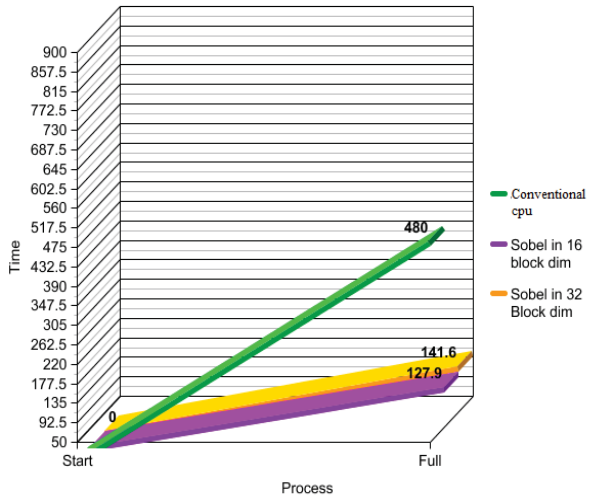


Figure 17. Process versus Time of 1920 x 1024 Input Image.

Figure 17 presents the comparison graph of conventional CPU based program and both 16 and 32 block dimension in GPU. Where conventional CPU based program took 480 ms and for our sobel in parallel process its 141.6ms and 127.9ms for 32 block dimension and 16 block dimension to compute a 1920 x 1024 pixel image.

Table II represents the comparison of our program with conventional CPU programming. Comparing with the CPU program we have calculated speedup of our program and for 16 block low it is 4.003x and for high quality edge detection it is 3.75x.

TABLE II. PROCESS EXECUTION TIME COMPARISON

Process	CPU-GPU time (ms)	Threads in Block	Block size	CPU Execution Time(ms)	Speedup
16 block dimension					
Quick	119.899	256	8192	480	4.003x
Half	127.809				3.7x
Full	127.992				3.75x
32 block dimension					
Quick	138.227	1024	2048		3.47x
Half	141.027				3.4x
Full	141.601				3.38x

5 Conclusion

This paper presented a new parallel edge detection method using Sobel and Contract Anytime Algorithm. As a parallel platform we utilize an NVIDIA GTX GPU and 8 Core CPU. For sample test images, we calculate the execution time of proposed CPU-GPU parallel method and conventional CPU based algorithm. In addition, by varying thread and block sizes, we observed the effect of computation time. Experimental results show that the proposed parallel implementation exhibits above 4X speedup over the conventional serial implementation.

6 References

[1] M. B. Ahmad and T. S. Choi, "Local threshold and boolean function based edge detection," IEEE Transactions on Consumer Electronics, vol. 45, no. 3, pp. 674–679, 1999.

[2] T. A. Abbasi and M. U. Abbasi, "A novel FPGA-based architecture for Sobel edge detection operator,"International Journal of Electronics, vol. 94, no. 9, pp. 889–896, 2007.

[3] W. Kywe, D. Fujiwara, and K. Murakami., "Scheduling of Image Processing Using Anytime Algorithm for Real-time System," Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, vol.3.2006.

[4] Ogawa, Kohei, Y. Ito, and K. Nakano. "Efficient Canny Edge Detection Using a GPU." First International Conference on Networking and Computing. 2010.

[5] M. Rahul, and A. A. Saba. "Anytime Algorithms for GPU Architectures." 2011 IEEE 32nd Real-Time Systems Symposium, 2011.

[6] Baxter, J W, J. Hargreaves, N. Hawes, and R. Stolkin. "Controlling Anytime Scheduling of Observation Tasks." Research and Development in Intelligent Systems XXIX: 219-24, Oct 2012.

[7] W. Kywe, D. Fujiwara, and K. Murakami, "An Approach to Linear Spatial Filtering Method based on Anytime Algorithm for Real-time Image Processing," 18th

International Conference on Pattern Recognition (ICPR'06), v ol. 4, no. 12, 2012.

[8] J. Uddin, E. Oyekanlu, C.H. Kim, and J. M. kim, "High Performance Computing for Large Graphs of Internet Applications using GPU," International Journal of Multimedia and Ubiquitous Engineering, Vol. 9, No. 3, pp. 269-280, 2014.

[9] Rostov Kremlin Available from:
<https://wallpaperscraft.com/wallpaper/rostov_velikij_kreml_rossiya_khram_103672>

[10] Gulls Available from:
<https://wallpaperscraft.com/download/gulls_birds_flying_flapping_106466/1600x900>

ISBN: 1-60132-444-8, CSREA Press ©

SESSION

PARALLEL PROCESSING + GPU and GPGPU BASED UTILIZATION AND SYSTEMS

Chair(s)

TBA

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

Numerical Solutions of Heat and Mass Transfer with the First Kind Boundary and Initial Conditions in Hollow Capillary Porous Cylinder Using Programmable Graphics Hardware

Hira Narang[1], Fan Wu[1], Abisoye Ogunniyan[1]
[1]Computer Science Department, Tuskegee University, Tuskegee, AL, USA

Abstract—*Nowadays, a heat and mass transfer simulation plays an important role in various engineering and industrial fields. To analyze physical behaviors of a thermal environment, we have to simulate heat and mass transfer phenomena. However to obtain numerical solutions to heat and mass transfer equations is much time-consuming. In this paper, therefore, one of acceleration techniques developed in the graphics community that exploits a graphics processing unit (GPU) is applied to the numerical solutions of heat and mass transfer equations. Implementation of the simulation on GPU makes GPU computing power available for the most time-consuming part of the simulation and calculation. The nVidia CUDA programming model provides a straightforward means of describing inherently parallel computations. This paper improves the computational performance of solving heat and mass transfer equations with the first boundary and initial conditions numerically running on GPU. We implemented simulation of heat and mass transfer using the novel CUDA platform on nVidia Quadro FX 4800 and compared its performance with an optimized CPU implementation on a high-end Intel Xeon CPU. The experimental results clearly show that GPU can perform heat and mass transfer simulation accurately and significantly accelerate the numerical calculation with the maximum observed speedups 10 times. Therefore, the GPU implementation is a promising approach to acceleration of the heat and mass transfer simulation.*

Keywords: Numerical Solution; Heat and Mass Transfer; High Performance Computation; General Purpose Graphics Processing Unit; CUDA.

1 Introduction

During the last 4-5 decades, many scientists and engineers working in Heat and Mass Transfer processes have focused their attention to finding solutions both analytically/numerically, and experimentally. To precisely analyze physical behaviors of thermal environments, we need to simulate several heat and mass transfer phenomena such as heat conduction, convection, and radiation. A heat transfer simulation is accomplished by combining multiple computer simulations of such heat and mass transfer phenomena. With the advent of computer, initially the sequential solutions were found, and later when super-computers became available, fast solutions were obtained to above mentioned problems. However, the simulation of heat and mass transfer requires much longer execution time than the other simulations. Therefore, acceleration of the heat and mass transfer simulation is essential to realize a practical large-scale heat and mass transfer simulation.

This paper exploits the computing power of graphics processing units (GPUs) to accelerate the heat and mass transfer simulation. GPUs are cost-effective in terms of theoretical peak floating-point operation rates [1]. Therefore, comparing with expensive cluster, GPUs is a powerful co-processor on a common desktop PC that is ready to achieve a large-scale heat and mass transfer simulation at a low cost. The GPU has several key advantages over CPU architectures for highly parallel, compute intensive workloads, including higher memory bandwidth, significantly higher floating-point throughput. The GPU can be an attractive alternative to CPU clusters in high performance computing environments.

Recent announcement like CUDA [2] by nVidia proved their effort to extend both programming and memory models. CUDA (Compute Unified Device Architecture) is a new data-parallel, C-language programming API that bypasses the rendering interface and avoids the difficulties of classic GPGPU. Parallel computations are instead expressed as general-purpose, C-language kernels operating in parallel over all the points in a domain.

This paper investigates the numerical solutions to Two-point Initial-Boundary Value Problems (TIBVP) of Heat and Mass with the first boundary and initial conditions arising in hollow capillary porous cylinder. These problems find applications in drying processes, under-ground contaminants transport, absorption of nutrients in human bodies, transpiration cooling of space vehicles at re-entry into atmosphere, and many other science and engineering problems. Although traditional approaches of parallel-distributed processing have been applied with advantage to the solutions of some of these problems, no more seem to have explored the high performance solutions to these problems with compact multi-processing capabilities of GPU, which is multi-processors technology on a chip. With the power of this compact technology and develop relevant algorithms to find the solution of TIBVP with the first boundary and initial conditions and compare with some of the existing solutions to simple known problems. All of our experimental results show satisfactory speedups. The maximum observed speedups are about 10 times.

The rest of the paper is organized as follow: Section II introduces some previous related work; Section III describes the background on GPU and CUDA briefly; Section IV presents the mathematical model of heat and mass transfer and numerical solutions to heat and mass transfer equations; Our experimental results are presented in Section V; Finally Section VI concludes this paper with our future direction.

2 Related Work

The simulation of heat and mass transfer has received much attention for years. And there is much work related to this field, such as modeling and dynamic simulation. Here we just refer to some recent work closely related.

Soviet Union was in the fore-front for exploring the coupled Heat and Mass Transfer in Porous media was researched as a part of chemical engineering discipline, and major advances were made at Heat and Mass Transfer Institute at Minsk, BSSR. Later England and India took the lead and made further advances in terms of analytical and numerical solutions to certain problems. Later Narang and Rajiv [4-9] explored the wavelet solutions and Ambethkar [10] explored the numerical solutions to some of these problems.

With the programmability of fragments on GPU, Krüger et al. [11] computed the basic linear algebra problems, and further computed the 2D wave equations and NSEs on GPU. Bolz et al. [12] rearranged the sparse matrix into textures, and utilized them multigrid method to solve the fluid problem. Similarly, Goodnight et al. [13] used the multigrid method to solve the boundary value problems on GPU. Harris [14, 15] solved the PDEs of fluid motion to get cloud animation.

GPU is also used to solve other kinds of PDEs. For example, Kim et al. [16] solved the crystal formation equations on GPU. Lefohn et al. [17] packed the level-set isosurface data into a dynamic sparse texture format, which was used to solve the PDEs. Another creative usage was to pack the information of the next active tiles into a vector message, which was used to control the vertices and texture coordinates needed to send from CPU to GPU. To learn more applications about GPU for general-purpose computations, readers can refer to [18].

3 An Overview of CUDA Architecture

The GPU that we have used in our implementations is nVidia's Quadro FX 4800, which is DirectX 10 compliant. It is one of nVidia's fastest processors that support the CUDA API and as such all implementations using this API are forward compatible with newer CUDA compliant devices. All CUDA compatible devices support 32-bit integer processing. An important consideration for GPU performance is its level of occupancy. Occupancy refers to the number of threads available for execution at any one time. It is normally desirable to have a high level of occupancy as it facilitates the hiding of memory latency.

The GPU memory architecture is shown in figure 1.

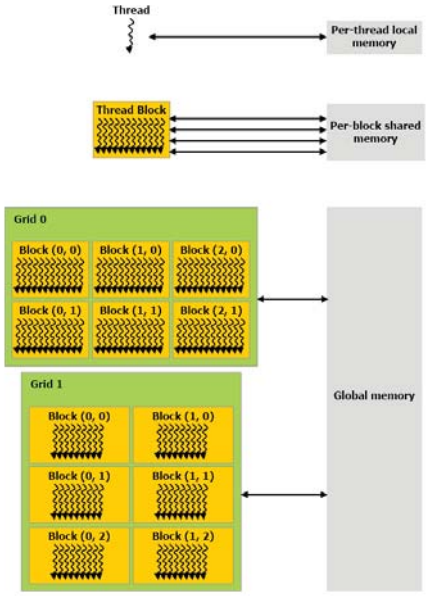


Figure 1: GPU Memory Architecture [2]

4 Mathematical Model and Numerical Solutions of Heat and Mass Transfer

4.1 Mathematical Model

Consider the Heat and Mass Transfer through a porous cylinder with boundary conditions of the first kind. Let the z-axis be directed upward along the cylinder and the r-axis radius of the cylinder. Let u and v be the velocity components along the z- and r- axes respectively. Then the heat and mass transfer equations in the Boussinesq's approximation, are:

$$\frac{\partial T}{\partial t} = k_1 \left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{\partial^2 T}{\partial z^2}\right) + k_2 \left(\frac{\partial C}{\partial t}\right) \quad (1)$$

$$\frac{\partial C}{\partial t} = k_3 \left(\frac{\partial^2 C}{\partial r^2} + \frac{1}{r}\frac{\partial C}{\partial r} + \frac{\partial^2 C}{\partial z^2}\right) + k_4 \left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{\partial^2 T}{\partial z^2}\right) \quad (2)$$

$$a < r > b, z > 0, t \geq 0$$

A prescribed constant temperature and concentration supplie d by the hot plate at the left end $X=0$ of the cylinder, the initi al and boundary conditions of the problem are:

$$r = a, z > 0, t > 0 \quad (3)$$

$$T(a, z, t) = T_a$$

$$C(a, z, t) = C_a$$

$$r = b, z > 0, t > 0 \quad (4)$$

$$
\begin{aligned}
&T(b,z,t) = T_b \\
&C(b,z,t) = C_b \\
&r > 0, z \to \infty, t > 0 \\
&T(r,\infty,t) \to T_\infty \\
&C(r,\infty,t) \to C_\infty
\end{aligned} \tag{5}
$$

Since the cylinder is assumed to be porous, μ_1 is the velocity of the fluid, T_p the temperature of the fluid near the cylinder, T_∞ the temperature of the fluid far away from the cylinder, C_p the concentration near the cylinder, C_∞ the concentration far away from the cylinder, g the acceleration due to gravity, β the coefficient of volume expansion for heat transfer, β' the coefficient of volume expansion for concentration, ν the kinematic viscosity, σ the scalar electrical conductivity, ω the frequency of oscillation, k the thermal conductivity.

From Equation (1) we observe that v_1 is independent of space co-ordinates and may be taken as constant. We define the following non-dimensional variables and parameters.

$$t = \frac{t_1 V_0^2}{4\nu}, z = \frac{V_0 z_1}{4\nu} \tag{8}$$

$$u = \frac{u_1}{V_0}, T = \frac{T_1 - T_\infty}{T_P - T_\infty}, C = \frac{C_1 - C_\infty}{C_P - C_\infty}, P_r = \frac{\nu}{k}, S_c = \frac{\nu}{D'}$$

$$M = \frac{\sigma B_0^2 \nu}{\rho V_0^2}, G_r = \frac{\nu g \beta (T_P - T_\infty)}{V_0^3}$$

$$G_m = \frac{\nu g \beta' (C_P - C_\infty)}{V_0^3}, \omega = \frac{4\nu\omega_i}{V_0^2}$$

Now taking into account Equations (5), (6), (7), and (8), equations (1) and (2) reduce to the following form:

$$\frac{\partial T}{\partial t} + \frac{\partial^2 T}{\partial r^2} - 4\frac{\partial C}{\partial t} + \frac{1}{r}\frac{\partial T}{\partial r} = \frac{4}{P_r}\frac{\partial^2 T}{\partial z^2} \tag{9}$$

$$\frac{\partial C}{\partial t} + \frac{\partial^2 C}{\partial r^2} - 4\frac{\partial T}{\partial t} + \frac{1}{r}\frac{\partial C}{\partial r} = \frac{4}{P_r}\frac{\partial^2 C}{\partial z^2} \tag{10}$$

with

$$t \le 0 \tag{11}$$

$$C(r,z,t) = 0,\ T(r,z,t) = T_0$$

$$t > 0 \tag{12}$$

$$\frac{\partial C(r,z,t)}{\partial z} + k\frac{\partial T(r,z,t)}{\partial z} = 0$$

$$t > 0 \tag{13}$$

$$T(r,\infty,t) = 0,\ C(r,\infty,t) = 0$$

4.2 Numerical Solutions

Here we sought a solution by finite difference technique of implicit type namely Crank-Nicolson implicit finite difference method which is always convergent and stable. This method has been used to solve Equations (9), and (10) subject to the conditions given by (11), (12) and (13). To obtain the difference equations, the region of the heat is divided into a gird or mesh of lines parallel to z and r axes. Solutions of difference equations are obtained at the intersection of these mesh lines called nodes. The values of the dependent variables T, and C at the nodal points along the plane $x = 0$ are given by $T(0,t)$ and $C(0,t)$ hence are known from the boundary conditions.

In the figure 2, Δz, Δr are constant mesh sizes along z and r directions respectively. We need an algorithm to find single values at next time level in terms of known values at an earlier time level. A forward difference approximation for the first order partial derivatives of T and C. And a central difference approximation for the second order partial derivative of T and C are used. On introducing finite difference approximations for:

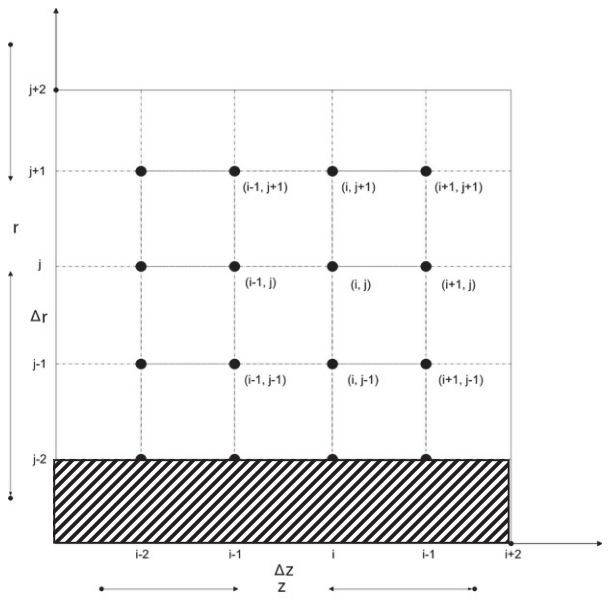


Figure 2: Finite Difference Grid

The shaded portion of the grid represents the hollow in the cylinder. For the purposes of coming up with a numerical

solution for the problem, the height of the hollow is 0.1, while the radius of the cylinder is 1.0.

$$\left(\frac{\partial^2 T}{\partial z^2}\right)_{i,j} = \frac{T_{i+1,j} - T_{i-1,j} + T_{i+1,j+1} - T_{i-1,j+1} - 2T_{i,j}}{2(\Delta z)^2} \quad (14)$$

$$\left(\frac{\partial^2 T}{\partial r^2}\right)_{i,j} = \frac{T_{i+1,j} - T_{i-1,j} + T_{i+1,j+1} - T_{i-1,j+1} - 2T_{i,j}}{2(\Delta r)^2}$$

$$\left(\frac{\partial T}{\partial r}\right)_{i,j} = \frac{T_{i+1,j} - T_{i-1,j} + T_{i+1,j+1} - T_{i-1,j+1}}{4(\Delta r)}$$

$$\left(\frac{\partial T}{\partial t}\right)_{i,j} = \frac{T_{i,j+1} - T_{i,j}}{\Delta t}, \left(\frac{\partial C}{\partial t}\right)_{i,j} = \frac{C_{i,j+1} - C_{i,j}}{\Delta t}, \left(\frac{\partial u}{\partial t}\right)_{i,j} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$

$$\left(\frac{\partial C}{\partial t}\right)_{i,j} = \frac{C_{i+1,j} - C_{i-1,j} + C_{i+1,j+1} - C_{i-1,j+1}}{4(\Delta t)}$$

$$\left(\frac{\partial^2 C}{\partial z^2}\right)_{i,j} = \frac{C_{i+1,j} - C_{i-1,j} + C_{i+1,j+1} - C_{i-1,j+1} - 2C_{i,j}}{2(\Delta z)^2}$$

$$\left(\frac{\partial^2 C}{\partial r^2}\right)_{i,j} = \frac{C_{i+1,j} - C_{i-1,j} + C_{i+1,j+1} - C_{i-1,j+1}}{2(\Delta r)^2}$$

$$\left(\frac{\partial C}{\partial r}\right)_{i,j} = \frac{C_{i+1,j} - C_{i-1,j} + C_{i+1,j+1} - C_{i-1,j+1}}{4(\Delta r)}$$

The finite difference approximation of Equations (9) and (10) are obtained with substituting Equation (14) into Equations (9) and (10) and multiplying both sides by Δt and after simplifying, we let $\frac{\Delta t}{(\Delta z)^2} = r' = 1$ (method is always stable and convergent), under this condition the above equations can be written as:

$$\frac{\partial C}{\partial t} = \frac{1}{2}\left(\frac{U + V - 2(T_{i,j} + C_{i,j})}{(\Delta r)^2} + \frac{U + V}{2r(\Delta r)} + \frac{U + V - 2(T_{i,j} + C_{i,j})}{(\Delta z)^2}\right)$$

$$\frac{\partial T}{\partial t} = \frac{1}{2}\left(\frac{2U + V - 2(2T_{i,j} + C_{i,j})}{(\Delta r)^2} + \frac{2U + V}{2r(\Delta r)} + \frac{2U + V - 2(2T_{i,j} + C_{i,j})}{(\Delta z)^2}\right)$$

Let U = $T_{i+1,j} - T_{i-1,j} + T_{i+1,j+1} - T_{i-1,j+1}$

Let V = $C_{i+1,j} - C_{i-1,j} + C_{i+1,j+1} - C_{i-1,j+1}$ (15)

5 Experimental Results and Discussion

5.1 Setup and Device Configuration

The experiment was executed using the CUDA Runtime Library, Quadro FX 4800 graphics card, Intel Core 2 Duo. The programming interface used was Visual Studio.

The experiments were performed using a 64-bit Lenovo ThinkStation D20 with an Intel Xeon CPU E5520 with processor speed of 2.27 GHZ and physical RAM of 4.00GB. The Graphics Processing Unit (GPU) used was an NVIDIA Quadro FX 4800 with the following specifications:

CUDA Driver Version:	3.0
Total amount of global memory:	1.59 Gbytes
Number of multiprocessors:	24
Number of cores:	92
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	16384 bytes
Total number of registers available per block:	16384
Maximum number of threads per block:	512

Banwitdh:

Host to Device Bandwith: 3412.1 (MB/s)
Device to Host Bandwith: 3189.4 (MB/s)
Device to Device Bandwitdh: 57509.6 (MB/s)

In the experiments, we considered solving heat and mass transfer differential equations in hollow capillary porous cylinder with boundary conditions of the first kind using numerical methods. Our main purpose here was to obtain numerical solutions for Temperature *T*, and concentration *C* distributions across the various points in a cylinder as heat and mass are transferred from one end of the cylinder to the other. For our experiment, we compared the similarity of the CPU and GPU results. We also compared the performance of the CPU and GPU in terms of processing times of these results.

In the experimental setup, we are given the initial temperature T_0 and concentration C_0 at point $z = 0$ on the cylinder. Also, there is a constant temperature and concentration N_0 constantly working the surface of the cylinder. The temperature at the other end of the cylinder where $z = \infty$ is assumed to be ambient temperature (assumed to be zero). Also, the concentration at the other end of the cylinder where $z = \infty$ is assumed to be negligible (≈ 0). Our initial problem was to derive the temperature T_1 and concentration C_1 associated with the initial temperature and concentration respectively. We did this by employing the finite difference technique. Hence, we obtained total initial temperature of ($T_0 + T_1$) and total initial concentration of ($C_0 + C_1$) at $z = 0$. These total initial conditions were then used to perform calculations.

For the purpose of implementation, we assumed a fixed length of the cylinder and varied the number of nodal points N to be determined in the cylinder. Since N is inversely proportional to the step size Δz, increasing N decreases Δz and therefore more accurate results are obtained with larger values of N. For easy implementation in Visual Studio, we employed the Forward Euler Method (FEM) for forward calculation of the temperature and concentration distributions at each nodal point in both the CPU and GPU. For a given array of size N, the nodal points are calculated iteratively until the values of temperature and concentration become stable. In this experiment, we performed the iteration for 10 different time steps. After the tenth step, the values of the temperature and concentration became stable and are recorded. We run the tests for several different values of N and Δz and the error between the GPU and CPU calculated results were increasingly smaller as N increased. Finally, our results were normalized in both the GPU and CPU.

5.2 Experimental Results

The normalized temperature and concentration distributions at various points in the cylinder are depicted in Table 1 and Table 2 respectively. We can immediately see that, at each point in the cylinder, the CPU and GPU computed results are similar. In addition, the value of temperature is highest and the value of concentration is lowest at the point on the cylinder where the heat resource and mass resource are constantly applied. As we move away from this point, the values of the temperature decrease and concentration increase. At a point near the designated end of the cylinder, the values of the temperature approach zero and concentration approach one.

Z	GPU Results	CPU Results
6.000	0.8235960	0.8219550
12.000	0.6032240	0.5999140
18.000	0.4146470	0.4104230
36.000	0.0903610	0.0874080
48.000	0.0235920	0.0222570
54.000	0.0110760	0.0102900
66.000	0.0022830	0.0020780
76.500	0.0007480	0.0007360
82.500	0.0005290	0.0005990
90.000	0.0004410	0.0006580
102.000	0.0004140	0.0011160
111.000	0.0004120	0.0017040
126.000	0.0004120	0.0024570
138.000	0.0003950	0.0019150
147.000	0.0001820	0.0005950
150.000	-	-

Table 1. Comparison of GPU and CPU Results (Tempreture)

Z	GPU Results	CPU Results
6.000	0.0038350	0.0037140
12.000	0.0076340	0.0075130
18.000	0.0094730	0.0093520
36.000	0.0065130	0.0063920
48.000	0.0030940	0.0029730
54.000	0.0018850	0.0017640
66.000	0.0006450	0.0005240
76.500	0.0005750	0.0004540
82.500	0.0011190	0.0009980
90.000	0.0036980	0.0031880
102.000	0.0192070	0.0175870
111.000	0.0558170	0.0528540
126.000	0.2360980	0.2314500
138.000	0.5522080	0.5487840
147.000	0.8816640	0.8807490
150.000	1.0000000	1.0000000

Table 2. Comparison of GPU and CPU Results (Tempreture)

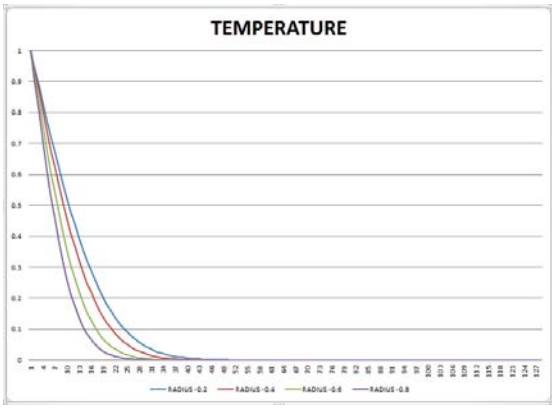


Figure 3: Shows the temperature distribution in the cylinder with 4 different radiuses

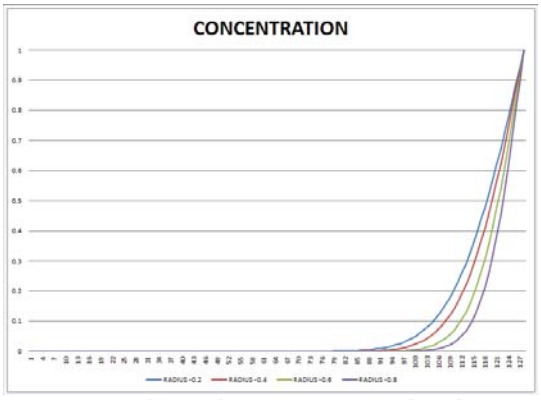


Figure 4: Shows the concentration distribution in the cylinder with 4 different radiuses

Furthermore, we also evaluated the performance of the GPU (NVIDIA Quadro FX 4800) in terms of solving heat and mass transfer equations by comparing its execution time to that of the CPU (Intel Xeon E5520).

For the purpose of measuring the execution time, the same functions were implemented in both the device (GPU) and the host (CPU), to initialize the temperature and concentration and to compute the numerical solutions. In this case, we measured the processing time for different values of N. The graph in Figure 3 depicts the performance of the GPU versus the CPU in terms of the processing time. We run the test for N running from 15 to 1005 with increments of 30 and generally, the GPU performed the calculations a lot faster than the CPU.

- When N was smaller than 512, the CPU performed the calculations faster than the GPU.
- For N larger than 512 the GPU performance began to increase considerably

Figure 5 shows some of our experimental results.

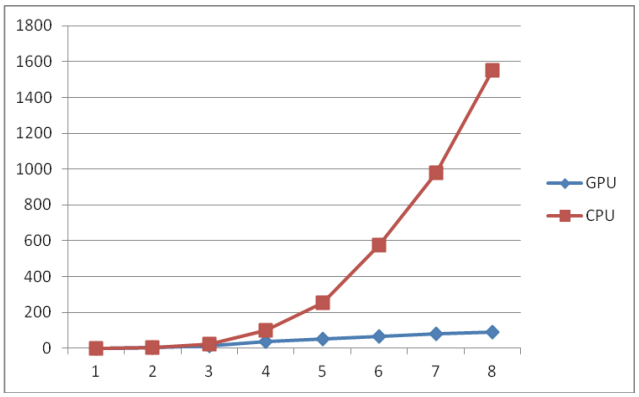


Figure 5: Performance of GPU and CPU Implementations

Finally, the accuracy of our numerical solution was dependent on the number of iterations we performed in calculating each nodal point, where more iteration means more accurate results. In our experiment, we observed that after 9 or 10 iterations, the solution to the heat and mass equation at a given point became stable. For optimal performance, and to keep the number of iterations the same for both CPU and GPU, we used 10 iterations.

6 Conclusion and Future Work

We have presented our numerical approximations to the solution of the heat and mass transfer equation with the first kind of boundary and initial conditions using finite difference method on GPGPUs. Our conclusion shows that finite difference method is well suited for parallel programming. We implemented numerical solutions utilizing highly parallel computations capability of GPGPU on nVidia CUDA. We have demonstrated GPU can perform significantly faster than CPU in the field of numerical solution to heat and mass transfer. Our experimental results indicate that our GPU-based implementation shows a significant performance improvement over CPU-based implementation and the maximum observed speedups are about 10 times.

There are several avenues for future work. We would like to test our algorithm on different GPUs and explore the new performance opportunities offered by newer generations of GPUs. It would also be interesting to explore more tests with large scale data set. Finally, further attempts will be made to explore more complicated problems both in terms of boundary conditions as well as hollow cylinder geometry.

7 References

[1] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A.E. Lefohn, T.J. Purcell,: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26(1) (2007) 80-113.

[2] NVIDIA Corporation. NVIDIA Programming Guide 2.3. Retrieved July, 2009. www.nvidia.com.

[3] A. V. Luikov. Heat and Mass Transfer in Capillary Porous Bodies, Pergamon Press, 1966.

[4] Hira Narang and Rajiv Nekkanti. Wavelet-based Solution to Time-dependent Two-point Initial Boundary Value Problems with Non-Periodic Boundary Conditions, Proceedings of the IATED International Conference Signal Processing, Pattern Recognition & Applications July 3-6 2001, Rhodes, Greece.

[5] Hira Narang and Rajiv Nekkanti. Wavelet-based Solution of Boundary Value Problems involving Hyperbolic Equations, Proceedings from the IATED International Conference Signal Processing, Pattern Recognition & Applications June 25-26, 2002

[6] Hira Narang and Rajiv Nekkanti. Wavelet-based solutions to problems involving Parabolic Equations, Proceedings of the IATED International Conference Signal Processing, Pattern Recognition & Applications 2001, Greece.

[7] Hira Narang and Rajiv Nekkanti. Wavelet-Based Solution to Elliptic Two-Point Boundary Value Problems with Non-Periodic Boundary Conditions, Proceedings from the WSEAS international conference in Signal, Speech, and Image processing Sept 25-28, 2002

[8] Hira Narang and Rajiv Nekkanti. Wavelet-Based Solution to Some Time-Dependent Two-Point Initial Boundary Value Problems with Non-Linear Non-Periodic Boundary Conditions, International Conference on Scientific computation and differential equations, SCICADE 2003, Trondheim, Norway, June 30. July 4, 2003

[9] Hira Narang and Rajiv Nekkanti. Wavelet based Solution to Time-Dependent Two Point Initial Boundary Value Problems with Non-Periodic Boundary Conditions involving High Intensity Heat and Mass Transfer in Capillary Porous Bodies, IATED International Conference proceedings, Gainesville, FL 2004.

[10] Vishwavidyalaya Ambethkar. Numerical Solutions of Heat and Mass Transfer Effects of an Unsteady MHD Free Conective Flow Past an Iffinite Vertical Plate With Constant Suction. Journal of Naval Architecture and Marine Engineering, pages 28-36, June, 2008.

[11] Jens Krüger and Rüdiger Westermann. Linear Algebra Operators for GPU Implementation of Numerical Algorithms. ACM Transactions on Graphics (Proceedings of SIGGRAPH), pages908-916, July 2003.

[12] Jeff Bolz, Ian Farmer, Eitan Grinspun and Peter Schröoder. Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid. ACM Transactions on Graphics (Proceedings of SIGGRAPH), pages917-924, July 2003.

[13] Nolan Goodnight, Cliff Woolley, David Luebke and Greg Humphreys. A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware. In Proceeding of Graphics Hardware, pages 102-111, July 2003.

[14] Mark Harris, William Baxter, Thorsten Scheuermann and Anselmo Lastra. Simulation of Cloud Dynamics on Graphics Hardware. In Proceedings of Graphics Hardware, pages 92-101, July 2003.

[15] Mark Harris. Real-Time Cloud Simulation and Rendering. PhD thesis, 2003.

[16] Theodore Kim and Ming Lin. Visual Simulation of Ice Crystal Growth. In Proceedings of SIGGRAPH/Eurographics Symposium on Computer Amination, pages 86-97, July 2003.

[17] Aaron Lefohn, Joe Kniss, Charles Hansen and Ross Whitaker. Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware. In IEEE Visualization, pages 75-82, 2003.

[18] GPGPU website. http://www.gpgpu.org.

hSA-DS: A Heterogeneous Suffix Array Construction Using D-Critical Substrings for Burrow-Wheeler Transform

Yu-Cheng Liao, Yarsun Hsu
Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan 30013, R.O.C.

Abstract— *Burrow-Wheeler Transform (BWT) algorithm is widely used in data compression and bioinformatics. Mathematically, BWT can be derived from the constructed suffix array. In this work, we analyze the current parallel implementations of SACAs and introduce the first heterogeneous implementation of the SA-DS algorithm on GPU. In order to achieve better performance, we also optimize the radix sort on GPU for our platform. As the result, the optimized radix sort on GPU can significantly decrease processing time compared with the latest Thrust library for sorting millions of keys. Our heterogeneous SA-DS demonstrates up to 4x speedup over the sequential version of SA-DS and has a performance gain up to 2x than the parallel BWT provided by the CUDPP library.*

Keywords: GPGPU, CUDA, Burrow-Wheeler Transform, Compression, SA-DS

1 Introduction

1.1 Motivation

Burrow-Wheeler Transform (BWT) [1] is an algorithm used in data compression techniques like *bzip2* [2]. Mathematically the transform can be obtained from constructing suffix array [3] in linear time [1]. The research of many previous studies on optimizing suffix array construction algorithms (SACAs) in both time and space also greatly improves the Burrow-Wheeler Transform.

For heterogeneous platform, in these days, the prevalence of flexible, programmable and inexpensive general-proposed graphics processing unit (GPGPU) opens a new era of SIMD programming. Consequently, the heterogeneous architecture with GPGPUs has been widely adopted in the field of high performance computing.

By reviewing recent works [4] [5] [6], we know these parallel SACAs adopted a well-known linear time SACA called *skew algorithm*. The famous linear time sequential SACAs are the skew algorithm, *KA algorithm* [7] and Ge Nong *et al.*'s *SA-IS* and *SA-DS* [8]. The skew algorithm has the worst time/space performance among these algorithms. It is interesting to compare between the parallel skew algorithm and the parallel Ge Nong *et al.*'s work. Further more, on the ground of concerning appropriate BWT block sizes for efficient compression, the conventional bzip2 selects the block size from 100K characters to 900K characters for BWT [2] to get the high compression rate along with moderate transforming time. We are motivated to find a better implementation for block size between 100K to 2M characters in this study.

1.2 Goal and Contribution

The objective of this work is to present a heterogeneous version of the SA-DS algorithm accelerated by NVIDIA GPU using CUDA programming model. We package the memory transactions between host and device to obviate the data transfer overhead of the heterogeneous platform. Also, the heterogeneous SA-DS is appended with a kernel to compute the final encoded string for Burrow-Wheeler Transform.

An additional contribution is a custom radix sort based on the Thrust library [9] on GPU. Since our heterogeneous platform equipped with a Tesla k20c graphics card, rather than calling existing Thrust primitives, we simplify the redundant sorting procedures, optimize the kernel and improve the load balance in device to achieve higher throughput.

After the optimizations, our radix sort kernels outperform the Thrust library's by 60%. Comparing the full characters and integers sorting to the latest Thrust library, our method shows up to 77% decrease in time with respect to small sequences comprised of thousands of elements, and up to 23% decrease in time with respect to large sequences comprised of millions of elements. Our heterogeneous SA-DS demonstrates up to 4x speedup over the C++ sequential version and 2x faster compared to the implementation of BWT using the CUDPP library [6] with the block size ranging from 100K to 2M characters in this study.

1.3 Organization

This paper is organized as follow. Section 2 discusses related works. Section 3 introduces Burrow-Wheeler Transform, and suffix array construction algorithms. Section 4 states the design and the implementations. Then, the performance evaluations are presented in section 5. Finally, section 6 describes conclusion.

2 Related Work

As a part of a greater ambition to research the feasibility of lossless data compression on GPU, R.A. Patel *et al.* provide a new approach for suffix array construction based on merge sort [10]. They first use a bitonic sort to sort eight suffixes

within a thread in GPU. Each thread fetches four characters of each suffix in a comparison. If two suffixes reside in one thread have the same prefix and are unable to be sorted, the thread would fetch the next four characters on-the-fly from the global memory. Once all of the threads complete sorting their suffixes, the threads in a block work cooperatively to merge the partitioned suffix array into one complete suffix array. As a result, they report severe degradation in performance while merging large sequence because of branch divergence and frequent global memory access. Further, it cannot take advantage of the relationship between suffixes. For the implementation on GPU, they report a 3x slower than the single thread CPU implementation by Seward *et al.* [11].

In 2013, M. Deo *et al.* brought the parallel DC3 algorithm to GPU. Their work is implemented on discrete GPU and APU respectively using OpenCL. It is inspired by the pDC3 but considered to be the first implementation on modern GPU architecture. They resolve some issues which are encountered while adapting the original pDC3 from the distributed system to heterogeneous platform and optimize the performance of pDC3 on GPU. Their paper also includes a brief explication that we can safely choose to ignore BWT and only discuss SA and its implementation since we are able to derive BWT trivially in one pass in parallel with computing SA.

In 2014, CUDPP library added a new primitive to compute the suffix array of a string. They use the recursive skew algorithm, similar to M. Deo *et al.*, for the suffix array construction on GPU using CUDA. The primitive has the same sorting procedure analogous to M. Deo *et al.*'s work, but in the final merging step, they adopted another merging technique, call *merge path*, presented by O. Green *et al.* [12] which is different from M. Deo *et al.*'s work. According to the author's note, their parallel skew algorithm is 1.35x faster than the fastest implementation on GPU. Their work is the latest implementation we are going to compare with.

In conclusion, to the best of our knowledge, we do not see the implementation of other linear suffix array construction algorithms such as the KA, SA-IS and SA-DS algorithm utilizing the computational power of GPGPU.

3 Background

3.1 Burrow-Wheeler Transform

Burrow-Wheeler Transform (BWT) is discovered by Wheeler in 1984. BWT first produces a list, also called a block, of strings consisting of all the cyclical rotations of the original string. The block is then sorted lexicographically and the last character of each rotation forms the permuted string. $ is the terminal symbol denoting the end of current string and is the lexicographically smallest character.

BWT is aimed at gathering the same characters and the transformed string must be capable of reversing back. For the serial implementation of BWT, Burrow and Wheeler suggested performing a radix sort on first character and second character of every rotations to obtain the preliminary order [1]. On their observation, most of the rotations can be sorted within the preliminary order. The order then followed by a quick sort to distinguish the strings sharing the same prefix.

3.2 BWT and Suffix Array

We first review the content of a suffix array (SA). Consider size-n string $S = s_1 s_2 \dots s_{n-1}\$$, and \$ is the terminal symbol. Let S_i denote the suffix of S ranging from the i-th character to the ending character \$. The suffix array (SA) stores the integers of starting index i that represents S_i for all suffixes in lexicographical order. Which means if the entry $SA[j]$ is i, S_i is the j-th smallest suffix in the string S and hence $\forall k \in [1, j] : S_{SA[k]} \leq S_i$.

From the given SA, BWT result can be conducted simply as the equation: $BWT[i] = S[SA[i] - 1]$. The process is trivial and the BWT can be derived in one pass in parallel for each entry. This relationship allows us to neglect BWT and focus on SA's construction.

3.3 Suffix Array Construction Algorithms

According to the property above, it is safe for us to only study the implementations of SACAs and their parallel implementations on GPGPU. Linear-time SACAs can easily prevail the original implementation in large scale strings. The up-to-date well-known linear-time SACAs use two genres of framework: skew algorithm and two-stage induction.

3.3.1 Skew Algorithm

Skew algorithm is an algorithm recursively constructing SA in linear time. It follows the pattern proposed for suffix tree construction by Farach *et al.* in 1997 [13]. The following is the brief review of the skew algorithm, or DC3 algorithm, from J. Kärkkäinen *et al.*. The skew algorithm consists of three steps. First, considering a string S, it is first reduced by excluding the suffixes starting at position $i \mod 3 = 0$, and the new problem size is $2/3$ of the original input. To construct the SA, sorting is performed by scanning the first three characters of each suffix in the reduced problem and renaming the sorted suffixes with their ranks. If all the names are different, step one is finished and we attain the SA; otherwise, skew algorithm would be recursively applied to the reduced arrays. Secondly, every suffix is the concatenation of the character of the starting position, and the suffix starting at the next position. The remaining SA is attained by radix sorting the first character followed by the entry in constructed SA that represents the following suffix. The last step is to merge the two SAs. Skew algorithm compares the lexicographical order of each suffix in two SAs, and put them in the final complete suffix array.

J. Kärkkäinen *et al.* constructing two suffix arrays with asymmetric length provides the simplicity in step 3. The implementation of skew algorithm by J. Kärkkäinen *et al.* is succinct, many researchers exploit the parallelism of skew algorithm based on their scheme [4] [5] [6]. Skew algorithm is simple, however, the algorithm can only reduce the problem size to one-third of the input size in each recursion.

3.3.2 Two-Stage Induction

Recent two-stage induction algorithms are variants of the SACA proposed by H. Itoh [14]. Since G. Nong *et al.* are dedicated to ameliorate the intrinsic sorting bottleneck of using S-distance lists method in KA algorithm [7], we would discuss their algorithm briefly in the following. SA-IS and SA-DS are the twin algorithms using the same framework. SA-IS consists of more sequential structures, so we concentrate and analyze thoroughly on the implementation of SA-DS algorithm as well as exploiting the potential parallelism in the algorithm.

SA-DS is based on KA algorithm. In addition to classifying two types, we need to further separate the leftmost S-type suffixes (*LMS*) among type S suffixes. Besides, these LMS characters are used to locate the intervals of *LMS-substrings*. As a result, the original S is replaced by a shorter string only comprised of LMS-substrings. The input problem is simpler than KA algorithm because arrays having consecutive type S suffixes are curtailed.

Despite the abbreviated problem size, suffixes in it are variable-length. They propose a new approach established on the radix sort and fixed-length substrings called *D-critical substrings*. A character is a d-critical character if and only if it is an LMS-character; or the character d length after is a d-critical, and no character between them is d-critical where $d \geq 2$. The suffix starts from the character is called a d-critical suffix.

SA-DS constructs the SA using the framework consisting of three steps. First, we can reduce the problem into an array containing the pointer of all the d-critical characters. The distance between any two neighboring d-critical characters is proven to be in $[2, d+1]$. Next we perform radix sort on the leading $d+2$ characters of each d-critical suffix after these suffixes are sorted by their types. If all the names are unique, step one is accomplished and we attain the suffix array of d-critical; otherwise, we have to recursively apply the SA-DS algorithm. Secondly, bucket all of the suffixes in S according to their first character, then initialize a new array for storing the final SA. We assign the buckets orderly to the SA array and record the ending position and the starting position of each bucket. The algorithm puts the sorted S-type suffixes into the correct entries in the final SA. Lastly, SA-DS incorporates one more induction process than KA algorithm for the position of other type S suffixes and remaining type L suffixes. For inducing positions of type L suffixes, the procedure is described in the KA algorithm [7]. For the remaining type S suffixes, each entry $SA[i]$ encountered during the scan, if the suffix $S_{SA[i]-1}$ is S-type, move the suffix to the recorded ending position of its bucket in SA. The LMS-suffix originally resides at the end of the bucket is swapped.

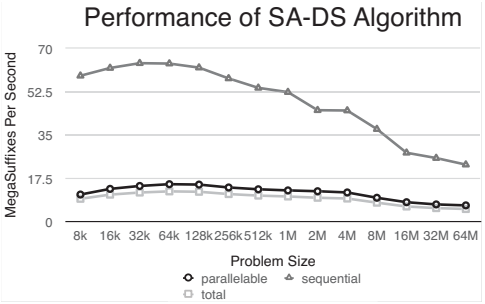


Fig. 1: Performance of two portions of SA-DS algorithm, clearly the overall performance is bounded by the parallelizable part.

Apparently SA-DS algorithm has better performance in large scale strings owing to the high reduction rate in recursive sorting step. Between KA algorithm and SA-DS, SA-DS provides a framework that gives the simplicity of the sorting step and is suitable for parallelization on GPU.

Now, we extract the possible parallelizable parts of SA-DS algorithm for the intent of mapping these portions onto GPU. Classification of suffixes' type is self-reliant. The type of a suffix is determined by itself and its next suffix. Comparison for all of the suffixes in the input string S can be done parallelly on GPU. LMS-characters are parallelly distinguishable by simply checking every previous suffix's type for type S suffixes. Since blocks in LMS-substrings are disjoint, d-critical substrings within each LMS-substrings can be located in parallel. Fast fixed-length radix sorting on GPU already exists. Assigning LMS-suffixes is also parallelizable since we only have to maintain the order in the same bucket. Lastly, the induction step contains dependencies in each iteration, therefore this part remains sequential and can not be parallelized. Fig. 1 shows the performance of different portions. It's clear that the overall performance of the SA-DS algorithm is bounded by the parallelizable part in all ranges of the input size. From the evaluation, more than four-fifth of the execution time is parallelizable. According to Amdahl's law [15], it is possible to improve performance up to 5x.

4 Design and Implementation

In this section, we describe the method we used to parallelize the original SA-DS algorithm. Table 1 presents the pseudo code of the SA-DS algorithm and the steps that are packed into kernels for GPU are also marked.

Table 1: The SA-DS algorithm pseudo code

	SA-DS(S, SA)	kernel
	//S is the input string	
	//SA is the output suffix array for S	
1	Find d-critical substrings in S	(1-3)
2	Reduce the original problem into a shortened P_1	(4-6)
3	Radix sort the d-critical substrings in P_1	(7-10)
4	Name each d-critical substring by its rank to get S_{DC}	(11-14)
5	**if** (allUniqueNames) SA_{DC} = S_{DC}	
6	**else** SA-DS(S_{DC}, SA_{DC}) //recursion	
7	Induce SA from SA_{DC} step 1	(15-19)
8	Induce SA from SA_{DC} step 2,3 //on CPU	
9	**end**	

4.1 Parallelizing SA-DS

In table 1, there are six primary sections culled from the sequential SA-DS algorithm. We do not describe the process of parallel radix sort since it is already explained in detail in D.G. Merrill *et al.*'s work [16].

4.1.1 Locating D-Critical Substrings

It includes three kernels responsible for classifying types of characters, identifying leftmost S-type characters, and assigning additional d-critical substrings between any two adjacent LMSs by the given d distance. In order to classify types of characters, each thread in the kernel is in charge of one particular character in the input string appointed by its thread index and block index. Every thread compares the current assigned character with the next character. The classification is done if the next character is lexicographically greater than the current character, which means the associated suffix is lexicographically smaller than the next suffix. If the character is equal to the next one, comparison between the next character and the next of the next is taken recursively. The kernel identifies leftmost S-type characters by fetching the type of characters, and comparing the type with the preceding character's type. If the fetched type is S-type and the preceding type is L-type, the character is a leftmost S-type character. Finally, continuous d-critical substrings inside independent blocks bounded by neighboring leftmost S-type characters are denoted parallelly using multiple threads in the kernel. After these kernels, there is an array T storing types of characters and another array $C_{boolean}$ in the same size of input string S storing '1' or '0'. For any entry containing '1' in an array indicates the starting position of a d-critical substring.

4.1.2 Shrinking Problem

The d-critical substrings can be assembled into a shortened array storing starting indexes of d-critical substrings called P_1. From the last section we know if an entry of $C_{boolean}$ holds a '1', the corresponding index is the starting position of a d-critical substring. The kernels first examine each entry in $C_{boolean}$, then exclude the entries containing '0', and aggregate the index of the entries containing '1' into an abbreviated array P_1.

4.1.3 Naming and Constructing SA

The sorted d-critical substrings are stored lexicographically in global memory. Initially, each thread in the kernel is responsible for one particular entry in the sorted P_1. It loads the first $d+2$ characters of its assigned index and the first $d+2$ characters of its previous index, and determine whether these two sets of $d+2$ characters are different. If two sets of the characters are different, the thread stores a '1' in the corresponding entry in a temporary array N; otherwise, stores '0'. Next, we use the similar three steps described in section 4.1.2. Instead of distributing the keys, the third kernel distributes the scanned ranks as values to each entry in N. The following kernel scatters the rank of each d-critical substring according to the index in P_1 to the final suffix array.

4.1.4 Inducing SA Step 1

In those sorted d-critical substrings, we only use the substrings starting with leftmost S-type characters. The dependency among these suffixes merely resides in an individual bucket. To extract LMS-suffixes from sorted d-critical suffixes, we use the same approach explained in section 4.1.2. A kernel marks the LMS-suffixes by inspecting the type of the target character and its previous character's type similar to section 4.1.1. Later the following three kernels remap the marked LMS-suffixes into a shortened array. The subsequent kernels collect the number of names/characters of each bucket, then scan the numbers once to acquire the index offsets by accumulating the size of buckets. Moreover, the LMS-suffixes are placed on the tail of each bucket. After all of the necessary variables are calculated, a kernel launches threads seeded by the offset of its bucket to fill the LMS-suffixes in the correct entries in the suffix array for inducing.

4.2 Optimization

In the heterogeneous SA-DS, the sorting stage using radix sort on GPU consumes most of the execution time. Rather than using the existing radix sort provided by the Thrust library, we rewrite the three steps of radix sort for customization on our Tesla k20c GPU. We fix the decode digits in one iteration to be four digits and implement a simple dynamic terminal policy on host that stops the radix sort kernels when our keys are sorted. The termination is determined by how many kernels that sorting four digits at a time are required for covering the first non-zero bit from MSB in all of the keys. We first find the position of the first non-zero bit, calculate the length of bits from the position we found to the LSB, and then divide the length by four. This terminal policy can prevent a plethora of useless kernel calls.

The radix sort kernels in Thrust library launch blocks or threads according to static profile created by themselves.

Table 2: Hardware specifications

CPU	Intel® Xeon® CPU E5620 @ 2.4GHz 4C/8T × 2
RAM	4GB DDR3-1066MHz × 6
GPU	Nvidia® Tesla™ K20c × 1
HDD	WD 2TB 7200RPM × 2 (RAID 1)

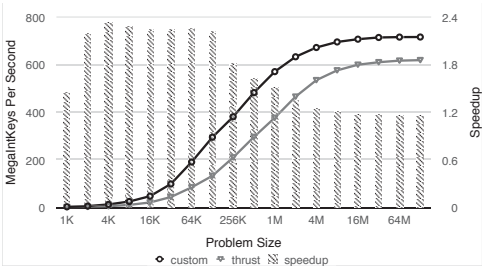


Fig. 2: Performance and speedup of sorting integers

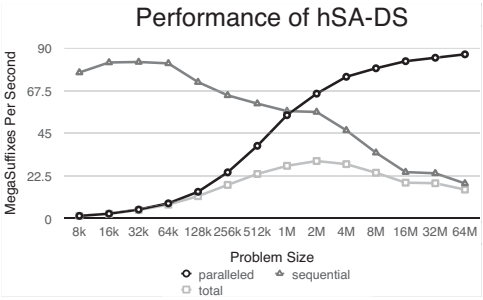


Fig. 3: Performance of the heterogeneous SA-DS, the parallelized portion gains improvement.

However, the load balance in each block among different input array sizes should be also considered. We configure the number of blocks using the size of current input array in our radix sort. With this strategy, our radix sort can have better response time for variable size of input array.

5 Evaluation

We investigate the implementations including the original sequential SA-DS algorithm, the skew algorithm implemented in the CUDPP library, and our heterogeneous SA-DS using CUDA. These implementation are performed and compared on the same hardware platform listed in table 2. The evaluations focus on the execution time of a set of kernels, including memory transfer between host and device.

5.1 Radix Sort

We evaluate the execution time for sorting characters of the three kernels including kernel configuration and execution time but excluding data transfer time between host and device. The input keys are already transfered to GPU's global memory since the overhead of data transfer for the two implementations is the same. We benchmark the kernels by using *nvprof* profiling tool. The result is the average of one hundred executions. The outcome shows our radix sort is 1.31x, 1.37x, and 1.61x faster than the Thrust library in upsweep, toplevelscan, and downsweep kernel respectively when processing one million characters.

We test the scalability of the complete radix sort. Fig. 2 shows the performance of sorting integers with varying sizes of input array. In the Thrust library, the radix sort encounters a performance drop for the problem size around 64K to 2M keys. Comparing with Thrust library, our customized radix sort achieves 4.3x speedup at input of 64K keys, 2x on average with character keys, 2.3x at 4K keys and 1.7x on average with integer keys.

5.2 Heterogeneous SA-DS

We first analyze the performance curves of parallelized portion and the intact sequential portion. The parallelized portion accounts for any additional overheads of memory transfer between host and device on kernel launch.

Fig. 3 shows the performance of each part in heterogeneous SA-DS. As we can see, the performance curve shows the parallelized portion gains vast improvement compared with the performance of the parallelizable part depicted in Fig. 1. With the utilization of GPU, the overall performance of SA-DS no longer suffers from the bound set by the sorting steps. The speedup compared with original SA-DS is 3.7x faster on average when the input problem is large enough.

We choose four datasets downloaded from the Internet with different properties. The content in a text file directly impacts the performance of SA construction. Although these datasets do not take into account of every condition of sorting suffixes, the four representative datasets provide us comprehensive measurements for the normal usage of SA construction.

Enwiki dataset is downloaded from the wikidepia website [17]. It is dumped from the English Wikipedia. Linux kernel tarball is the latest Linux-4.1 kernel, and it contains the source code of Linux kernel. The content can be regarded as random characters. Enwiki abstract is different from the case 1, it contains the abstracts of English wikipedia as million lines of websites. Lastly, we generate strings with different sizes varying from 8K to 32M characters. Fig. 4 shows that the speedup of the hSA-DS rises from negative 7x at small inputs to a steady positive 3.7x at large input. For random characters, because our heterogeneous SA-DS is benefited form the parallel sorting stage, it outperforms sequential SA-DS in most of the problem sizes. However, the inducing stage requires more iterations to construct the complete suffix array. Consequently, the overhead of CPU to construct long suffix array degrades the performance at large input sequence.

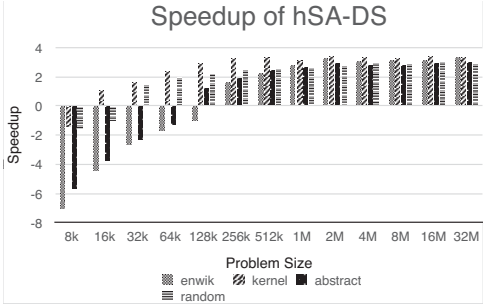


Fig. 4: Speedup of the hSA-DS for four datasets.

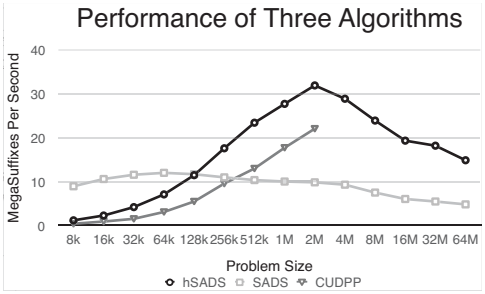


Fig. 5: Comparison between the SA-DS, DC3 on GPU and the hSA-DS.

5.3 Comparisons with CUDPP Library

CUDPP library utilizing NVIDIA GPU with CUDA programming model is considered to be the fastest implementation of parallel DC3 algorithm. The result is shown in Fig. 5. The *readme* file of the CUDPP library describes that their BWT can not process strings larger than 1M characters, but we test up to 2M characters to generate the curve since we are interested in performance for strings with sizes smaller than 2M characters. Finally, the figure shows that our hSA-DS has the best performance for size of strings smaller than 2M characters.

6 Conclusion

Our hSA-DS improves the performance of the original SA-DS by parallelizing its slowest portion. The heterogeneous platform using both GPU and CPU is the best choice for our algorithm since the sequential portion must be performed on a powerful CPU. The customized radix sort further optimizes the distributed workloads of each processing elements, and incorporates a dynamic terminal strategy for keys with different length. As the result, our customized radix sort on GPU gains up to 2.3x and 4x speedup with respect to integer keys and character keys compared to the Thrust library. The hSA-DS algorithm can obviate the performance bound incurred by sequential sorting overhead, and gain up to 3.7x speedup over the sequential SA-DS, and up to 2x speedup over the parallel skew-algorithm-based BWT. The hSA-DS has the best performance for block sizes ranging from 100K to 2M characters.

7 Acknowledgement

The authors thank the support from MOST under grant 104-2220-E-007-006.

8 References

[1] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.
[2] "bzip2-1.0.6," 2015. [Online]. Available: http://www.bzip.org
[3] U. Manber and G. Myers, "Suffix arrays: a new method for on-line string searches," *siam Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993.
[4] M. Deo and S. Keely, "Parallel suffix array and least common prefix for the gpu," in *ACM SIGPLAN Notices*, vol. 48, no. 8. ACM, 2013, pp. 197–206.
[5] F. Kulla and P. Sanders, "Scalable parallel suffix array construction," *Parallel Computing*, vol. 33, no. 9, pp. 605–612, 2007.
[6] "Cudpp-2.2," 2014. [Online]. Available: http://cudpp.github.io
[7] P. Ko and S. Aluru, "Space efficient linear time construction of suffix arrays," in *Combinatorial Pattern Matching*. Springer, 2003, pp. 200–210.
[8] G. Nong, S. Zhang, and W. H. Chan, "Two efficient algorithms for linear time suffix array construction," *Computers, IEEE Transactions on*, vol. 60, no. 10, pp. 1471–1484, 2011.
[9] J. Hoberock and N. Bell, "Thrust: A parallel template library," 2010, version 1.8.1. [Online]. Available: http://thrust.github.io/
[10] R. Patel, Y. Zhang, J. Mak, A. Davidson, J. D. Owens, *et al.*, *Parallel lossless data compression on the GPU*. IEEE, 2012.
[11] J. Seward, "On the performance of bwt sorting algorithms," in *Data Compression Conference, 2000. Proceedings. DCC 2000*. IEEE, 2000, pp. 173–182.
[12] O. Green, R. McColl, and D. A. Bader, "Gpu merge path: a gpu merging algorithm," in *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 2012, pp. 331–340.
[13] M. Farach, "Optimal suffix tree construction with large alphabets," in *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*. IEEE, 1997, pp. 137–143.
[14] H. Itoh and H. Tanaka, "An efficient method for in memory construction of suffix arrays," in *String Processing and Information Retrieval Symposium, 1999 and International Workshop on Groupware*. IEEE, 1999, pp. 81–88.
[15] 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*. ACM, 1967, pp. 483–485.
[16] D. G. Merrill and A. S. Grimshaw, "Revisiting sorting for gpgpu stream architectures," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*. ACM, 2010, pp. 545–546.
[17] "Enwiki," 2015. [Online]. Available: https://dumps.wikimedia.org/enwiki/

Periodic Steady State Solution of Power Networks using the Current Injections Method and Parallel Processing based on GPUs

Marcolino H. Díaz-Araujo[1], **Aurelio Medina-Rios**[1], **Ernesto Magaña-Lemus**[1], **Antonio Ramos-Paz**[1]
[1]Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo, Francisco J. Múgica S/N, Ciudad Universitaria, C.P. 58030, Morelia, Mich., México

Abstract — *This paper details an efficient, fast and non-iterative algorithm for the simulation of the steady state response of power networks under non-sinusoidal conditions in the frequency domain. The algorithm uses the injection current method, LU decomposition, and parallel processing techniques based on Graphic Processing Units (GPUs). It is shown that for the explicit harmonics representation, the implementation on a GPU-based platform becomes an efficient computational resource to find the steady state solution since floating-point operations and repetitive calculations increase in proportion to the number of harmonics and size of the network; both related with the computer effort.*

Keywords — Injection current method, LU decomposition, parallel processing, Graphic Processing Unit.

1 Introduction

The complexity of power systems has increased in direct proportion with the network size and the presence and continuous incorporation of nonlinear elements. In addition, the need for more efficient, accurate and robust simulation techniques has also increased. Engineers need to determine many variables and quantities of interest, such as harmonic levels in the power network [1].

Admittedly, the periodic steady state solution of power systems can be obtained in three main frameworks, i.e. frequency domain, time domain and hybrid frequency-time domain, respectively [2]. A concise review is given in [2] regarding the main advantages, drawbacks, formulation and convergence characteristics related to the different methods belonging to the frames of reference above indicated. In particular, this contribution deals with a method for the efficient periodic steady state solution of power networks in the frequency domain.

A widely used methodology to compute the steady state solution of power networks is based on a unified iterative approach where phases, linear and nonlinear components, number of harmonics explicitly represented, harmonic cross-coupling and unbalance effects are combined together for the entire system [3]. Its application to larger scale three-phase sytems may lead, however, to excessive computer effort as high dimension problems may need to be solved [4].

On the other hand, phasor equivalents of linear and non-linear power system waveforms consist of an infinite number of terms. For the purposes of simulation, this representation is truncated to a finite number of terms. This procedure is intuitive and it should be verified that the frequency set used in the analysis provides accurate results by increasing the number of frequencies, then repeating the simulation, and checking that the simulation results change by a negligible amount. In practical terms, it is sufficient for harmonic analysis to account for the first 50 harmonics. However, it may result on a time consuming process.

Due to the time critical nature of such computation, the need for parallel processing for the simulation of realistic systems becomes a necessity. Parallel processing is a technique that allows one program to execute multiple tasks concurrently. A thread consists of a stream of control that can execute its instructions independently so a multi-treaded process, or program, can perform numerous tasks concurrently [5].

Over the last 15 years, significant changes have ocurred to summarize and review the relationship between power system analysis and high performance computing. By way of example, in [6] a study for large-scale transient stability simulation based on the massively parallel architecture of multiple GPUs is made.

In this paper, the current injection method [1] is applied to obtain the periodic steady state analysis of power systems under non-sinusoidal conditions by means of parallel processing based on GPUs and LU decomposition.

2 Harmonic Analysis

2.1 Linear Circuit Analysis

The steady state solution of power systems operating under sinusoidal and non-sinusoidal conditions can be obtained using phasor analysis. The power network is solved for each frequency of interest as opposed to only the fundamental frequency [7].

In general, a linear circuit analysis operating under non-sinusoidal conditions can be represented by the following set of linear equations:

$$\begin{bmatrix} i_h^1 \\ i_h^2 \\ \vdots \\ i_h^j \\ \vdots \\ i_h^N \end{bmatrix} = \begin{bmatrix} y_h^{1,1} & y_h^{1,2} & \cdots & y_h^{1,j} & \cdots & y_h^{1,N} \\ y_h^{2,1} & y_h^{2,2} & \cdots & y_h^{2,j} & \cdots & y_h^{2,N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ y_h^{j,1} & y_h^{j,2} & \cdots & y_h^{j,j} & \cdots & y_h^{j,N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ y_h^{N,1} & y_h^{N,2} & \cdots & y_h^{N,3} & \cdots & y_h^{N,N} \end{bmatrix} \begin{bmatrix} v_h^1 \\ v_h^2 \\ \vdots \\ v_h^j \\ \vdots \\ v_h^N \end{bmatrix} \quad (1)$$

where i^j is the current phasor for frequency h injected at node j; $y^{i,j}$ is the equivalent admittance for frequency h between nodes i and j *(mutual when $i \neq j$ and self when $i=j$)*; v is the voltaje phasor for frequency h at node j, and N is the number of nodes of the network.

In compact form (1) can be written as,

$$I_h = Y_h V_h \quad (2)$$

where h is the harmonic, I_h is the harmonic current injection vector, Y_h is the harmonic equivalent admittance matrix, and V_h is the harmonic voltage vector.

2.2 The Harmonic Current Injection Method

The harmonic current injection method is widely used to carry out harmonic propagation studies in distribution systems [7]. The salient features of the method are outlined below:

- Build Y_h of the power system including the contribution for all sources and loads. A different Y_h must be calculated for each harmonic h.
- Obtain I_h by extracting the term of the appropiate frequency from each nonlinear load.
- Use (2) to calculate V_h. Both magnitude and phase information are important. If a time domain solution required for each bus voltage, the calculated harmonics are superimposed.

2.3 LU Decomposition

Since (2) should be repetitively used, once for each harmonic. It is advisable to form Y_h with an algorithm being time and memory efficient. For instance, triangular factorization may be applied. The triangular factors of Y_h and the voltages are calculated by forward and backward substitution, respectively.

In this expression L and U are the lower left and upper right triangular factors of Y_h. The vector W is solved by forward substitution, and the vector V_h is subsequently calculated by backward substitution.

2.4 Parallel Processing Based on GPU

NVIDIA is one of the leading manufactures of GPUs. Architectures Fermi and Kepler are the most widely used for parallel processing. The GPU used in this research is the Tesla C2075 with Fermi architecture [8], discussed next.

The first Fermi architecture based GPU, implemented 3.0 billion transistors and features up to 512 CUDA cores. A CUDA core executes a floating point or integer instruction per clock for a thread. The 512 CUDA cores are organized in 16 streaming multiprocessors (SM) of 32 cores each. The GPU has 64-bit memory partitions, for a 384-bit memory interface; supporting up to a total of 6 GB of GDDR5 DRAM memory. A host interface connects the GPU to the CPU via PCI-Express. The GigaThread global scheduler distributes thread blocks to SM thread schedulers. Fig. 1, shows the elements of SM and a core. CUDA is the hardware and software architecture that enables NVIDIA GPUs to execute programs written with C, C++, Fortran, Open CL, DirectCompute, and other languages [9]. A CUDA program calls parallel kernels. A kernel executes a process in parallel across a set of parallel threads. The programmer or compiler organizes these threads in thread blocks and grids of thread blocks. The GPU instantiates a kernel program on a grid of parallel thread blocks. Each thread within a thread block executes an instance of the kernel, and has a thread ID within its thread block, program counter, registers, per-thread private memory, inputs, and output results.

A CUDA program has two parts: the serial part and the parallel part. In the serial part, no parallelism exists and the instructions are executed in the CPU. In the parallel part, which involves massive data parallelism, instructions are executed in the GPU. A high-level view of the CUDA programming model is illustrated in Fig. 2.

2.5 Component Modelling

To assemble the elements of a power system into a bus impedance matrix for each harmonic, a frequency dependent model for each element must be developed [11]. This section summarizes the typical representations of common network components for harmonic analysis.

1) *Transmission lines:* For the case of a transmissionline, the total resistance and inductive reactance of the line is included in the series arm of the equivalent-π and the total capacitance to neutral is divided equally between its shunt arms.
2) *Generators:* These are considered to be linear components whose harmonic impedance is

$$Z_G = R\sqrt{h} + jX_d h \quad (3)$$

where R is derived from the generator power losses and X_d is the generator subtransient reactance.

3) *Transformer:* These are considered to be linear components whose harmonic impedance is

$$Z_T = R\sqrt{h} + jX_t h \quad (3)$$

where R is derived from the tranformer power losses and X_t is the transformer's short-circuit reactance.

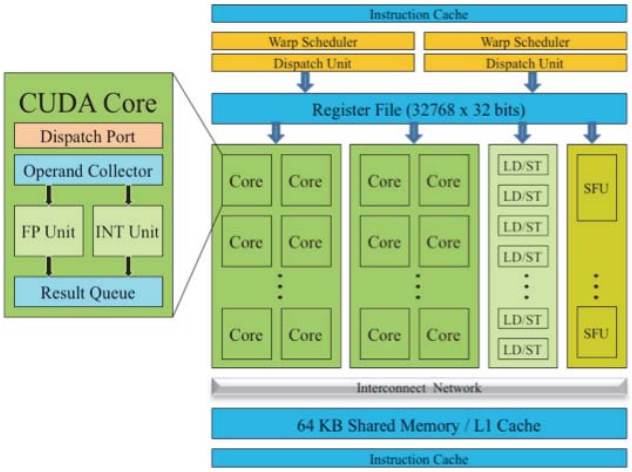


Figure 1. SM and core processor scheme.

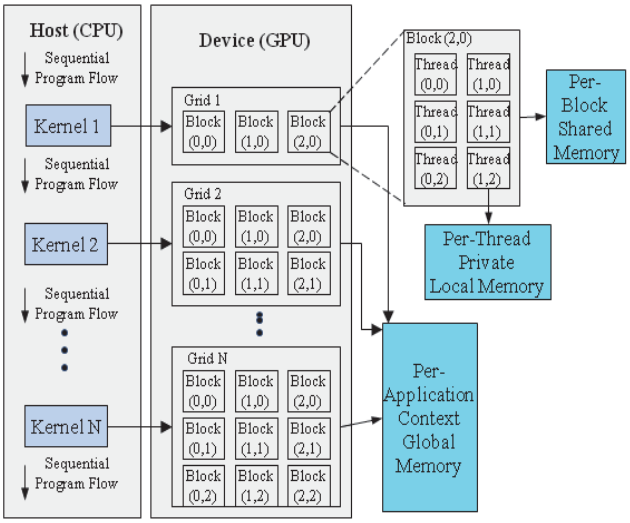


Figure 2. Schematic of execution of a NVIDIA´s CUDA programming model.

4) *Capacitor banks:* These are considered to be passive components, where

$$Z_C = -jV^2/hQ \tag{5}$$

where V is the line voltage and Q is the reactive power.

5) *Passive loads:* Linear passive loads do not produce harmonics but have a significant effect on the system frequency response. A general model for passive load is given in [12]

$$Z_L = RX_L/(R + X_L) + X_S \tag{6}$$

where

$$R = V^2/P \tag{7}$$

$$X_L = jhR/6.7(Q/P - 0.74) \tag{8}$$

$$X_S = j0.073hR \tag{9}$$

6) *Non-linear loads:* These are represented by a harmonic current injection source. Harmonic current injection sources are used to represent the harmonic contributions from static VAR compensators (SVCs), induction arc furnaces, rectifiers and electronic devices. For example, a SVC is represented by the harmonic current injection given by,

$$I_h = (\%_h)I_1 \tag{10}$$

where $\%_h$ is a percentage of the current at fundamental frequency given by

$$I_1 = (Q/\sqrt{3}V)\, e^{j(theta + pi/2)} \tag{11}$$

Where *theta* is the voltage angle obtained from a conventional power flow, and *pi/2* is the required phase shift, since the current leads or lags the voltage by *90°*.

3 Harmonic Propagation Method

The algorithm for the harmonic propagation in the power network combines a conventional power flow study with the injection current method and LU decomposition.

Fig. 3 shows the flow chart for the harmonic propagation method. It is basically composed by three blocks, i.e. the data block that reads the parameters of the power system, the power flow block that performs a conventional power flow study and the harmonic voltage block that determines the harmonic propagation throughout the system.

Some parts of the algorithm are executed sequentially and some parts in parallel. The system data block is programmed sequentially. Then two tasks are simultaneously run. Each task is performed by one thread in the CPU (OpenMP). One of the threads (thread 1) performs the power flow study meanwhile the other thread (thread 0) copy the system data from the CPU to the GPU. These two tasks are executed in parallel and have different computation time, so they have to be synchronized. To synchronize this part of the algorithm a flag is used. It is initially flag = 0 and changes to flag = 1 when the power flow concludes. If the power flow study has no finished yet, thread 0 will have to wait until thread 1 finishes its process. The last part of the algorithm is executed in the GPU (CUDA). For each harmonic, it is necessary to obtain the equivalent admittance matrix, the current injection vector, and solve for the harmonic voltage vector. Superposition effects are accounted to obtain the final result. The following steps summarize the complete method:

- Find the steady state solution given by a conventional power flow study.
- With the resultant voltages at fundamental frequency, compute the passive equivalent circuit.
- Obtain the driving point impedance seen from node where the non-linear load is connected.
- Solve (2) for each frequency to get the final result by superposition of effects.

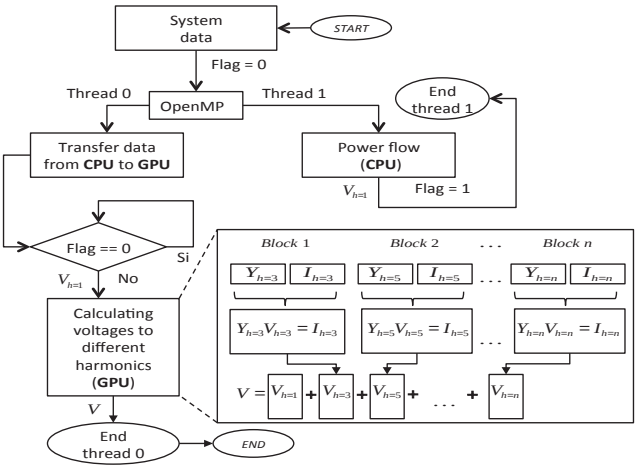


Figure 3. Algorithm of the proposed method.

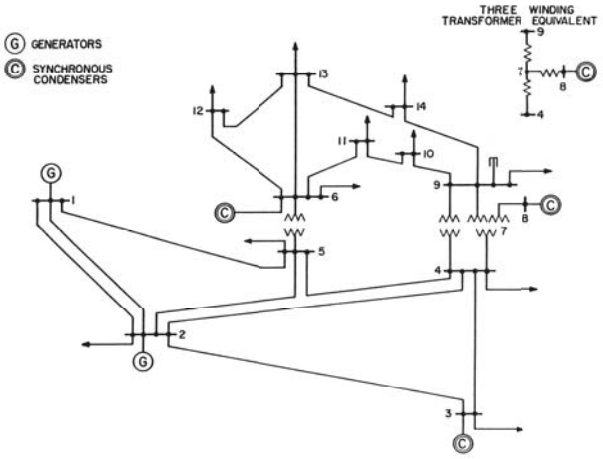


Figure 4. IEEE-14 bus test system.

4 Test Case

The test system of Fig. 4 has been used to illustrate the performance of the implemented method. The electric power system includes three SVCs connected in three diferent buses. The maximum magnitudes of harmonic current injected by the SVCs are given in Table I. The fundamental power flow solution is given in Table II. The SVCs are considered to be delta connected, hence no zero sequence harmonic current is injected into the system.

The impedance of the capacitor bank at node 9 is given by

$$Z_C = -j(1.1005)^2/h0.29 = -j4.1762/h$$

and the load in node 4 is represented by the equivalent impedance

$$Z_L = RX_L/(R + X_L) + X_S$$

where,

$$R = (1.024)^2/0.47 = 2.2310$$
$$X_L = jh2.2310/6.7(0.039/0.478 - 0.74) = -jh0.5057$$
$$X_S = j0.073h(2.2310) = jh0.1628$$

TABLE I. MAXIMUM AMPLITUDE OF HARMONIC CURRENTS IN SVC [7]

Harmonic	*% of fundamental*
5	5.05
7	2.59
11	1.05
13	0.75
17	0.44
19	0.35
23	0.24
25	0.20

TABLE II. FUNDAMENTAL FREQUENCY POWER FLOW (P.U.)

Node	$\lvert V\rvert$	θ	P_G	Q_G	P_D	Q_D
1	1.060	0	2.385	0	0	0
2	1.045	-5.109	0.400	0.122	0.217	0.127
3	1.010	-12.91	0	-0.182	0.942	0.190
4	1.024	-10.72	0	0	0.478	-0.039
5	1.023	-9.079	0	0	0.076	0.016
6	1.070	-14.53	0	-1.250	0.112	0.075
7	1.083	-14.03	0	0	0	0
8	1.090	-14.03	0	0.037	0	0
9	1.100	-15.68	0	0	0.295	0.166
10	1.087	-15.77	0	0	0.090	0.058
11	1.075	-15.32	0	0	0.035	0.018
12	1.155	-18.07	0	0	0.061	0.016
13	1.136	-17.53	0	0	0.135	0.058
14	1.170	-19.08	0	0	0.149	0.050

By using the same procedure, the parameters for the rest of load buses are obtained. The generator reactance for the slack bus is $Z_G = jh0.0001$ and for regulated buses are $Z_G = jh0.001$.

For the SVC at bus 14, the fundamental frequency current is given by

$$I_{SVC} = (0.40/\sqrt{3}\cdot 1.1704)\, e^{j(-03331 + 3.1415/2)} = 0.1973\angle 70.914°$$

In Table I, the harmonic currents are given as percentage of the fundamental component.

The system is solved for each frequency of interest with (2), where I_h have values different from zero only in entries where the non-linear loads are connected. Table III shows the harmonic voltages in percentage of the fundamental.

TABLE III. NETWORK HARMONIC VOLTAGES

V \ h	*5%*	*7%*	*11%*	*13%*	*17%*	*THD*
$\lvert V_{10}\rvert$	0.5548	0.6397	0.1646	0.0727	0.0238	0.8662
θ_{10}	129.60	104.52	-10.29	-19.272	-23.88	
$\lvert V_{11}\rvert$	0.2875	0.3267	0.0806	0.0346	0.0105	0.4441
θ_{11}	128.89	103.72	-11.29	-20.38	-25.22	
$\lvert V_{12}\rvert$	1.1192	0.7802	0.4406	0.3717	0.2815	1.5527
θ_{12}	149.58	148.30	153.31	154.19	155.32	
$\lvert V_{13}\rvert$	0.9679	0.6763	0.3441	0.2946	0.2244	1.3182
θ_{13}	147.67	143.67	151.85	153.07	154.29	
$\lvert V_{14}\rvert$	1.5306	1.1408	0.3504	0.3292	0.2644	2.0175
θ_{14}	141.39	129.81	144.64	149.31	151.55	

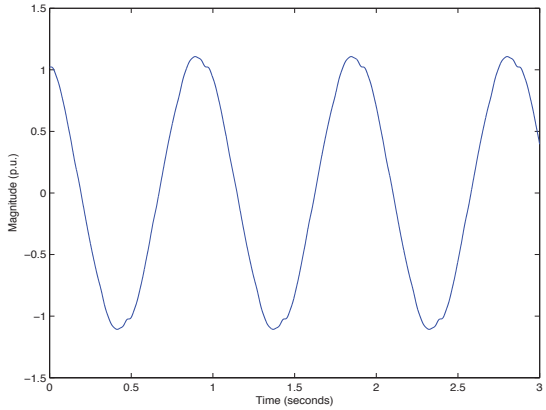


Figure 5. Distorted voltage waveform at node 14.

4.1 Harmonic Propagation in Larger Systems

The parallel code has been applied to larger systems to show the relevance of the harmonic injection current method processed in parallel. The IEEE-57, 118 and 300 test bus systems have been solved. Table IV shows the results obtained by the sequential and the parallelized algorithm. The first column shows the node number, the second the time for data reading of power system, and the third is the time consumed by the power flow study. Columns fourth and fifth show the computation time of the current injection method processed en parallel (PCI) an sequential (SCI), respectively. Columns sixth and seventh show the processing time for the complete parallel (TPCI) and sequential (TSCI) code, respectively. Last two columns show the speed-up of the complete algorithm and the speed-up of the current injection method, respectively.

The comparison of the fourth column to the fifth shows the importance of using parallel processing based on GPU for the current injection method. The best speed-up is 12.95 times for the IEEE-300 bus test system and 4.84 times for the complete algorithm. For the case of the IEEE 118 bus test system the results were 6.94 and 3.83, respectively, and for the case of the IEEE 57 bus test system were 1.11 and 1.07, respectively. It is clear that the speed-up significantly increases with the size of the power network. The speed-up can be improved if the entire algorithm and not just to the current injection algorithm are solved by using parallel processing.

5 Conclusions

The application of parallel GPU-computing in harmonic propagation studies in large electrical networks has been presented.

The algorithm is non-iterative, and a solution is always obtained. The algorithm exploits the injection current method and LU decomposition. The introduction of the parallel processing technique has been very effective in reducing the required CPU time and in increasing memory efficiency. Even for the small test system analyzed, it has shown that the parallel algorithm is faster than the sequential algorithm. The algorithm implemented on a Tesla C2075

TABLE IV. CPU TIME IN LARGER SYSTEMS

IEEE system	*Data*	*Power Flow*	*PCI*	*SCI*	*TPCI*	*TSCI*	*Speedup SCI/PCI*
57	0	21	46	51	67	72	1.11
118	1	56	52	361	109	418	6,94
300	2	1250	592	7667	1844	8919	12.95

GPU improves efficiency from 1.11 to 12.95 times for the injection current method and from 1.07 to 4.84 times in total simulation, as compared with a conventional algorithm processed in series.

6 References

[1] G. T. Heydt, Electric Power Quality, Stars in a Circle Publications, May 1991.
[2] A. Medina, J. Segundo, P. Ribeiro, W. Xu, K. L. Lian, G. W. Chang, V. Dinavahi, N. R. Watson, "Harmonic Analysis in Frequency and Time Domain", IEEE Transactions on Power Delivery, Vol. 28, Issue 3, July 2013, pp. 1813-1821.
[3] J. Arrillaga, A. Medina, M. Lisboa, and M. Cavia, "The Harmonic domain. A Frame of Reference for Power System Harmonic Analysis", IEEE Transactions, vol. 10, pp. 433-440, Feb 1995.
[4] J. Segundo, A. Medina, "Periodic Steady State Solution of Electric Systems Including UPFCs by Extrapolation to the Limit Cycle", IEEE Transactions on Power Delivery, Vol. 23, No. 3, July 2008, pp. 1506-1512.
[5] H. F. Jordan and G. Alaghband, Fundamentals of Parallel Processing, Pearson, Aug 2002.
[6] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-Scale Transient Stability Simulation of electrical Power Systems on Parallel GPUs", IEEE Transactions, vol. 23, pp. 1255-1266, Nov 2011.
[7] E. Acha and M. Madrigal, Power Systems Harmonics, John Wiley & Sons, New York, 2001.
[8] NVIDIA, "Specification: Tesla C2075 GPU Computing System", 2011.
[9] NVIDIA CUDA: Programming Guide, 2015.
[10] "Modeling and Simulation of the Propagation of Harmonics in Electric Power Networks", IEEE Transactions, vol. 11, pp. 452-465, Jan 1996.
[11] CIGRE Working Group, "Harmonics, Characteristic Parameters, Methods of Study, Estimates of Existing Values in the Networks", Electra, No. 77, pp. 35-54, July 1981.
[12] T. J. E. Miller, Reactive Power Control in Electric Systems, John Wiley & Sons, New York, 1982.
[13] D. Owens, M. Houston, and D. Luebke, "GPU Computing", IEEE Proceedings, vol. 96, pp. 879-899, May 2008.

7 Acknowledgment

Financial support by Consejo Nacional de Ciencia y Tecnología (CONACYT) is gratefully acknowledged. The first author wants to acknowledge the scholarship granted by CONACYT for his doctoral studies at the División de Estudios de Posgrado, Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo, Morelia, México.

A Modular Application of Runtime Performance Analytics Using GPU and CPU Execution on Planetary Formation Simulation

Philip M. Westhart[1], Kevin Walsh[2], and Ben Abbott[1]

[1]Southwest Research Institute®
P.O. Drawer 28510
San Antonio, TX 78238, USA
philip.westhart@swri.org
ben.abbott@swri.org

[2]Southwest Research Institute®
1050 Walnut St
Boulder, CO 80302, USA
kevin.walsh@swri.org

Abstract - *The Symplectic Massive Body Algorithm (SyMBA) for planetary formation simulation was targeted for optimization including runtime performance analytics and execution on a graphics processing unit (GPU). The approach identified specific functions that would benefit from GPU acceleration and migrated this limited functionality to execute on the GPU. Benchmarks saw speedups of up to 26.48x (541s from 14327s). The overall performance highly depended on the nature of the test case. To accommodate these data-dependent performance differences, runtime performance analysis was integrated into the software to allow the decision of whether to execute on the central processing unit (CPU) or GPU to be made at runtime. The runtime performance analysis was tested and was able to successfully use the more efficient execution environment.*

Keywords: GPU, OpenCL, Runtime performance analytics, Planetary formation

1 Introduction

The fast pace of technological advances continues to improve the power of modern day machines. In line with these improvements, new computationally-intensive problems have emerged. These problems may involve simulations of large numbers of particles and exhaustively detailed flow modeling. The targeted software of this optimization effort falls into this category, involving the physics simulation of thousands to millions of particles on time scales ranging from thousands to hundreds of millions of years. The high demands of these problems occasionally fall outside the capabilities of today's computers, taking orders of magnitude longer than target times. The optimization of these problems are intrinsically bound by Amdahl's law, which states that the maximum reduction in time by optimizing a subcomponent cannot exceed the time spent in the subcomponent before optimization. An ideal optimization should therefore aim to broadly optimize components to reduce this limitation. The discussed approach aims to utilize a broad range of optimization techniques including graphics processing unit (GPU) execution, runtime performance analytics, and other techniques as applied to the Symplectic Massive Body Algorithm (SyMBA) problem.

2 Background

The gravitational N-body code, SyMBA, has led to numerous advances in planet formation studies [1]. It is valuable to the planetary science community due to its energy conservation properties. Owing to its symplectic algorithm, it can support the number of time steps and encounters necessary for billion year integrations of planetary building blocks interacting, colliding, and growing. However, this code had not previously been ported to run on GPU hardware.

Work has shown planet formation models have a strong dependence on the number of particles considered [2], and parallelization attempts have not been successful for running multi-processor jobs. Due to the chaotic nature of these solar system dynamics problems there is a need for numerous simulations to build statistics about the system behavior (in a recent work Fisher and Ciesla 2014 used 50 simulations [3]). Current simulations can take months, and thus there are strong limitations on problems to attack based on available hardware and manpower. Both of these issues point to a need for optimizing this code to run on an accelerated platform such as a GPU.

The purpose of an N-body algorithm is to calculate the acceleration on each particle caused by the gravitational interaction with all of the other particles. The calculation only requires data about the location and mass of all of the particles in the system at the time of the calculation. At the most basic level, the calculation has no data dependencies and is fully parallelizable. Since this calculation is a standard part of many different celestial mechanics software, there are published designs of different approaches to memory management and data structures/organization on the GPU to maximize the speedup (Elsen et al. 2006 [4], Portegies et al. 2007 [5], Dindar et al. 2012 [6], Grimm and Stadel 2014 [7]).

3 Methods

3.1 Initial Profiling

To understand where the software was spending most of its execution time, the application was run through a software profiler. The software profiler indicated two particular functions within the code that consumed the majority of the execution time.

The first function iterates across all particles and calculated the instantaneous acceleration for each particle. The instantaneous acceleration was calculated as the summation of each particle's interaction with all other massive particles in the system. In a simulation where all particles are massive, this function experienced execution times proportional to the square of the number of particles in the system (i.e. $O(N^2)$).

The second function checks to see if each particle is having an encounter with a massive particle. Again, in the simulation case where all particles are massive, the execution time followed an $O(N^2)$ relationship with the total number of particles.

Both of these functions are highly-parallelizable in nature. Each function iterates across every particle in the system and performs a function related to every other particle in the system. The highly-parallelizable nature of these functions was a strong indicator that these would be ideal candidates for GPU-execution.

3.2 OpenCL Implementation

To accomplish the GPU-execution, the targeted functions were migrated from the original Fortran 77 implementation into an OpenCL implementation. A wrapper function was written in C to handle the OpenCL interface calls, namely explicit declaration and allocation of variables on the execution device (in our case, a Tesla K40 GPU), as well as explicit migration of data onto the device. The target functions were then coded into OpenCL kernels.

3.3 Runtime Performance Analytics

In GPU execution, the transfer of data between the central processing unit (CPU) and the GPU incurs a large amount of overhead. The gains achieved through parallel execution must outweigh this memory transfer cost in order for GPU-execution to be a viable option. If the memory cost of GPU-execution cannot be overcome, faster execution times can be achieved by exploring the parallelism within the CPU. This is less preferable since GPUs offer parallelism on the scale of 1000's of simultaneous operations while current CPUs typically offer less than 32 simultaneous operations. The knowledge of which execution environment performs better might not be known prior to runtime.

In the SyMBA simulations, the benefits of parallelism depend on N, the number of total particles in the system, and M, the number of massive particles in the system. Different simulation scenarios can have significantly different performance based on the values of M and N. However, these variables can also change over the course of the simulation as particles collide and merge. To account for the dependency on M and N, a runtime performance analysis was implemented.

The runtime performance analysis used a sample-based approach to dynamically determine which execution environment, the GPU or the CPU, performed better for the current dataset. This was accomplished by periodically executing the code on both environments and comparing the execution times. The interval at which this dual execution was performed varied based on the historical results such that the interval was extended if a consistent trend was identified to minimize the overhead of executing on both environments. Determining the best environment at runtime allows for the same code to be used for a variety of input datasets. Datasets better suited for GPU execution can be run on the same program as datasets better suited for CPU execution and datasets that change which environment performs better during runtime. The last case can occur when particles merge enough to reduce overall particle count to a point where the gains of GPU parallelization are no longer outweighed by the overhead of data transfer.

3.4 Additional Optimizations

Since the GPU execution was implemented on a single function level, a large portion of the code remained executed on the CPU. Many functions involved too many data structures and would have incurred too large of an overhead to calculate on the GPU. These functions were targeted for general CPU optimizations such as loop transformations, optimized search algorithms, and control flow restructuring.

3.5 Problem Sets

To evaluate the various optimizations, three representative datasets were tested and benchmarked. The first dataset featured 444 total initial particles and 23 massive initial particles. The second dataset featured 5100 total initial particles with 100 initial massive particles. The final dataset featured 10,000 total initial particles with all of the particles initially being massive. These test cases cover the range of plausible test cases of interest from early to late solar system creation. The variation between the total particle count and the massive particle count also yields a large range of execution time per iteration (i.e. how long it takes to calculate the particle physics for a single time step). In initial tests, the 10,000 test particle case took the longest amount of time per iteration, while the 444 test particle test case took the shortest amount of time per iteration.

4 Results and Discussion

The overall speedup achieved varied greatly between test cases. The 10,000 particle test case was observed to complete 26.48x faster, while the 5100 particle test case completed 2.10x faster, and the 444 particle test case completed 2.34x faster. The runtime performance analysis was observed to favor GPU execution for the 10,000 test case, while favoring the CPU for the other two test cases. When runtime performance analytics were not used, the 5100 particle and 444 particle test cases were observed to run approximately 10x *slower on the GPU* than their original CPU execution times. The runtime performance analytics was able to successfully determine this and direct execution to the ideal environment for each dataset. Final execution times and speedups are summarized in Table 1.

Table 1. Execution Times and Speedups

	10,000 particles All massive	**5,100 particles 100 massive**	**444 particles 23 massive**
Baseline Time (s)	14,327	93,600	25,600
Final Wall Clock Time (s)	541	44,656	10,923
Overall Speedup Compared to Original Baseline	26.48	2.10	2.34

The 10k test case saw significant gains from the GPU execution. This was due to the $O(N^2)$ nature of the problem. In general, as the number of test particles (N) and the number of massive particles (M) increase, the computations grow by an order of N*M, while the data transfer cost grows linearly in relation to N. Therefore, the larger test cases experienced greater speedups on the GPU while the smaller test cases performed faster on the CPU.

5 Summary and Conclusions

The SyMBA code for simulating planetary formation was successfully optimized and achieved speedups ranging from 2.10x to 26.48x faster depending on the problem set. Runtime performance analytics were used to dynamically determine whether the optimized code should execute on a GPU or on a CPU. The runtime performance analytics successfully diverted execution to the more efficient environment, allowing for a single code base to be used for problems that span a wide range of performance-affecting parameters. We expect that these approaches can be leveraged to successfully optimize a variety of other problems with similar runtime dependent performance characteristics.

6 References

[1] M. J. Duncan, H. F. Levison and M. H. Lee, "A Multiple Time Step Symplectice Algorithm For Integrating Close Encounters," *The Astronomical Journal*, pp. 2067-2077, October 1998.

[2] D. P. O'Brien, A. Morbidelli and H. F. Levison, "Terrestrial Planet Formation With Strong Dynamical Friction," *Icarus*, pp. 39-58, 2006.

[3] R. A. Fischer and F. J. Ciesla, "Dynamics of the Terrestrial Planets From a Large Number of N-body Simulations," *Earth and Planetary Science Letters*, pp. 28-38, 2014.

[4] E. M. H. V. V. E. D. P. H. a. V. P. Elsen, "N-Body Simulation on GPUs," in *Supercomputing 06 Conference*, 2006.

[5] S. R. B. a. P. G. Portegies Zwart, "High Performance Direct Gravitational N-body Simulations on Graphics Processing Unit," *ArXiv Astrophysics e-prints*, February 2007.

[6] S. L. Grimm and J. G. Stadel, "The GENGA Code: Gravitational Encounters in N-body Simulations with GPU Acceleration," *The Astrophysical Journal*, November 2014.

[7] S. Dindar, E. B. Ford, M. Juric, Y. I. Yeo, J. Gao, A. C. Boley, B. Nelson and J. Peters, "Swarm-NG: a CUDA Library for Parallel N-body Integrations with Focus on Simulations," *New Astronomy*, vol. 23, pp. 6-18, September 2013.

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

SESSION

DISTRIBUTED ALGORITHMS AND PROCESSING + BIG DATA ANALYTICS + REAL-TIME ALGORITHMS

Chair(s)

TBA

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

Optimization of Machine Learning on Apache Spark

Rohit Taneja(IBM), Rajaram B. Krishnamurthy(IBM), Gang Liu(IBM)
11400 Burnet Road, Austin, Texas, USA 78758

Abstract - *Wide adoption of Spark for running big data analytics jobs not only requires focus on the functional performance of applications, but also on their operational efficiency with optimal usage of hardware resources. Since Spark has a large number of tunables, a bottom up approach to finding the optimal runtime by varying Spark executors and Spark executor cores can create an explosion of tuning runs for a given application because of the multiplicative nature of possible configurations. Instead, we use a hybrid top-bottom approach by first characterizing the application and then custom tuning the Spark environment to further enhance performance.*

We have experimented extensively with Spark's tunables and share our tuning methodology by providing a detailed walk-through of an Alternating Least Squares Based Matrix Factorization application. Using our methodology, we have been able to improve runtimes by a factor of 2.22X. We have successfully applied this methodology to complex Spark workflows consisting of Spark SQL and ML Pipelines, and achieved substantial performance improvements. Our methodology has been applied to a variety of cluster architectures to validate our approach.

Keywords: Apache Spark, Machine learning, Performance evaluation and optimization, Matrix Factorization

Submission: Regular Research Paper

1 Introduction

Data sets are growing more rapidly than ever: every day more than 2.5 exabytes of data is created, while at the same time the world's technological capacity to store information has roughly doubled every 40 months since the 1980s[1]. To leverage the power of big data, scientists and engineers have been working on innovative solutions for high performance big data engines. Apache Spark[2] has quickly emerged as one of the most popular in-memory big data processing engines for being easy to use, fast and exhaustive with its built-in modules for streaming, SQL, machine learning and graph processing.

For a big data analytic application running on the Spark framework, the efficient use of underlying cluster resources is vital and often requires a lot of time to apply the necessary tuning in order to achieve the efficiency. Determining the optimal performance point influences total cost of ownership and performance per dollar of applications. In fact, without understanding the characteristics of the workload, efficient parallel execution on a distributed computing platform may not be achieved. To minimize execution times and to achieve performance benefits from parallelism in a Spark cluster environment, characterization in terms of application's resource usage is very crucial. This also involves the identification of critical sections and hardware bottlenecks in order to fine tune the application before deployment on a distributed computing platform.

The rest of the paper is organized as follows: In Section 2, we have discussed about the abstraction that Spark provides and understood the vital parameters in deploying workloads on Spark. In Section 3, we have evaluated the popular machine learning model: Alternating Least Squares (ALS) based Matrix Factorization (MF)[3], on a Spark cluster. Finally we have described the various optimization iterations that we performed on the MF application and conclude our paper in Section 4.

2 Background

2.1 Apache Spark

Apache Spark is usually launched on top of an existing Hadoop Cluster with Hadoop file system spanning worker nodes, while master node drives the work-flow and provide management services. Spark provides distributed computing through working on collection of objects called resilient distributed dataset (RDD). RDDs can be created with a file in the Hadoop file system or by transforming the existing RDD. This abstraction allows in-memory caching of data, but at the same time makes spark more sensitive to data locality. RDDs are kept as partitions spread across the worker nodes in a cluster, and a driver program is initiated to manage the user application. Worker nodes execute the tasks on these partitions in parallel. These set of tasks are grouped into spark stages and various stages are grouped together to be called as a Spark Job, which is the highest layer of abstraction in Spark framework.

For the optimization process, the understanding of key spark parameters is necessary:-

- Spark executor Instances: This parameter decides the number of executors per node. A worker node can have more than one executor, but for the purpose of this paper, we use 1 executor per worker.
- Spark executor cores: Number of threads or logical CPUs per executor on a worker node.
- Spark executor memory: The maximum heap size per executor on a worker node.

- Spark shuffle location and manager: The shuffle operation in Spark writes the 'map' output before performing the 'reduce' on the data. Depending upon the available memory, intermediate shuffle data may remain entirely in memory or get spilled to shuffle location, which is usually HDDs or SSDs. Selection of the shuffle manager based on the characterization of an application can also provide substantial gain in performance. There are three types of shuffle managers: Hash based, Sort based and Tungsten-sort.
- RDD persistence storage level: Caching of RDD partitions across nodes in a cluster can be done only in memory, or stored on disk if RDD does not fit in memory. Serialized java objects, where only one byte array per partition is created, are more space-efficient, but more CPU-intensive to read.

The latest change in the Spark execution engine, called as Project Tungsten[4] improves memory and CPU efficiency through its explicit memory management, binary processing and better cache aware computation. We have evaluated the impact of Project Tungsten's features on ALS MF application.

2.2 Sparkbench Suite

Tailored for Apache Spark, Sparkbench suite[5] has four major categories of applications: Machine learning, SQL query, Streaming applications and Graph computation. This comprehensive suite makes extensive use of the built-in modules of Spark and exposes different bottlenecks with different application characteristics.

For the work described in this paper, we have selected a Machine learning model: ALS based MF to be evaluated and optimized. MF is used in Recommendation systems as it provides model based collaborative filtering and can be configured to use feedback from users. The memory hungry nature of this application with heavy shuffle operations makes it an important and interesting benchmark to evaluate and optimize.

2.3 Spark Cluster environment

We have created Spark environment for our experiments on IBM Power Systems S812LC[6] server machines, which comprises of POWER8 processors[7]. The high thread density of 8 per CPU core enhances the parallelism provided by Spark compute model. 10-core POWER8 3.54 GHz processor module has 512 KB of L2 cache per core, 8 MB of L3 cache per core, and a system memory of 512GB. We have deployed Hadoop file system for data storage to take advantage of the efficient distribution of data across worker nodes in a cluster.

3 Performance evaluation and optimization

3.1 MF application

We have evaluated the ALS MF application with synthetic dataset generated with help of Sparkbench testing framework, which allows for the easy reproduction of our experiments. For the experiments described in this paper, Table 1 provides details about the number of rows and columns and size of dataset. The dataset resides in the Hadoop file system once the data generation phase finishes. The Cluster environment details are mentioned in Table 2.

Data generation parameters	**Value**
Rows in data matrix	62000
Columns in data matrix	62000
Data set size	100 GB

Table 1: Parameters used for data generation in MF application

Spark parameter	**Value for MF**
Master node	1
Worker nodes	6
Executors per Node	1
Executor cores	80 / 40 /24
Executor Memory	480 GB
Shuffle Location	HDDs
Input Storage	HDFS

Table 2: Spark environment details for application evaluation

3.2 MF work-flow

The run phase of the ALS MF model training consists of a series of jobs, which are further divided into stages that perform Spark actions like 'first', 'count', 'saveasTextfile' etc, and transformations like 'map', 'join', 'cogroup' etc. We have highlighted the jobs and the associated APIs in Table 4 to better understand the actions and work flow of the MF application. We have depicted the order in which jobs of the MF application execute in Figure 1, based on Job Ids of Table 4. The reason for more than one API call in some jobs is due to the different stages within a job – and each stage calling different APIs.

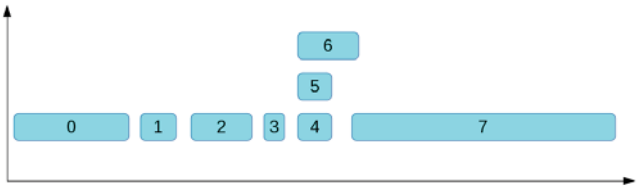


Figure 1. ALS MF jobs execution over time

Configuration	1	2	3	4	5	6	7	8	9	10	11
Spark executor cores	80	80	40	40	40	40	40	40	24	24	24
GC options	Default	Default	Default	ParallelGCthreads=40	ParallelGCthreads=40	ParallelGCthreads=40	ParallelGCthreads=40	ParallelGCthreads=40	ParallelGCthreads=24	ParallelGCthreads=24	Default
RDD compression	True	False	False	False	True	True	False	False	False	False	False
Storage level	memory_and_disk	memory_only	memory_only	memory_only	memory_and_disk_ser	memory_only_ser	memory_only	memory_only	memory_and_disk_ser	memory_and_disk_ser	memory_and_disk_ser
Partition numbers	1000	1000	1000	1000	1000	1000	800	1200	1000	1000	1000
Shuffle Manager	Sort based	Sort based	Sort based	Sort based	Sort based	Sort based	Sort based	Sort based	Sort based	Tungsten-sort	Tungsten-sort
Run-time (minutes)	40	34	26	24	20	25	26	27	21	19	18

Table 3: Various configurations tried in optimizing MF application on Spark

Job	Function	Description / API called
7	Mean at MFApp.java	AbstractJavaRDDLike.map MatrixFactorizationModel.predict JavaDoubleRDD.mean
6	Aggregate at MFModel.scala	MatrixFactorizationModel.predict MatrixFactorizationModel.countApproxDistinctUserProduct
5	First at MFModel.scala	ml.recommendation.ALS.computeFactors
4	First at MFModel.scala	ml.recommendation.ALS.computeFactors
3	Count at ALS.scala	ALS.train and ALS.intialize
2	Count at ALS.scala	ALS.train
1	Count at ALS.scala	ALS.train
0	Count at ALS.scala	ALS.train

Table 4. Description of jobs in MF application

3.3 Optimizing MF application

We have carried out several experiments to evaluate the performance impact with different Spark parameters. The Spark parameters that we have tuned for this work are listed in Table 3, in which each column is representative of an optimization iteration. We have tested configuration 1 and 2 from Table 3 for the purpose of characterizing the MF application, thus providing the maximum resources available in the cluster. For the purpose of a better understanding, we have plotted the CPU utilization over time for a node in Figure 2, and the memory footprint on a node in Figure 3. The low CPU utilization sections in the Figure 2 is representative of garbage collection (GC) pause times. We have confirmed that garbage collection happens during the low CPU activity sections by looking the history logs of stages and tasks.

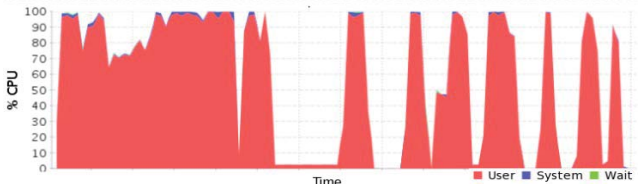


Figure 2. CPU utilization on a worker node (configuration 1 in Table 3)

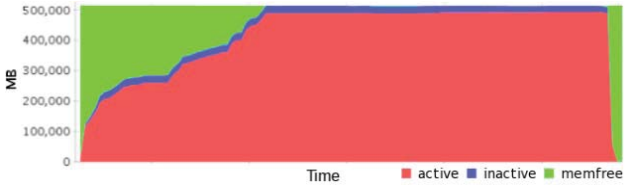


Figure 3. Memory utilization on a worker node (configuration 1 in Table 3)

In configurations 3, 4, 5 and 6 from Table 7, we selected the spark executor cores as 40 which improves the performance by providing more executor-memory per spark-executor-core and reducing contention in the last level cache with fewer spark executor cores. The difference in the above mentioned configurations is the storage level and the java garbage collection options. Performance with configuration 5 is twice that of configuration 1, which highlights the advantage by using same GC parallel threads as executor cores.

In the next set of experiments, we evaluated the performance impact by changing the number of partitions in the RDD. Configurations 7 and 8 show the partition numbers and the corresponding run times that we achieved. As the number of partitions decides the number of tasks that are spawned, careful selection of number of partitions to efficiently utilize CPU is vital to efficiently exploit the parallelism provided by the underlying compute resources. Configuration 7 and 8 did not yield any improvement over configuration 5, so we continued the optimization process using settings from Configuration 5.

In order to further reduce the last level cache contention, we configured Spark executor cores to 24 in configuration 9, 10 and 11, and it positively impacts the performance. Also, GC time is considerably reduced with 24 executor cores in configuration 9, but it still faces overhead from too many java objects created and garbage collection.

We evaluated the tungsten-sort shuffle manager from Project Tungsten in configurations 10 and 11. We disabled the java GC options in configuration 11, because tungsten-sort works directly against the binary data instead of java objects, thus alleviating the garbage collection overhead. This configuration gives us the best result with the experiments we performed for the scope of this paper. Using 24 out of 80 available threads along with tungsten-sort as the shuffle manager helps MF application to do more cache-aware computation, causes less last level cache contention and releases the pressure that comes from JVM object model.

The overall run time and memory footprint is greatly reduced as seen in Figure 5. We have highlighted the most time consuming stage in Figure 4, which is also the last stage of ALS MF application. We have compared the run time and GC time for this stage in Table 6, which shows the impact of using different values of spark executor cores and tungsten-sort on run-time and GC time.

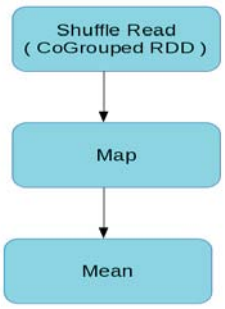


Figure 4. Directed Acyclic Graph of last stage in ALS MF

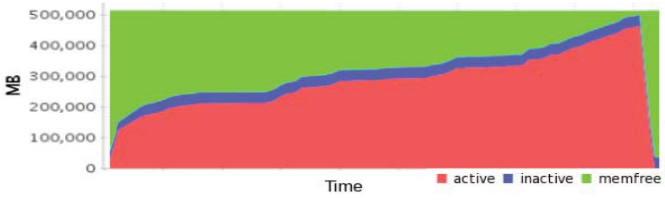


Figure 5. Memory footprint of configuration 11 from Table 3

Configuration	Run time of last stage	GC time of last stage
1	12 min	4.4 min
4	4.4 min	1.8 min
9	3.5 min	1.6 min
11	**47s**	**16s**

Table 6. Run time and GC time of Stage 68 for different configurations in Table 3

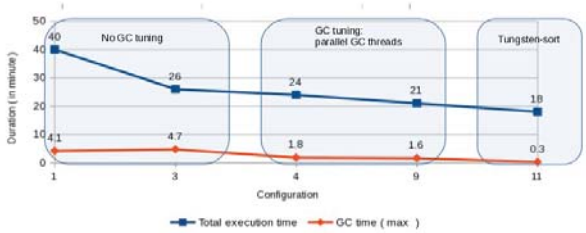


Figure 6. Run time and GC time for configurations in Table 3

4 Conclusion

The work presented in this paper explores the wide optimization space for the applications running on Spark. We use a hybrid top-bottom approach by searching the configuration space carefully after workload characterization through resource monitoring. This systematic methodology helps reduce the tuning iterations and achieving performance targets faster.

The shuffle heavy characteristic of ALS MF puts pressure on storage, network and also increases memory pressure from garbage collection due to JVM object model. Use of tungsten-sort feature alleviates these issues by doing explicit memory management and binary processing. We also observe that tungsten-sort does not bring advantage for all applications, and thus our hybrid "top-down" approach of workload characterization followed by searching configuration space becomes vital in order to apply custom optimization strategy.

Performance evaluation and optimization has resulted in a speed up of over 2x for the ALS MF application as well as other SQL and ML pipeline bases applications. The categorization of different applications and applying custom tuning for the different categories can not only improve operational efficiency, but also can be economical in the long run for an organization. We continue to evaluate more applications and improve our approach so as to reach best performance point with minimum optimization iterations for a Spark application.

References

[1] Hilbert, Martin; López, Priscila (2011). "The World's Technological Capacity to Store, Communicate, and Compute Information". Science 332 (6025): 60. doi:10.1126/science.1200970. PMID 21310967

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX NSDI, Berkeley, CA, USA, 2012.

[3] Haoming Li, Bangzheng He, Michael Lublin,Yonathan Perez (2015). "Matrix Completion via Alternating Least Square (ALS)". http://stanford.edu/~rezab/dao/notes/lec14.pdf

[4] Reynold Xin and Josh Rosen (2015). “Project Tungsten: Bringing Spark Closer to Bare Metal”. https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html

[5] MinLi, Jian Tan, Yandong Wang, Li Zhang, Valentina Salapura (2015). “SparkBench: a comprehensive benchmarking suite for in memory data analytic platform Spark”. CF '15 Proceedings of the 12th ACM International Conference on Computing Frontiers, Article No. 53

[6] IBM Power System S812LC, https://www.ibm.com /marketplace/cloud/big-data-infrastructure/us/en-us

[7] POWER8, https://en.wikipedia.org/wiki/POWER8

A Computational Reordered Algorithm with Overlapping of Communication and Computation for the All Pairs Shortest Path Problem in Distributed Memory Environments

Eduardo A. Colmenares[1], **Per Andersen**[2], **Yu Zhuang**[3]
[1]Department of Computer Science, Midwestern State University, Wichita Falls, Texas, USA
[2]High Performance Computing Center (HPCC), Texas Tech University, Lubbock, Texas, USA
[3]Department of Computer Science, Texas Tech University, Lubbock, Texas, USA

Abstract - *A very well-known and relevant problem in science is the All Pairs Shortest Path problem (APSP). The APSP has a considerable number of applications ranging from network traffic to circuit design. Through the years different versions of Floyd's algorithm have been implemented to solve the APSP, some of these versions have been adapted to contemporary architectures in order to better exploit hardware and software resources in the quest for additional benefits in total execution time. In this paper we follow the recommendations of the Department of Energy (DOE), and we present a study for a proposed parallel variant of Floyd's algorithm in a distributed shared memory environment (cluster), while trying to maximize the opportunities for communication hiding through the use of overlapping of communication and computation. The proposed algorithm relies in two key factors to attempt better performance, computation reordering and overlapping. In this research we also show why a cluster characterization is important, and how it can help to explain unexpected results.*

Keywords: computation reordering, all pairs shortest paths, overlapping, cluster.

1 Introduction

In today's world High Performance Computing (HPC) researches have access to a considerable number of architectures capable of delivering outstanding performance, if exploited properly. It is important to remember, and understand, that not all architectures provide the same hardware and software support to strategies that can help to improve performance. For example, the creation of opportunities where communication hiding can be applied.

In this research, we (1) follow the DOE recommendations [7]. The result, is a computational reordered version of Floyd's that solves the APSP problem, and attempts to achieve better performance than the traditional two dimensional approach of Floyd's. In order to pursue this goal, the proposed algorithm relies in the support for overlapping of communication and computation. (2) We show why a characterization of the testing environment is not only important, but also relevant, as well as required, in order to explain and justify results.

2 Floyd-Warshall Algorithm

Floyd's algorithm is a widely known and studied algorithm. Its purpose is to find the shortest path between all pairs of nodes in a graph [1, 10]. Its applications include but are not limited to electrical circuit design, network routing, maps, and subroutine for other algorithms, for example transitive closure [12].

A condensed definition of Floyd's algorithm is as follows: A graph G can be represented as G = (V, E); where the cost of going from vertex i to vertex j is always positive and denoted as E[i,j]. V={1,….,n} are the vertices of G. Floyd's algorithm computes the following matrices.

$$D_{i,j}^{(0)} = \begin{Bmatrix} 0 \rightarrow i = j; (diagonal) \\ E[i,j] \rightarrow i \neq j \end{Bmatrix}$$

$$d_{i,j}^{(k)} = \min\left\{ d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)} \right\}$$

Where $d_{i,j}^{(k)}$ means the shortest path from *i* to *j* that does not goes through any vertex bigger than k. Solving the $D_{i,j}^{(n)}$ matrix is the main objective of Floyd's algorithm, $D^{(0)}$ is also known as the adjacency matrix. [1, 5, 9, 12].

2.1 Sequential Approach

According to the work presented in [1, 10], it is possible to represent the sequential version of Floyd's as shown in figure 1.

2.2 Traditional 2D Parallel

There are a considerable number of parallel variants for Floyd's algorithm, some of which can be found in [1, 9]. This version is also known as the checkerboard approach, because the processes are distributed in a squared grid fashion

way [1, 9]. It is relevant to highlight three important characteristics associated with this approach: (1) the grid dimensions must have an exact square root value, this is done to avoid load balancing issues, (2) the distribution of the processes corresponds to a logical but not physical distribution, and (3) this algorithm does not make an attempt for overlapping of communication and computation to improve performance. For the remainder of this paper, we will refer to this approach as the Trad-2D approach.

```
1.   procedure FLOYD_ALL_PAIRS_SP(E)
2.   begin
3.    D(0) = E;
4.     for k := 1 to n do
5.      for i := 1 to n do
6.       for j := 1 to n do
7.         d(k)i,j = min{d(k-1)i,j, d(k-1)i,k + d(k-1)k,j};

                (EQUATION 1)

8.   end FLOYD_ALL_PAIRS_SP
```

Figure 1: Sequential Floyd's Algorithm

```
1.  procedure FLOYD_2DBLOCK(D(0))
2.  begin
3.   for k := 1 to n do
4.    begin

5.     each process Pi,j that has a segment of
       the kth row of D(k-1); broadcasts it to the
       P*,j processes;

6.     each process Pi,j that has a segment of
       the kth column of D(k-1); broadcasts it to
       the Pi,* processes;

7.     each process waits to receive the needed
       segments;

8.     each process Pi,j computes its part of the
       D(k) matrix;
9.  end
10. end FLOYD_2DBLOCK
```

Figure 2: Algorithm for Floyd's Parallel Formulation Using 2D-block Mapping.

Figure 2 [9] presents the pseudocode for this version, which can be summarized as follows: assuming a number of processors equal to *p*, and a total number of vertices *n*, it is possible to infer that the squared logical topology will consist of $\sqrt{p}$ rows, $\sqrt{p}$ columns; and *p* blocks, with one block assigned to each process. The total number of elements per block is the equal to $\left(\frac{n}{\sqrt{p}}\right)^2$, distributed in $\frac{n}{\sqrt{p}}$ rows and $\frac{n}{\sqrt{p}}$ columns [9]. It is also important to facilitate the reference of a particular process in the conceptual topology; this can be easily done by using a $P_{i,j}$ nomenclature where the pair (i, j) make reference to row and column of interest in the logical grid. Based on this nomenclature, $P_{i,*}$ references all processes in the *i*th row, and $P_{*,j}$ all processes in the *j*th column. Step 8, in figure 2 corresponds to a synchronization step [1, 9], which computes the value of $d_{i,j}^{(k)}$ by using equation 1.

In some cases, the distribution of matrix $D^{(k)}$, among *p* process, will create situations in which critical and necessary segments of information won't be located in the local process. Instead, these relevant and needed values will be located in other processes, some of which are located along the same row as our local process, and some others along the same column. In order to complete the computation of $d_{i,j}^{(k)}$ for the current value of k, local process $P_{i,j}$ must receive those relevant segments of the k[th] row and k[th] column of $D^{(k-1)}$ matrix [1, 9].

During the k[th] iteration, each one of the $P_{i,j}$ processes owning a segment of the k[th] column will broadcast that segment to all other processors located in the same row. Similarly, each one of the $P_{i,j}$ processes owning a segment of the k[th] row will broadcast that segment to all other processors located in its column. This broadcasting protocol is further explained in [1, 9]. It is significant to remember that each process is assigned a square sub-block of matrix $D^{(k)}$, which consist of $\frac{n}{\sqrt{p}}$ row and $\frac{n}{\sqrt{p}}$ columns; as a consequence, it is possible to conclude that when a process broadcasts a relevant segment of the k[th] row, or k[th] column, $\frac{n}{\sqrt{p}}$ elements are broadcasted, and the number of recipients processes is $\sqrt{p}-1$ [1, 4, 9].

2.3 Proposed Algorithm (PAPSP)

This algorithm makes and attempt for extensive use of overlapping of communication and computation, with the sole purpose of achieving better performance than the Trad-2D algorithm introduced section 2.2. This approach is not only based in the checkerboard version, but is also derived based on two critical observations.

The first observation states that if a process owns D(k) samples, and these samples are needed by any other processes during the next iteration of k, then, the process that owns the samples does not have to compute those D(k) samples right after computing any other samples of D(k); according to [4], this is possible because no order (data dependency) is imposed by Floyd's algorithm on the computation of different samples of D(k) for each possible value of k.

The second observation is as follows. A processor that owns D(k) samples, which will be needed by any other processes during the next iteration of k, does not have to

send those D(k) samples immediately before other processes need them for computing D(k+1) samples [4].

The two previous observations are simple but critical, and are the base of the strategy to follow during the k^{th} iteration.

During the k^{th} iteration, a receiving process computes its D(k) samples, while D(k) samples required by it for the $(k+1)^{th}$ iteration, are traveling from the source process (sending processes) to this receiving process; it is at this point when overlapping of communication and computation for this algorithm takes place. According to [4], the corresponding algorithm for this approach is as shown in figure 3.

For the remainder of this paper, we will refer to this algorithm as the proposed algorithm for All Pairs Shortest Path Problem (PAPSP).

2.3.1 T0-Test

T0-Test corresponds to the first variant of the computation-communication overlapped parallel Floyd algorithm (PAPSP). In this variant the functions used to collect and send messages are non-blocking instructions, in this case MPI_Irecv and MPI_Isend correspondingly.

2.3.2 T1-Test

T1-Test is a variant of T0-Test. In this variant all MPI_Isend have been replaced by MPI_Send.

2.3.3 T2-Test

T2-Test is the second variant of T0-test. In this variant, all MPI_Irecv have been replaced by MPI_Recv. As a second change, the MPI_Recv's have been moved to a more strategic position. This new position is strategic for several reasons. First, the process will receive late; second, as the process receives late there is no longer a valid reason to use some buffers that were required in T0 and T1, this not only reduces the amount of memory to be used, but also avoids the need of a loop to update vbuf and hbuf [4]. These changes are easily identifiable in figure 4.

2.3.4 T3-test

T3-Test is the third variant of T0-test. In this variant, all MPI_Isend have been replaced by MPI_Send, and the MPI_Irecv's have been moved to a strategic position, same reasons as 2.3.2. This test combines blocking and non-blocking instructions, all the differences between T0 and T3 are shown in figures 5, 6, 7.

Each Process initializes the submatrix of $D^{(0)}$ with the submatrix of A.

Processor $P_{i,j}$ $(j=1,\ldots,p^{1/2})$ broadcast the segment of the 1^{st} row of $D^{(0)}$ to processes $P_{i',j}$ for all $i' \neq 1$; and processes $P_{i',j}$ for all $i' \neq 1$ receive the segment of the 1^{st} row of $D^{(0)}$.

Processor $P_{i,1}$ $(i=1,\ldots,p^{1/2})$ broadcast the segment of the 1^{st} column of $D^{(0)}$ to processes $P_{i,j'}$ for all $j' \neq 1$; and processes $P_{i,j'}$ for all $j' \neq 1$ receive the segment of the 1^{st} column of $D^{(0)}$.

For k starting from 1 through (n-1)

Step 1. Each Processor $P_{i,j}$ owning a segment of $(k+1)^{th}$ row of $D^{(k)}$ computes entries on the $(k+1)^{th}$ row of $D^{(k)}$ as indicated by equation 1 of figure 1, and broadcast these just computed entries to processors $P_{i',j}$ for all $i' \neq i$.

Step 2. Each Processor $P_{i,j}$ owning a segment of $(k+1)^{th}$ column of $D^{(k)}$ computes entries on the $(k+1)^{th}$ column of $D^{(k)}$ as indicated by equation 1 of figure 1, and broadcast these just computed entries to processors $P_{i,j'}$ for all $j' \neq j$.

Step 3. Each Processor $P_{i',j}$ owning no segment of $(k+1)^{th}$ row of $D^{(k)}$ compute uncomputed$^{(*)}$ entries of $D^{(k)}$ using equation 1 of Figure 1; and then receive entries on the $(k+1)^{th}$ row of $D^{(k)}$ broadcasted from the processor owning them.

Step 4. Each Processor $P_{i,j'}$ owning no segment of $(k+1)^{th}$ column of $D^{(k)}$ compute uncomputed$^{(*)}$ entries of $D^{(k)}$ using equation 1 of Figure 1; and then receive entries on the $(k+1)^{th}$ column of $D^{(k)}$ broadcasted from the processor owning them.

Step 5. The processor owning a segment of $(k+1)^{th}$ row and a segment of $(k+1)^{th}$ column of $D^{(k)}$ compute uncomputed entries using equation 1 of Figure 1.

(*) Note: A processor owning no segment of the $(k+1)^{th}$ row of $D^{(k)}$ might own a segment of $(k+1)^{th}$

Figure 3: The Computation-Communication Overlapped Parallel Floyd Algorithm (PAPSP).

Figure 4: Differences between T0 and T2 Tests.

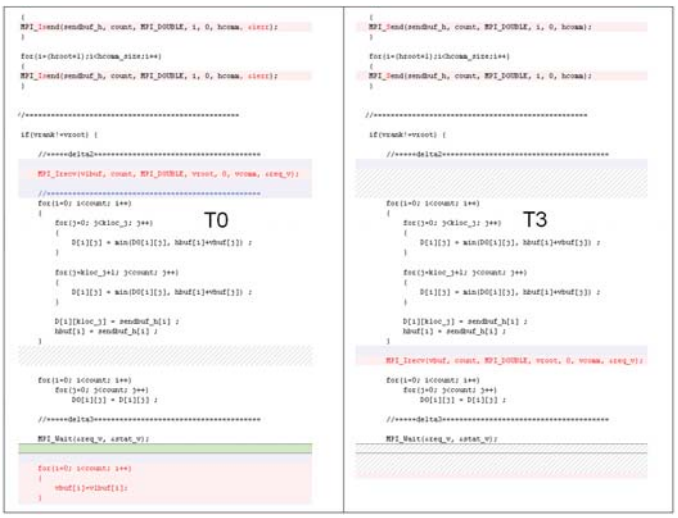


Figure 5: First Difference between T0 and T3.

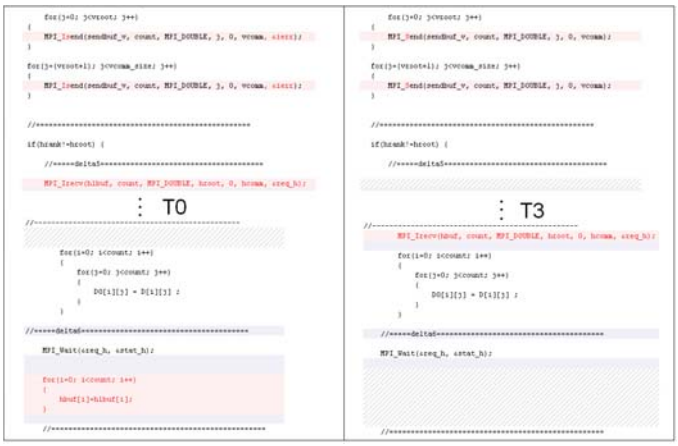


Figure 6: Second Difference between T0 and T3.

Figure 7: Third Difference between T0 and T3.

3 Testing Environments

Initially, we used two different clusters as our testing environments. The first cluster corresponds to a multi-user cluster (MUC) which provides scientific computational capabilities to a research community. For the remainder of this paper will refer to it as MUC. 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 OUC [2, 3]. Table 1 summarizes the major hardware differences among the nodes of OUC; for further details, please see [2, 3].

TABLE I. HARDWARE DIFFERENCES AMONG NODES FOR OUC

Node	Memory (MB)	Hard Disk
0, 3	511.46	40020 MB-T340016A, ATA
1	1023.4	40020 MB-T340016A, ATA
2,4, 5	1023.4	20547 MB-MAXTOR 6L020J1, ATA
6	1023.4	40020 MB-WDC WD400BB-75DEA0, ATA
7, 8	1023.4	40027 MB-MAXTOR 6L040J2, ATA

4 Clusters Characterization

The proposed algorithm, introduced in section 2.3, makes intensive use of overlapping of communication and computation in an attempt to achieve better performance than the traditional parallel 2D approach presented in section 2.2.

It is possible that in some cases, under the same testing conditions, the differences in execution time between these two approaches will be really close; as consequence, both can be considered time sensitive applications. When dealing with time sensitive applications, it is important to identify potential sources of error that may affect our results. In our case, those that may negatively impact communications, overlapping capabilities, as well as the behavior or non-blocking instructions. For such reasons, a characterization of both clusters was conducted. The corresponding methodologies, tests, and results are presented in full detail in [2, 3].

5 Testing Results

5.1 No optimization Flags vs Optimization Flags

In order to evaluate the potential benefits of optimization flags, the following approaches: Trad-2D, PAPSP with Broadcast, and PAPSP-T0, were executed a total of 141 times in OCU, under the following testing conditions: 9 processes and five different problem sizes (180, 720, 1440, 2520 vertices). The commands for these tests were as follows:

No optimization flags:
mpicc program_name.c –o executable_name –lm.

With Optimization flags:
mpicc –g0 –O3 –lm program_name.c

The results are presented in tables 2, 3, and 4, not optimized and optimized readings corresponding to the average of all 141 samples.

TABLE II. STATISTICS FOR TRAD-2D | 141 SAMPLES| 9P

Number of Vertices	TRAD 2D - 141 samples		
	NO OPTIMIZED (secs)	OPTIMIZED (secs)	% Improvement
180	0.0789	0.0732	7.2573
720	2.2573	2.0903	7.3991
1440	15.9298	15.1823	4.6926
2520	80.9305	78.1582	3.4256

TABLE III. STATISTICS FOR PAPSP-BCAST | 141 SAMPLES | 9P

Number of Vertices	BCAST - 141 samples		
	NO OPTIMIZED (secs)	OPTIMIZED (secs)	% Improvement
180	0.0806	0.0734	8.8977
720	3.1085	2.9021	6.6393
1440	22.1504	21.0979	4.7515
2520	114.6667	109.2057	4.7625

TABLE IV. STATISTICS FOR PAPSP-T0 VARIANT | 141 SAMPLES | 9P

Number of Vertices	IRECV "T0 TEST" - 141 samples		
	NO OPTIMIZED (secs)	OPTIMIZED (secs)	% Improvement
180	0.0905	0.0824	9.0200
720	4.1089	3.8315	6.7523
1440	28.5915	28.0922	1.7463
2520	184.9787	183.3901	0.8588

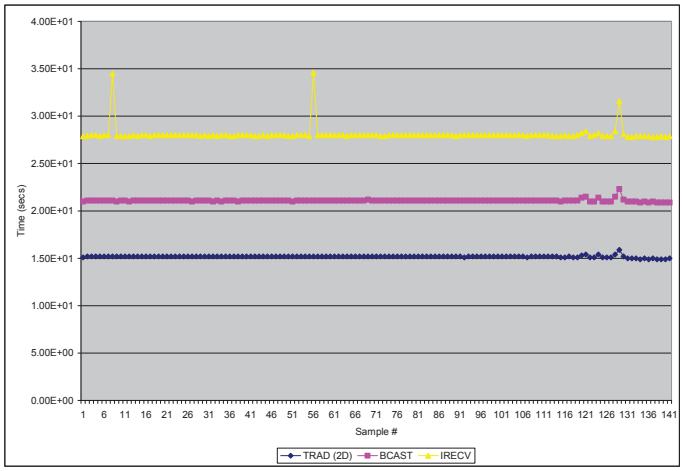


Figure 9: Trad-2D vs PAPSP Bcast, and PAPSP-T0 | 1440 Vertices and 9 Processes in OCU.

5.2 Test Results for PASPS T1, T2, T3, T3.1 variants

A visual inspection of figure 9 shows that T0-Test does not perform as well as the Trad-2D and the Bcast version of our PAPSP algorithm.

This section presents the testing results for the remaining four variants of the proposed algorithm (PAPSP). Test were conducted for four different problem sizes, 180, 720, 1440, and 2520 vertices, and a total 9 processes. Each one of the PAPSP variants was executed 141 times, resulting in a total of 141 samples per problem size. Tables 5 through 8 show the statistics for each one of the testing problem sizes.

TABLE V. STATISTICS FOR PAPSP-T1, T2, T3, T3.1 VARIANTS | 180 VERTICES

180 V - 9P				
Test	Average (secs)	Stdev (secs)	min (secs)	max (secs)
T1	7.04E-02	0.001458	6.72E-02	7.71E-02
T2	7.91E-02	0.020923	7.40E-02	3.25E-01
T3	6.95E-02	0.001338	6.56E-02	7.42E-02
T3.1	7.12E-02	0.005906	6.79E-02	1.39E-01

TABLE VI. STATISTICS FOR PAPSP-T1, T2, T3, T3.1 VARIANTS | 720 VERTICES

720 V - 9P				
Test	Average (secs)	Stdev (secs)	min (secs)	max (secs)
T1	2.91E+00	0.006848	2.90E+00	2.95E+00
T2	3.84E+00	0.137558	3.80E+00	4.65E+00
T3	2.91E+00	0.006582	2.90E+00	2.93E+00
T3.1	2.89E+00	0.008683	2.87E+00	2.93E+00

TABLE VII. STATISTICS FOR PAPSP-T1, T2, T3, T3.1 VARIANTS | 1440 VERTICES

1440 V - 9P				
Test	Average (secs)	Stdev (secs)	min (secs)	max (secs)
T1	2.11E+01	0.169734	2.10E+01	2.24E+01
T2	2.82E+01	1.114721	2.79E+01	3.48E+01
T3	2.11E+01	0.427721	2.10E+01	2.61E+01
T3.1	2.08E+01	0.035857	2.07E+01	2.09E+01

TABLE VIII. STATISTICS FOR PAPSP-T1, T2, T3, T3.1 VARIANTS | 2520 VERTICES

2520 V - 9P				
Test	Average (secs)	Stdev (secs)	min (secs)	max (secs)
T1	1.10E+02	0.959884	1.08E+02	1.17E+02
T2	1.82E+02	2.141736	1.80E+02	1.96E+02
T3	1.10E+02	0.914479	1.09E+02	1.15E+02
T3.1	1.07E+02	0.202567	1.07E+02	1.08E+02

Figure 10 shows that T3.1 is faster than T0, T1, T2 and T3 for those testing conditions. A close examination of figure 11, which corresponds to the testing conditions of 2520 vertices and 9 processes, allows us to conclude that although

T3.1 is faster than the PAPSP-Bcast version, it is not as fast as the traditional approach "Trad-2D"; similar behavior was observed for 720 and 1440 vertices and 9 processes.

It is also possible to deduct that T3.1 provides an improvement over T0 which uses only non-blocking instructions; but the traditional implementation Trad-2D is still better.

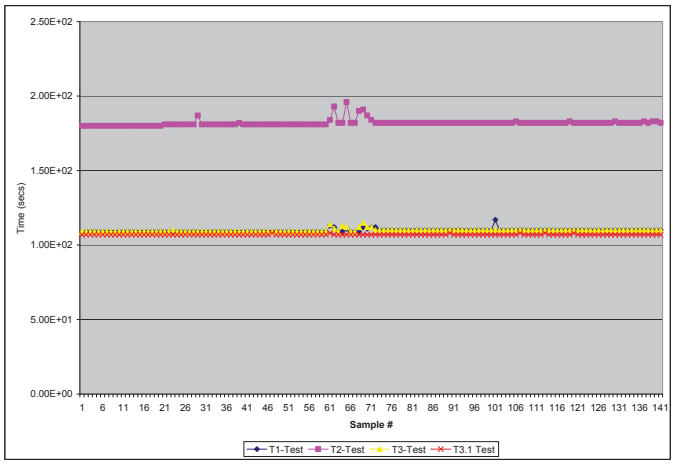


Figure 10: Comparison T1, T2, T3 T3.1 | 2520 Vertices | 9 Processes in OCU

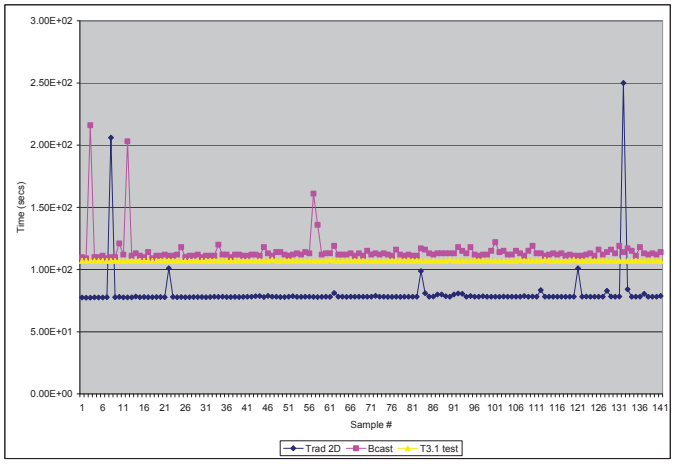


Figure 11: Comparison Trad-2D, PAPSP-Bcast and T3.1|2520 Vertices | 9 Processes in OCU

5.3 Cluster Characterization Results

Interesting findings were achieved during the cluster characterization. These findings are fully explained in [2, 3]. In this section, we want to emphasize the importance of the results associated with the support for overlapping, and other potential sources of error.

According to [2], the public queues of the MUC cluster are susceptible to pulsing behaviors, probably associated with scheduling layers of this distributed memory multi-user environment. The pulsing behavior shown in [2] can negatively impact time sensitive readings, introducing an undesired margin of error. [3] Shows that the OUC cluster presents a more steady behavior in terms of communication, making it a better candidate for our tests.

The second finding comes from [3], where the message size sweep test is introduced, and explained. This test is designed to study how the size of the message affects the overlapping capabilities in different clusters. Table 9 is taken from [3], and shows the results of the message size sweep for two different clusters, MSC a Multi-User Solaris Cluster, and OUC a One User Cluster. According to [3], by comparing both halves of table 9 is possible to observe that the size of the message plays an important role in the achievement of overlapping synchronization and computation.

It is clear that the size of the buffer used by each system to send and receive message has a different limit. If the size of the message to be sent is smaller than the suggested limit then the system is able to support overlap of synchronization with computation, otherwise the support of such overlap will not be available [4, 8, 6]. The first half of table 9 shows that for MSC, the limit appears to be between 60KB and 70KB. The second half of table 6 shows a limit between 10KB and 20KB for OUC.

TABLE IX. MESSAGE SIZE SWEEP RESULTS ON (MSC) AND (OUC)

SOLARIS-CLUSTER - "MPICH1"			OUC - "MPICH2"	
Message Size (KB)	TP0 (secs)	TP1 (secs)	TP0 (secs)	TP1 (secs)
10	61.0	1.08E-03	3.8	1.24E-03
20	61.0	1.54E-03	3.8	3.8
30	61.0	1.92E-03	3.8	3.9
40	61.0	2.69E-03	3.8	3.8
50	97.6	3.62E-03	3.8	3.8
60	61.0	2.82E-03	3.8	3.8
70	61.0	61.0	3.8	3.8
80	61.0	61.1	3.8	3.8

5.4 Retesting Trad-2D vs T3.1

Section 5.3 shows that the size of the buffer plays and important role in our support for overlapping. With that in mind, retesting under those conditions was necessary in order to evaluate the performance and potential benefits of our proposed algorithm.

Figure 12 shows 141 samples, for both versions of Floyd's, the Trad-2D approach and variant of our PAPSP algorithm "T3.1". Each one of the samples corresponds to a full execution for the testing conditions of 180 vertices and 9 participant processes.

A visual examination of figure 12 will show that with the correct support for overlapping, our proposed algorithm will perform better than the traditional two dimensional approach "Trad-2D".

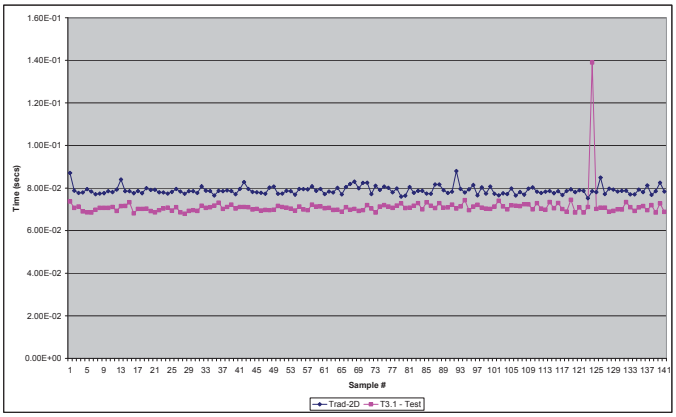


Figure 12: Comparison Trad-2D and T3.1| 180 Vertices | 9 Processes

6 Conclusions

Figure 9 shows that although optimization flags do not improve the overall performance with respect to overlapping of communications and computation. It is clear that the use of optimization flags improves the overall performance between the same implementation, especially when the size of the problem to be solved is small; this improvement decreases when the problem size becomes larger [3].

Non-blocking instructions in MPICH-2 and non-blocking instructions in MPICH-1 provide support to overlapping of computation and communication; however this support is limited, and is directly related to the size of the message and/or the cumulative size of all messages to be sent; if such size is bigger than the system's socket buffer, then non-blocking instructions no longer will behave as expected, and the support for overlap will be negatively impacted [2, 3, 4, 6].

The proposed algorithm, specifically T3.1 will perform better than its traditional two dimensional counterpart "Trad-2D" as long as size of the message to be transmitted happen to be smaller than the threshold of interest, which is dictated by the system's socket buffer.

7 References

[1] A. Grama et al., Introduction to Parallel Computing, Pearson Addison Wesley, 2003.

[2] Colmenares E., Andersen P, "A Data Communication Reliability and Trustability Study for Cluster Computing," in the International Conference on Parallel and Distributed Processing Techniques and Applications, July 27-30, 2015.

[3] Colmenares E., Andersen P, Wei B. "An Overlap study for Cluster Computing". CSCI'15, the 2015 International Conference on Computational Science and Computational Intelligence, pp 626, 631

[4] Colmenares E, Andersen P, Zhuang Y., "Overlapping Communication and Computation with MPI-for Floyd's Algorithm", MSCS Thesis, Texas Tech University, 2008.

[5] Diament, B. and Ferencz, A. (1999, May). Comparison of Parallel APSP Algorithms. Available: http://citeseer.ist.psu.edu/334881.html

[6] Hoefler, T.; Lumsdaine, Andrew; Rehm, Wolfgang, "Implementation and performance analysis of non-blocking collective operations for MPI," Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on vol., no., pp.1,10, 10-16 Nov. 2007, doi: 10.1145/1362622.1362692

[7] J. Ang, K. Evans, A. Geist, M. Heroux, P. Hovland, O. Marques, L. McInnes, E. Ng, and S. Wild, Report on the workshop on extreme-scale solvers: Transitions to future architectures. Office of Advanced Scientific Computing Research, U.S. Department of Energy, 2012. Washington, DC, March 8-9, 2012

[8] J. White III and S. Bova. (1999). Where is the Overlap? an Analysis of Popular MPI Implementations. Available: http://citeseer.ist.psu.edu/297838.html

[9] Kumar, V. and Singh, V. . "Scalability of Parallel Algorithms for the All-Pair Shortest Path Problem". International Conference on Parallel Processing. Mar. 21, 1991.

[10] SARC European Project, "Parallel Programming Models for Heterogeneous Multicore Architectures." IEEE Micro 30.5 (2010): 42-53

[11] Saif, T. , and M. Parashar. "Understanding The Behavior and Performance of Non-blocking Communications in MPI". 10th International Euro-Par Conference, Italy, 2004, pp.173-182.

[12] Vinjamuri, S.; Prasanna, V.K., "Transitive closure on the cell broadband engine: A study on self-scheduling in a multicore processor," Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on , vol., no., pp.1,11, 23-29 May 2009, doi: 10.1109/IPDPS.2009.5161072

Design and Performance of a Low Cost Cluster using ARM-based Platform

Felipe dos Anjos Lima[1], Edward David Moreno[1], Wanderson Roger Azevedo Dias[2]
[1]Department of Computer Science (DCOMP) – Federal University of Sergipe (UFS)
[2]Coordination of Informatics (COINF) – Federal Institute of Sergipe (IFS)
Aracaju, SE, Brazil
{felipes1474, edwdavid, wradias}@gmail.com

***Abstract*—This article presents the results obtained during the simulations that measured the performance of a low-cost cluster with ARM processors. For the simulations, we consider the impact caused by parallel applications responsible for big chunks of calculations (HPL benchmark, matrix multiplication and scalar product).Thus, after the simulations, we could observe that the cluster performance increased considerably compared to sequential solutions, when the measured amount of processed data also increased. For matrix multiplication, the gain came to 66% when compared to the sequential solution.**

Keywords—cluster; raspberry pi; performance; parallelism

I. INTRODUCTION

Along with technological development, computer processing power has greatly increased over the last years. Therefore, the quantity of information has also increased, hence the search for faster systems and processors which bear the new demands still a big priority.

Nowadays, the task of maintaining the computing power of the processors continuously rising, according to Moore`s Law [1] has been a great challenge for computer scientists and engineers. That main reasons for that are the difficulties on developing technology which reduces transistor size, as well as the heat these components dispute [4, 5].

In order to solve this problem, processor designers started to considerate the creation of parallel systems. In other words, instead of trying to increase processing power from a single core, they came to considerate the creation of processors with several cores inside one unique integrated circuit.

By taking advantage of parallelism with processors compound of several cores, it was possible to increase the process power to higher rates than the ones we have managed to reach when a single core was used.

By associating two or more computers connected to a LAN (*Local Area Network*), it is also possible to create a cluster, which can be seen as a single system with multiple processors [21].

In order to take advantage from the rescuers available on systems that support parallel process, there are several APIs and libraries that help the development of parallel programs. As an example, we may cite the libraries MPI (*Message Passing Interface*) [25] and OpenMP (*Open Multi-Processing*) [2]. Thereby, all management of shared memory that was needed for maintaining consistence on the device's computing could be done safely.

In this paper, we analyze the performance of a cluster of low cost embedded processors from ARM and Raspberry Pi platform. After the cluster assembly and configuration simulations conducted to verify the same behavior. For this, we use mathematical applications that performed the calculation of the scalar product and matrix multiplication. Moreover, with the help of the HPL benchmark can extract performance metrics such as runtime and GFLOPS. With MPI framework was possible to parallelize and share the load of data between cluster nodes.

The paper is organized into six sections, Section 2 presents the simulation environment the analyzed algorithms; Section 3 is devoted to analysis of the related works; Section 4 presents the architecture of the cluster; Section 5 presents the results during simulations, and finally, Section 6 presents the conclusions and ideas for future work.

II. THEORETICAL GROUNDING

Actually, a the majority of modern computer systems is already using some resource that makes them more powerful and capable of handling higher data processing demands [23]. The capacity of processing data in parallel is one of them, which allows a computer system to run various tasks at the same time. Therefore, it's important to remember that in order to run several tasks at the same time; a system must have the disposition of many processors or one or more processing cores inside the same chip. The following section presents the main concepts related to parallel computer system.

A. Concurrent Systems

According to [22], every concurrent program has multiple control threads (see Figure 1). Therefore, this does not guarantee them to be run in parallel. In case the system has only one core available while having several tasks to be run, multiple control threads alter the usage of the processor for a specific time interval.

Though there is no parallel data processing, single-core systems may increase their performance by the usage of threads [23].

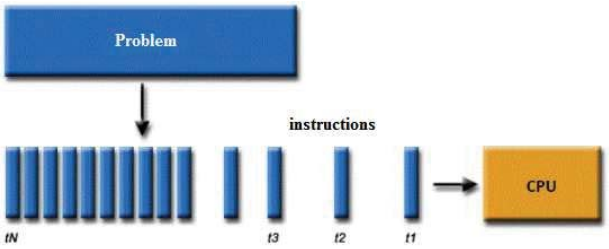


Fig. 1. Multiple threads system [22].

For example, it is possible to cite multitask operational systems, which have the objective of maximizing processor usage. Thus, every time a thread is in a stand by state, the operational system then begins to run another thread so that the processor is never lazy [22]. This mechanism is known as thread scheduling.

B. Multicore Systems

According to Pacheco [10], transistor size decreases as processing power increases. Therefore, the heat dissipated by these circuits also increases, which is problem on system's performance. The best solution so far was to assemble several processing units in a single circuit. These circuits are known as multicore systems [21].

Even though the word multicore is often used to designate systems with several processing cores in a single chip, it is also used to designate systems in that the processors stay on different chips [23]. Figure 2 presents the scheme of a processor with more than one core.

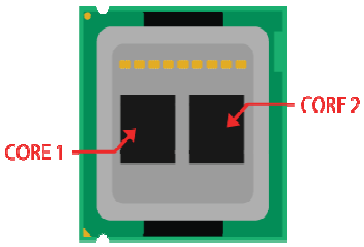


Fig. 2. Multicore processor [23].

In order to operate correctly, multicore systems need to efficiently manage memory usage, so that the consistency of the data that is processed by several cores can be maintained.

In shared-memory systems with multiple processors the connection of the memory to the processors can be done in two ways [10]: UMA (*Uniform Memory Access*) and NUMA (*Nouniform Memory Access*).

On the UMA model of connection the processors are directly connected to the main memory. In this sense, the access time to the addresses from the memory is the same for every processor in the system [21]. Figure 3 presents the operation sketch of the UMA model of connection.

On NUMA model every processor accesses a single memory block from the main memory [21]. In case a processor desires to access a memory block from another processor, it will be necessary to utilise an additional hardware because this access cannot be done directly. Figure 4 presents the operation sketch from NUMA model of connection.

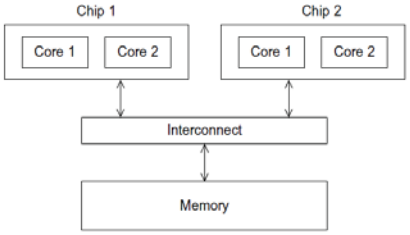


Fig. 3. UMA model of connection [10].

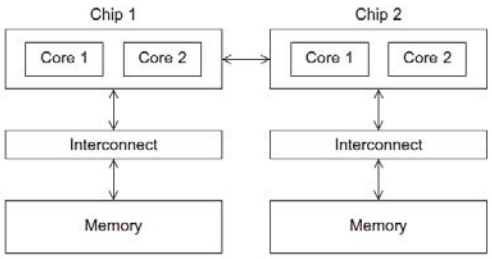


Fig. 4. NUMA model of connection [10].

C. Cluster

A cluster is a collection of two or more computers designed to run a single task [21]. Normally, cluster nodes are connected via a local area network that must allow an efficient connection between the nodes. Figure 5 depicts the operation sketch of a cluster.

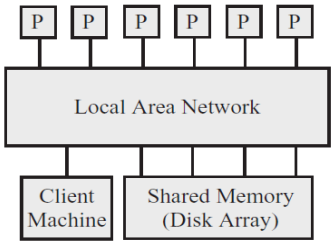


Fig. 5. Multiple node cluster [21].

In order to maintain consistency during the task execution, the cluster uses a NUMA *shared-memory* model because each processor has access to its own memory [10]. Some authors call as distributed shared-memory system, a system that uses NUMA shared memory model.

D. Parallel Algorithms

To make it possible to benefit from the resources available on parallel architectures, it is necessary that the algorithms were prepared to operate in this type of architecture, in order to improve its runtime.

Nowadays, there are many types of algorithms which do not recognize the resources available on parallel hardware architectures [10]. Systems that process images, climate data, scientific computing, energetic researches that manipulate

substantial data quantities, may have their performance increased if they did benefit from parallel hardware architectures.

According to [23], in order to correctly execute a parallel algorithm in a parallel system, it should be able to accomplish three basic tasks:

- *Mapping*: the distribution of tasks to the different processors.
- *Sharing*: the sharing of the task's execution according to data dependency.
- *Identification*: the identification of the data that travels through processors.

E. Parallel Algorithms

There are many types of parallelism that are involved in a computational system. To make an algorithm able to execute in a parallel way, it is necessary the hardware to provide the needed support. Next, the main types of parallelism are presented [21].

- *Bit-Level Parallelism*: the bits of an instruction are processed in parallel.
- *Instruction-Level Parallelism*: through a resource known as pipeline, processors are able to process instruction in parallel because they posses multiple functional unities.
- *Data-Level Parallelism*: the data are stored and can be accessed after having its results combined.
- *Task-Level Parallelism*: Uses a separate control flow for each task that is to be executed independently.

F. MPI Library

MPI (*Message Passing Interface*) is a "Message-Passing" library developed to be a pattern, initially on distributed-memory environments, for "Message-Passing" and for Parallel Computing [21]. All parallelism is explicit, in other words, the programmer is responsible to identify the parallelism and to implement the algorithm using MPI constructions.

In this model of parallel computing, the same source code is run on each processor. Thus the variables declared are relative to each processor. The MPI is responsible for managing memory between the cores of the system [21, 13]. Figure 6 shows the memory model used on clusters.

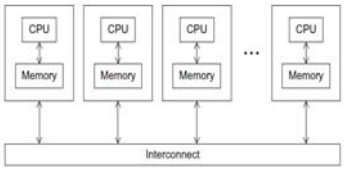


Fig. 6. Distributed-memory system [10].

The processes that are being executed by cluster nodes communicate to each other by message sending. For that, the MPI provides a set of functions described as follows:

- *MPI_Init*: responsible for initial configurations on the system. It is the very first function to be executed by the system.
- *MPI_Finalize*: frees every resource utilised by the MPI. It is the last function to be executed by the system.
- *MPI_Send*: loads the information exchanged between the processes.
- *MPI_Recv*: it is used to receive messages that were sent by other processes.

All function presented above can be called by C, C++ and FORTRAN written software.

III. Relate Work

The authors in [14] present an analysis of the impact thread usage may cause on applications. The simulations were performed on different operational systems (Windows and Linux) and on different architectures (32 and 64 bits), in order to verify the impact these alterations could cause on application performance.

To model the concurrent applications between thread manipulations, the author utilised the libraries *Pthreads* and *WinAPI*. The library Pthreads defines a default programming interface on UNIX systems [9]. On the other hand, WinAPI specifies a set of interfaces for developing desktop applications and also server applications on Windows [8].

To measure performance on the libraries, the authors worked with four problems: calculating the value of π using Leibniz method, integral calculating by rectangle rule, MIPS (*Millions of Integer Operations Per Second*) and MFLOPS (*Millions of Floating Point Operations Per Second*) calculations. For each problem, 33 executions were performed, with the number of threads varying from 1 to 4. The results showed that the increase on the number of threads provoked the enlargement of MPIS and MFLOPS rates. For integral and π value calculating, the increase on thread numbers caused a rising of speedup.

In [17], the author verified the impact the implementation of parallel programs using OpenMP caused on application performance. To perform the tests, the following applications were used: *JPEG*, *Huffman Codec*, *Radixsort*, *AES Encryption*, traveling *Salesman*, *Block Matching*, *Discrete Fourier Transform*. The simulations were performed on an ARM11 MPCore platform.

Simulation results showed that an increasing number of threads implies on a considerable rise of process power and efficiency. In addition, the increasing number of threads caused a growth of energy consumption. Figure 7 shows the impact parallelization with OpenMP has caused.

Simon *et al* [15] presents a low-cost cluster implementation compounded by sixty-four Raspberry Pi chips with ARM processors. Initially, the authors verified the process power of each cluster node individually. For that, LINPACK benchmark was used. This benchmark measures the ability each cluster node has on solving linear equation

systems with n x n density, For n = 200, the performance measured for each node was 55571 KFlops^{-1} with a standard deviation of 304 KFlops^{-1}.

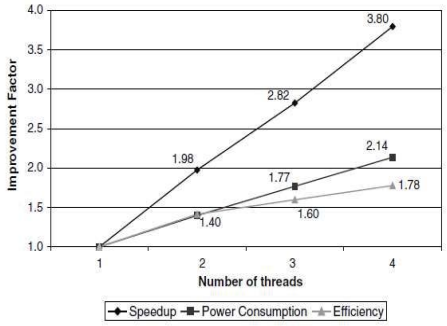


Fig. 7. Impact of OpenMP usage [17].

To verify performance for every cluster node running in parallel, the HPL (*High-Performance LINPACK*) was used. During the tests, the number of nodes in a cluster, as well as the size of the linear systems varied. Analyzing Figure 8 it is possible to realize that using less than 24 nodes for smaller linear systems (with n = 1280) was irrelevant, because the performance kept practically constant.

Furlinger *et al* [11] presents an analysis of energy consumption of a cluster assembled by five apple TV devices. Each device had an ARM Cortex-A8 processor. In order to measure the performance of cluster nodes individually, the authors used the Coremark benchmark. With Coremark it was possible to perform performance tests on integer number operations. To measure performance on operation that manipulates float point numbers on each cluster node, LINPACK benchmark was used. On the other hand, HPL allowed them to measure parallel process power on the cluster. The best performance number they could reach was 160.6 MFlops on arithmetic operations.

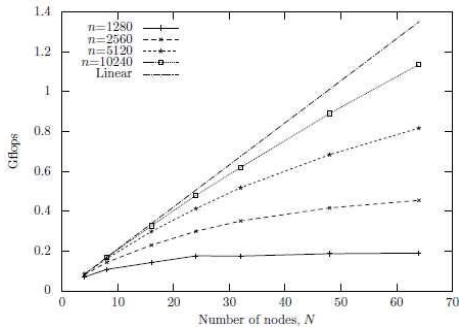


Fig. 8. Performance measure during the execution of HPL benchmark [15].

Silva and Yokoyama`s [16] work aimed to compare the performance of the libraries that manipulated threads. In order to accomplish these tests they have utilized the libraries PTh (*GNU Portable Threads*) [27], Protothreads [28] and PM2 Marcel [29].

PTh libraries allowed every thread to be run at the same address space of the process. The scheduling of multiple threads is no-preemptive. The PM2 Marcel library was developed to support a greater quantity of threads at the same time. The threads created by Protothreads are known as "light" threads, in other words, those threads are run on the same stack.

During the tests, the authors verified the time spent on initialization, creation and join operations. The analysis of results allows us to conclude that Protothreads library had the best performance. This is due to the way the library was implemented. The Figures 9 and 10 represent the results they obtained.

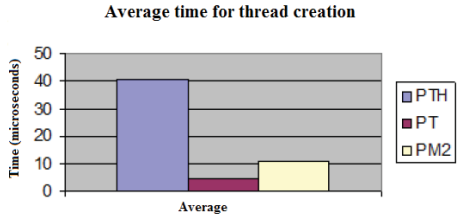


Fig. 9. Average time for thread creation [16].

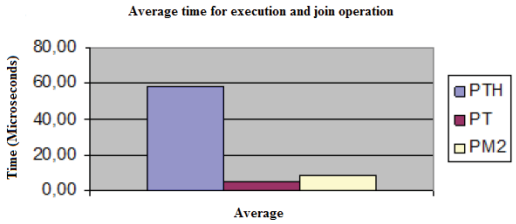


Fig. 10. Average time for execution and join operation [16].

Torelli and Bruno [18] presented a comparative analysis on the libraries OpenMP and PThreads. To accomplish the tests, the authors used both sequential and parallel versions of the algorithm that calculates the Euclidian Distance between two points of an image. The programs were executed with 2D images of different sizes. After testing the sequential version of the algorithm with OpenMP the authors calculated the speedup (see Figure 11).

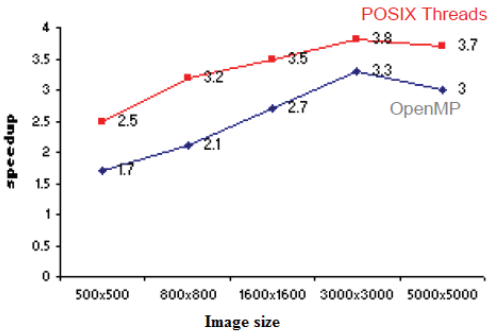


Fig. 11. Speedup from solutions using OpenMP, Pthreads and OpenMP [18].

Jin *et al* [24], have done studies to verify the performance of parallel programs which utilise hybrid solutions. In other words, solutions implemented by a combination of the OpenMP and MPI libraries. By adopting this approach, the

authors have verified a reduction on energy consumption, when compared to the same application developed with only MPI.

IV. Cluster Architecture using ARM Processors and Raspberry Pi

In this section, the details on the construction of the proposed cluster are presented, as well as its characteristics.

A. Raspberry Pi Characteristics

The proposed cluster utilizes Raspberry Pi [31] chips (see Figure 12) with the following settings:

- ARM Processor with 700 MHz.
- SDRAM of 512 Mbytes.
- GPU Dual Core Videocore IV.
- Network Interface 10/100 Ethernet RJ-45.
- HDMI Interface for video output.
- 4 USB 2.0 ports.

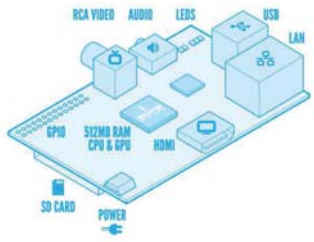


Fig. 12. Raspberry Pi components [31].

B. Raspbian Operating System

In order to control each cluster chip, we use the operating system Raspbian [32], which is optimized for running on Raspberry Pi chips and is derived from Debian Operating System.

C. Cluster Assembling

To assemble the cluster eight type B Raspberry Pi chips are used. The chips communicate through network modules present on each one. The system processor belongs to the AMR platform [12]. Figure 13 presents the example of a Raspberry Pi type B model.

Fig. 13. Raspberry Pi type B model [31].

One of the advantages of using Raspberry Pi is its low-cost, because it possesses every component present on higher value computer. In this sense, it is possible to install an operating system that will manage the chip's components.

After assembling the cluster, the tasks sent via network are parallelized so that they are run by every node of the cluster. Figure 14 presents the sketch from the cluster assembling.

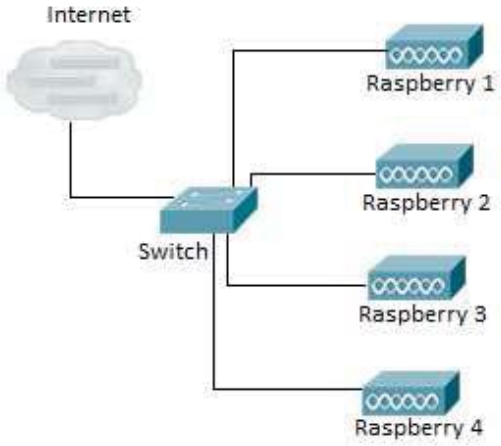


Fig. 14. Raspberry Pi cluster.

Cluster chips are activated individually so that it is possible to utilise specific chips during the program execution. When tasks are sent, the programmer must specify the number of chips he would like.

It is also part of the job the implementation of parallel algorithms to be run by the cluster and, thus, measuring the cluster performance. With this information we have done some analysis and have verified the real efficiency of the cluster when compared to other sequential computing systems.

V. Simulations and Results

In this section, we present the results of simulations to verify the performance of the embedded cluster. For this, we use the *MPI_Wtime()* function, provided by the MPI API to calculate the average time spent after a hundred executions of each algorithm.

In addition to the execution times of each algorithm, we also present the speed-up, which measures the performance of parallel solutions when compared with the respective sequential solutions.

A. Running Matrices Multiplications

Analyzing the results obtained by the simulations with the algorithm that held the matrix multiplication, we can see that the parallelization led to a considerable improvement in the cluster's performance. The simulations take into account the number of nodes and the order of the matrices. The reduction in run time to four nodes in the cluster has reached 66% compared to sequential execution with only one node. Figure 15 shows the simulation times for the matrix multiplication.

By doing an analysis of the found speed-ups, we can see that for matrices with the same, an increase in the number of cluster nodes also caused an increase in the speed-ups. Figure

16 shows the speed-ups to the problem of matrix multiplication.

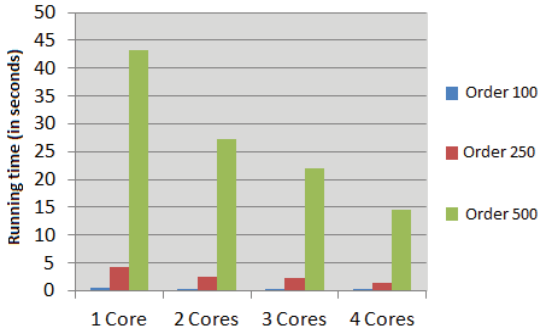


Fig. 15. Simulation times for the matrix multiplication.

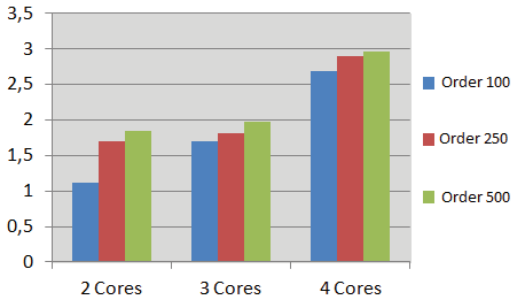


Fig. 16. Speed-ups to the problem of matrix multiplication

B. Running Dot Product

After the simulations with the algorithm that held the dot product of two vectors with sizes 100, 250 and 500, we can see that the parallelization caused a reduction in runtime that reached 52%. Figure 17 shows the algorithm runtimes that held the dot product.

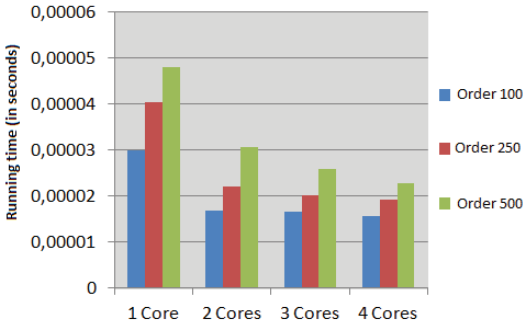


Fig. 17. Simulation times for the dot product

The analysis of speed-ups shows that an increase in the number of cluster nodes caused an increase in performance by reducing the run time (see Figure 18).

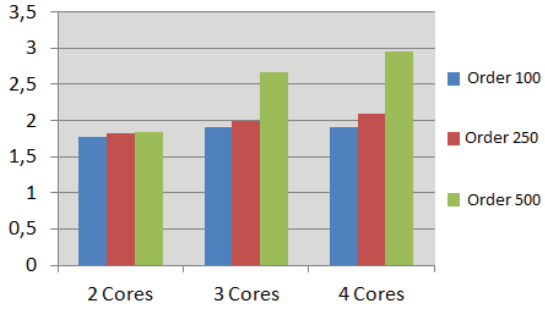


Fig. 18. Speed-ups to the problem of dot product

C. Running Linpack Benchmark

After running the HPL benchmark, we can measure the time taken to solve a dense linear system of order N = 4000, and the amount of floating-point operations (GFLOPS) performed per second.

Analysis of the results shows that an increase in the number of cluster nodes caused a considerable reduction in run time. For the case in which four nodes performed in parallel, the reduction was 64% compared to sequential execution. Regarding the number of transactions, we can see that hears an increase of approximately 65%. Figures 19 and 20 show the results obtained after execution of the benchmark HPL.

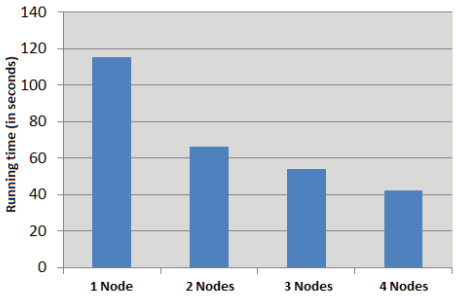


Fig. 19. Execution times for the benchmark HPL

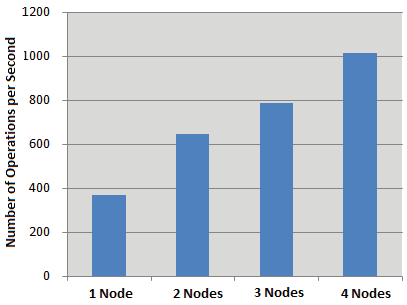


Fig. 20. Number of operations in GFLOPS

VI. Conclusions and Future Work

In this work, we implemented and analyzed the performance of a cluster of low cost with embedded processors from ARM and Raspberry Pi platform. To measure the cluster's performance, we use the HPL benchmark, in addition to algorithms that carried out the matrix multiplication and the dot product of two vectors.

After performing this work, we can make the following observations: (i) the increase in the number of cluster nodes caused a considerable reduction in the application runtime; (ii) an increase in the amount of processed data proved an improvement in the performance of the cluster. This is due to the fact that for small amounts of data, the time spent in communicating between nodes exceeds the processing time, making the solution more efficient sequential; (iii) after analyzing the applications performed and the results obtained by running the HPL benchmark, we can conclude that the cluster shipped with ARM processors performed well when running parallel applications.

References

[1] Moore, G. E., "Cramming More Components onto Integrated Circuits". Electronics, 38(8):114-117, April, 1965

[2] OpenMP, Available in: http://openmp.org/wp/. Accessed December 18, 2014.

[3] MPBench, Available in: http://icl.cs.utk.edu/projects/llcbench/mpbench.html. Accessed December 18, 2014.

[4] COSTA, Ricardo A. G., "Desempenho e Consumo de Energia de Algoritmos Criptográficos do MiBench em Sistemas Móveis". Monograph, Amazonas State University, November 2007.

[5] GUTHAUS, M.; RINGENBERG, J.; ERNST, D.; AUSTIN, T.; MUDGE, T. and BROWN, R., "MiBench: A Free, Commercially Representative Embedded Benchmark Suite". In Proc. of the IEEE 4th Annual Workshop on Workload Characterization (WWC), USA, pages 3-14, December 2001.

[6] LIN, C.-H.; SHIH, C.-S.; LIU, J.-C.; CHENG, M.-H. and LEE, Y.-W. "Energy Efficiency Measurement for Multimedia Audio Decoding on Embedded Systems". In: 2nd International Conference on Ubiquitous Information Management and Communication (ACM ICUIMC), Suwon, Korea, January 31 - February 01, 2008.

[7] Intel, "Inte's Tera-Scale Research Prepares for Tens, Hundreds of Cores". Technology@Intel Magazine, 2006, Available in: http://www.intel.com/technology/magazine/computing/tera-scale-0606.pdf. Accessed January 4, 2015.

[8] Windows API Index, Available in: https://msdn.microsoft.com/en-us/library/windows/desktop/hh920508%28vs.85%29.aspx. Accessed December 20, 2014.

[9] POSIX Threads Programming, Available in: https://computing.llnl.gov/tutorials/pthreads/. Accessed December 21, 2014.

[10] PACHECO, Peter, "An Introduction to Parallel Programming". – Morgan Kaufmann, 1st edition, 2011, 392p.

[11] FÜRLINGER, Karl; KLAUSECKER, Christof; KRANZLMÜLLER, Dieter, "The AppleTV-Cluster: Towards Energy Efficient Parallel Computing on Consumer electronic Devices". 2011.

[12] ARM Company Profile, Available in: http://www.arm.com/about/company-profile/index.php. Accessed November 12, 2014.

[13] NEILL, Richard; SHABARSHIN, Alexander and CARLONI, Luca P. "A Heterogeneous Parallel System Running Openmpi on a Broadband Network of Embedded set-top Devices". In Proceedings of the 7th ACM International Conference on Computing Frontiers, CF'10, pages 187-196, New York, NY, USA, 2010.

[14] SÁ, Aléx G. C.; PEREIRA, Marluce, R.; MOURA, Pedro M. and PEIXOTO, Luiz Henrique R., "A Comparative Study of Multithreaded Applications Performance in Different Scenarios". Magazine of Information System the FSMA, 1(9):45-53, 2012.

[15] COX, Simon J.; COX, James T.; BOARDMAN, Richard P.; JOHNSTON, Steve J.; SCOTT, Mark and O'BRIEN, Neil, "Iridis-pi: a Low-Cost, Compact Demonstration Cluster". Cluster Computing, 1(17):349-358, June 2013.

[16] SILVA, R. R. and YOKOYAMA, R. S., "Avaliação de Desempenho da Utilização de Threads em User Level em Linux". Magazine of Theoretical and Applied Informatics (RITA). 1(18):112-132, 2011.

[17] BLUME, H.; LIVONIUS, J. Von; ROTENBERG, L.; NOLL, T. G.; BOTHE, H. and BRAKENSIEK, J., "OpenMP-Based Parallelization on an MPCore Multiprocessor Platform – A Performance and Power Analysis". Journal of Systems Architecture (JSA), 11(54):1019-1029, November, 2008.

[18] TORELLI, J. C. and BRUNO, O. M., "Programação Paralela em SMPS com OPENMP e POSIX Threads: Um Estudo Comparativo". Proceedings of the IV Brazilian Congress of Computing (CBComp). Volume 1, pages 486-491, 2004.

[19] JAIN, Raj and PAUL, Subharthi, "Network Virtualization and Software Defined Networking for Cloud Computing: A Survey". IEEE Communications Managzine, 51(11):24-31, November, 2013.

[20] SkaMPI-5 benchmark, Available in: http://liinwww.ira.uka.de/~skampi/. Accessed January 12, 2015.

[21] McCOOL, Michael; REINDERS, James and ROBISON, Arch, "Structured Parallel Programming - Patterns for Efficient Computation". – Morgan Kaufmann; 1st edition, 2012, 432p.

[22] SILBERSCHATZ, A. S.; GALVIN, P. and GAGNE, G., "Sistemas Operacionais - Conceitos e Aplicações". – São Paulo: Campus, 2001. 585p.

[23] GEBALI, Fayez, "Algorithms and Parallel Computing". – New Jersey: Wiley, 1st edition, 2011, 341p.

[24] JIN, Haoqiang; JESPERSEN, Dennis; MEHROTRA, Piyush; BISWAS, Rupak; HUANG, Lei and CHAPMAN, Barbara, "High performance computing using MPI and OpenMP on multi-core parallel systems", Journal Parallel Computing, 37(9):562–575, September, 2011.

[25] MPI, Available in: http://www.open-mpi.org/. Accessed December 18, 2014.

[26] Intel OpenMP, Available in: https://software.intel.com/en-us/articles/getting-started-with-openmp. Accessed December 19, 2014.

[27] Pth, Available in: www.gnu.org/s/pth/. Accessed January 12, 2015.

[28] Protothreads, Available in: http://dunkels.com/adam/pt/. Accessed January 12, 2015.

[29] PM2 Marcel, Available in: http://runtime.bordeaux.inria.fr/marcel/. Accessed January 12, 2015.

[30] ZOMAYA, Albert Y. Z and LEE, Young Choon, "Energy Efficient Distributed Computing Systems". – Wiley-IEEE Computer Society Press, August, 2012, 856p.

[31] Raspberry Pi, Available in: www.raspberrypi.org/. Accessed November 7, 2014.

[32] Raspbian, Available in: www.raspbian.org/. Accessed November 7, 2014.

A Framework for Scheduling Real-Time Systems

Zhuo Cheng*, Haitao Zhang†, Yasuo Tan*, and Yuto Lim*
*School of Information Science, Japan Advanced Institute of Science and Technology, Japan
{chengzhuo, ytan, ylim}@jaist.ac.jp
†School of Information Science and Engineering, Lanzhou University, China
htzhang@lzu.edu.cn

***Abstract*—Real-time system is playing an important role in our society. For such a system, sensitivity to timing is the central feature of system behaviors, which means tasks in the systems are required to be completed before their deadlines. To guarantee this requirement, the design of scheduling is crucial. In this paper, based on *satisfiability modulo theories (SMT)*, we provide a framework to design scheduling for real-time systems. In the framework, the problem of scheduling is treated as a satisfiability problem. The key work is to formalize the satisfiability problem using first-order language. After the formalization, a SMT solver (e.g., Z3, Yices) is employed to solver such a satisfiability problem. An optimal schedule can be generated based on a solution model returned by the SMT solver. To demonstrate the practicality of the framework, we give design guidelines for real-time systems with multiprocessor. Through the demonstration, the framework is found flexible and sufficiently general to apply to different kinds of real-time systems. To the best of our knowledge, it is the first time that systematically introducing SMT to solve a series problems covering a wide range in real-time scheduling domain.**

Keywords*—real-time scheduling, SMT, multiprocessor, satisfiability problem

I. Introduction

Real-time system is playing an important role in our society. For example, chemical and nuclear plant control, space missions, flight control, telecommunications, and multimedia systems are all real-time systems [1]. In such a system, sensitivity to timing is the central feature of system behaviors, which means, tasks in the system are required to be completed before their deadlines. To provide such guarantee, the design of scheduling is crucial.

The research on real-time scheduling has lasted for decades, but still lots of challenges remain [5]. For example, limited task models for multiprocessor systems, limited policies for access to shared resources, ineffective schedulability tests, limited scheduling methods. In this paper, we try to address the challenge *limited scheduling methods*.

For designing scheduling method, many research has contributed to this area [8, 9, 10, 11]. But one important problem is that all the proposed methods are specified on either a specific system architecture (e.g., uniprocessor) or specific scheduling target (e.g., make all task meet deadline). Usually, it is quite difficult, even impossible, to adapt one scheduling method to another application scenario. This becomes a main obstacle for designing scheduling for a new application system and results in a high design cost. In this paper, we try to solve the problem by proposing a framework to design scheduling for real-time systems. The main contributions of this paper are as follows.

i) We propose a scheduling framework based on satisfiability modulo theories (SMT). In this framework, the problem of scheduling is treated as a satisfiability problem. The key work is to formalize the satisfiability problem using first-order language. We use a *sat model* to represent the formalized problem. This sat model is a set of first-order logic formulas (within linear arithmetic in the formulas) which express all the scheduling constraints that a desired optimum schedule should satisfy. After the sat model is constructed, a SMT solver (e.g., Z3 [6], Yices [7]) is employed to solve the formalized problem. An optimal schedule can be generated based on a *solution model* returned by the SMT solver. The correctness of this method and the optimality of the generated schedule are straightforward.

ii) The proposed scheduling framework is flexible. In the SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. It means if we want to design scheduling to achieve other objectives, only the target constraint needs to be modified. Or, if we want to achieve the same scheduling objective for another real-time system with different system architecture, only the system constraints need to be modified.

iii) We give practical design guidelines for scheduling multiprocessor systems. These design guidelines are for systems with multiprocessor, and of course, they are also applicable for system with uniprocessor, as scheduling uniprocessor systems is a sub problem of scheduling multiprocessor systems. The model for the multiprocessor system is defined in a very general way, and in the design guidelines, we have considered systems with mixed-criticality (containing both firm and soft) real-time functions, task dependency relation, task migration cost, heterogeneous processors (processors with different processing speed and architectures), heterogeneous network channels (network channels with different data transfer speed and supporting different network protocols). All these efforts make the framework practicable and sufficiently general to apply to different kinds of real-time systems and different scheduling targets, which can benefit system designers to efficiently design scheduling.

The remainder of this paper is organized as follows. In Section II, we present scheduling framework which is based on satisfiability modulo theories. The system model is denoted in Section III. We give the design guidelines for system constraints in Section IV, while Section V gives the design guidelines for target constraints. Related work are summarized in Section VI. Section VII concludes the paper.

Haitao Zhang is the corresponding author.

TABLE I. SYMBOLS AND DEFINITIONS

Symbol	Definition
t	system time instant
δ	network precision
$\mathcal{F}$	set of functions of a real-time system
$\mathcal{FH} \subseteq \mathcal{F}$	set of functions with firm deadlines
$\mathcal{FS} \subseteq \mathcal{F}$	set of functions with soft deadlines
$F_i \in \mathcal{F}$	function of a real-time system, i is the index of the function
r_i	triggered time instant of function F_i
d_i	deadline of function F_i
v_i	obtained value by completing function F_i before deadline
$\mathcal{T}$	set of all the tasks in the real-time systems
$T_i \subseteq \mathcal{T}$	set of tasks corresponding to function F_i
$\tau_j \in \mathcal{T}$	task, j is index of the task
c_j	computation cost of τ_j
m_j	migration cost of τ_j from a processor to another one
τs_i	start task of task poset $(T_i, \prec)$
τe_i	end task of task poset $(T_i, \prec)$
$\mathcal{P}$	set of processors
$p_a \in \mathcal{P}$	processor, where a is the index of the processor
ps_a	speed of processor p_a
$TS_a \subseteq \mathcal{T}$	task set that can be completed by processor p_a
$TS_{a\to b} \subseteq \mathcal{T}$	task set that can migrate on network channel $n_{a\to b}$
$\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$	set of network channels
$n_{a\to b} \in \mathcal{N}$	network channel from processor p_a to p_b
$ns_{a\to b}$	speed of $n_{a\to b}$

II. THE SMT-BASED SCHEDULING FRAMEWORK

A. Satisfiability Modulo Theories (SMT)

Satisfiability modulo theories checks the satisfiability of logic formulas in first-order formulation with regard to certain background theories like linear integer arithmetic or bit-vectors [2]. A first-order logic formula uses variables as well as quantifiers, functional and predicate symbols, and logic operators [3]. A formula F is *satisfiable*, if there is an interpretation that makes F `true`. For example, formula $\exists a, b \in \mathbb{R}, (b > a + 1.0) \wedge (b < a + 1.1)$, where $\mathbb{R}$ is real number set, is satisfiable, as there is an interpretation, $a \mapsto -1.05, b \mapsto 0$, that makes F `true`. On the contrast, a formula F is *unsatisfiable*, if there does not exist an interpretation that makes F `true`. For example, if we define $\exists a, b \in \mathbb{Z}$, where $\mathbb{Z}$ is integer set, the formula $(b > a+1.0) \wedge (b < a+1.1)$ will be unsatisfiable.

For a satisfiability problem that has been formalized by first-order logic formulas, a SMT solver (e.g., Z3, Yices) can be employed to solver such a problem. If all the logic formulas are satisfiable, SMT solver returns the results `sat` and a *solution model* which contains an interpretation for all the variables defined in the formulas that makes the formulas `true`. For the case $\exists a, b \in \mathbb{R}$, the model is: $a \mapsto -1.05, b \mapsto 0$. If there is an unsatisfiable logic formula, SMT solver returns the results `unsat` with an empty model, for the case $\exists a, b \in \mathbb{Z}$.

B. The Scheduling Framework

The framework of the SMT-based scheduling is illustrated in Fig. 1. In a real-time system, a schedule (execution order of tasks) is generated by a scheduler. The problem of scheduling can be treated as a satisfiability problem.

In order to use SMT to solve this satisfiability problem, the key work is to formalize the problem using first-order language. We use a *sat model* to represent the formalized problem. This sat model is the set of first-order logic formulas (within linear arithmetic in the formulas) which expresses all the constraints that the desired schedule should satisfy. There are two kinds of constraints: *system constraints* and *target constraints*. System constraints are based on the specific system. For example, if two tasks run on a processor, a schedule should make sure that the execution of these two tasks cannot have overlap in time domain. Target constraint is based on the scheduling target. For example, under normal workload condition, the desired schedule should make all the functions meet their deadlines (completed before deadlines).

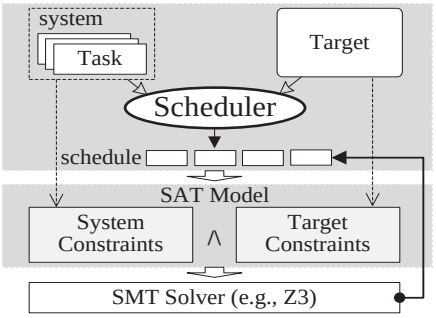


Fig. 1. The framework for scheduling real-time system based on SMT

After the sat model is constructed, it can be inputted into a SMT solver (e.g., Z3). A *solution model* will be returned by the SMT solver. This solution model gives an interpretation for all the variables defined in the sat model, and under the interpretation, all the logic formulas in the sat model are evaluated as `true`. It means the satisfiability problem represented by the sat model is solved, and based on this interpretation, the desired schedule can be generated.

III. SYSTEM MODEL

A. Function Set

Function set define the functions that can be achieved by a real-time system. Let $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ denote the function set. Each function $F_i \in \mathcal{F}$ is achieved by a corresponding series tasks, represent as *poset* $(T_i, \prec)$, $T_i \neq \emptyset$ denotes the set of the corresponding task, and $\prec$ denotes the dependency relation of tasks in T_i (the detail of the poset will be explained in the next subsection). For real-time systems, when functions are triggered at system time instant r, they are required to be completed before a specific time, which is called *deadline*, represented by d_i. Moreover, different functions have different degrees of importance to the system, we use v_i to denote the values obtained by the system through completing functions F_i before its deadline d_i. Based on above explanation, we define the function $F_i = ((T_i, \prec), r_i, d_i, v_i)$.

Note that, unlike many research on real-time scheduling that set deadlines to tasks, we set deadline to the function level rather than task level. This setting can better reflect the reality that the deadline requirement is for the functions of real-time systems, while a function is achieved by a series tasks cooperated together.

This function definition denotes the functions which have firm deadlines. That is, for such a function, if it misses its

deadline, system will not obtain any value through completing it. Usually, in a complex real-time system, not all functions are firm real-time functions. Some functions are soft real-time functions. For such a function, if it misses deadline, it will still be useful for the system, but the value obtained by completing such function will be less than completing it before its deadline. To denote the function with soft deadlines, without losing generality, we define such functions as: $F_i = ((T_i, \prec), r_i, d_i, f_i(t))$, where $f_i(t)$ is the coefficient function to indicate the value that the system can obtain by completing function F_i at system time instant t. The reasonable value of the coefficient function is in interval $[0\ 1]$. For convenience, we use $\mathcal{FH} \subseteq \mathcal{F}$ to denote the set of functions with firm deadlines, and use $\mathcal{FS} \subseteq \mathcal{F}$ to denote the set of functions with soft deadlines.

B. Task Poset

A multiprocessor real-time system comprises a set of tasks, denoted by $\mathcal{T}$. For each function, it is achieve by a series tasks cooperated together. Poset $(T_i, \prec)$ is used to denote such a series tasks, where $T_i \subseteq \mathcal{T}$ is the task set corresponding to F_i, and $T_i = \{\tau_1, \tau_2, \ldots, \tau_m\}$, where $\tau_j \in T_i$ is a task, and m is the number of tasks. We use $\tau_{i,j}$ to indicate task $\tau_j \in T_i$. We assume that, $\texttt{if}\ \ |\mathcal{F}| > 1, \texttt{then}\ \ \forall T_i, T_j \subset \mathcal{T}, i \neq j \implies T_i \cap T_j = \emptyset$. That is, no tasks are shared by different functions [1]. The symbol $\prec$ indicates the dependency relation between two tasks. That is, $\tau_k, \tau_j \in T_i, \tau_k \prec \tau_j$ indicate that task τ_j can start to run only after task τ_k has been completed. The dependency relation is transitive. That is, $\tau_k \prec \tau_j, \tau_j \prec \tau_l \implies \tau_k \prec \tau_l$.

Definition (start task). *A start task of $(T_i, \prec)$ is such a task $\tau_i \in T_i$ that starts earliest of all the tasks in T_i, that is, $\forall \tau_j \in T_i, i \neq j \implies \tau_i \prec \tau_j$.*

Definition (end task). *A end task of $(T_i, \prec)$ is such a task $\tau_i \in T_i$ that starts latest of all the tasks in T_i, that is, $\forall \tau_j \in T_i, i \neq j \implies \tau_j \prec \tau_i$.*

Without losing generality, we assume that there is one start task and one end task of $(T_i, \prec)$, and use τs_i and τe_i to indicate the start task and end task of task poset $(T_i, \prec)$, respectively [2]. Each task has two parameters, $\tau_j = (c_j, m_j)$, where j is the index of the task. c_j is the required computation cost, which means the number of time slots (ticks of processor) needed by a unit speed processor to complete task τ_j; and m_j is the required migration cost for task τ_j migrating from a processor to another one. We use the parameter m_j combined with parameters of network (the details will be explained later) to calculated the overheads of migrating tasks.

C. Processor Set

In multiprocessor real-time systems, different processors are used to execute tasks. We use $\mathcal{P} = \{p_1, p_2, \ldots, p_l\}$ to denote the set of processors, where l is the number of processors. Each processor p_a is a 2-tuple, $p_a = (ps_a, TS_a)$, where a is the index of the processor. ps_a is the speed of the processor. When task τ_i running on processor p_a, the number of time slots needed for processor p_a to complete task τ_i, represented by *task completion* tc_a^i:

$$tc_a^i = \frac{c_i}{ps_a} \tag{1}$$

TS_a is the task set that can be completed by processor p_a. This parameter is for *heterogeneous* systems, as in such systems, processors have different architectures, some tasks can only be executed on some specific processors. If $TS_a = \emptyset$, it means processor p_a cannot be used to execute any task in the system.

Processors have independent local clocks, they are synchronized with each other in the time domain through synchronization protocol. The maximum difference between the local clocks of any two processors in the networked systems is called *network precision* (also called synchronization jitter) which is a global constant. We denote the network precision with δ.

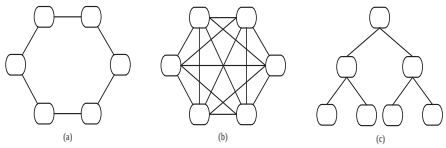


Fig. 2. Different types of network topologies: a. ring, b. mesh, c. tree

D. Network Channel Set

In multiprocessor real-time systems, processors are connected through network channels. We use $\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$ to denote the set of network channel. $n_{a \to n} \in \mathcal{N}$ denotes the network channel from processor p_a to p_b, where $p_a, p_b \in \mathcal{P}, a \neq b$. Since we consider bi-directional network channel, we have $\forall n_{a \to b} \in \mathcal{N} \implies n_{b \to a} \in \mathcal{N}$. We use $ns_{a \to b}$ to represent the speed of $n_{a \to b}$.

Note that, define the network channel set as $\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$ makes the system model become very general which includes any types of network topologies. For example, as shown in Fig. 2, the network channel set for mesh topology (b in the Fig. 2) equals to $\mathcal{P} \times \mathcal{P}$, while the ring and tree topologies is the subset of $\mathcal{P} \times \mathcal{P}$. Moreover, this definition is also suitable for processor with multi-cores. For example, for a processor A with four cores, in this definition, can be represented as four processors connect with network channels in mesh topology, and the speed of networks is set based on the data transfer speed inside the processor A.

When the data of the computed results of task τ_i migrates from processor p_a to p_b[3], the time slots spent on network channel, represented by $tm_{a \to b}^i$, can be calculated as:

$$tm_{a \to b}^i = \frac{m_i}{ns_{a \to b}} \tag{2}$$

Based on $tm_{a \to b}^i$, we can get the time instant that processor p_b receives the data of task τ_i migrating from processor p_a

[1] Note that, this assumption is for concise expression. In real systems, if task $\tau_k \in \mathcal{T}$ is used by function F_i and F_j, we can use two tasks τ_{ik} and τ_{jk}, to represent τ_k used in function F_i and F_j, respectively.

[2] To express a function with many starts (end) tasks, we can set a virtual task, with empty operation, start before all the starts tasks (start after all the end task) to be the start (end) task.

[3] For conciseness, we say "task τ_i migrates from processor p_a to p_b" to mean "the data of the computed results of task τ_i migrates from processor p_a to p_b" in the reset of the paper.

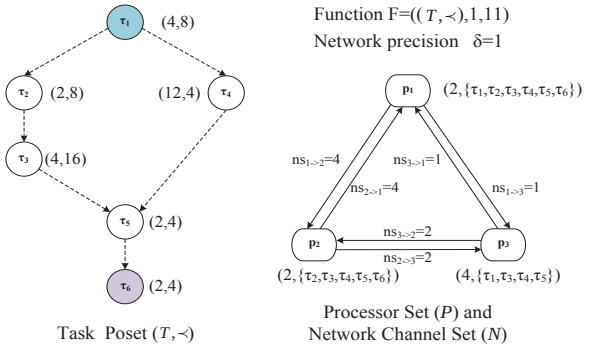


Fig. 3. An example for scheduling multiprocessor real-time systems

through network channel $n_{a\to b}$, represented by $r^i_{a\to b}$, as

$$r^i_{a\to b} = s^i_{a\to b} + tm^i_{a\to b} + \delta \tag{3}$$

where, $s^i_{a\to b}$ is the start time of τ_i migrating through network channel $n_{a\to b}$, and δ is the network precision.

For a distributed real-time system, different processors are connected through network channels which are built by routers. As different routers support different network protocols, some tasks may not be migrated through some network channels. To capture this characteristics, similar as the heterogeneous processors, we also can define the heterogeneous network channels. We use $TS_{a\to b}$ to denote that task set that can be transferred through network channel $n_{a\to b}$.

E. Assumptions

Applied to this system model, we require that all the parameters of the functions and tasks are known a prior. This requirement makes the model become a generalization of the widely studied *period task model*, in which all the tasks in the system are released periodically. This means our method applies more broadly than other methods which are specified on period task model. To guarantee a certain level of determinacy, in this paper, task preemption is not allowed.

To illustrate the defined system model, an example of scheduling for multiprocessor real-time systems is shown in Fig. 3. In this example, there are three processors p_1, p_2, p_3 in the system. These processors are connected with each other through six network channels $n_{1\to2}, n_{2\to1}, n_{1\to3}, n_{3\to1}, n_{2\to3}, n_{3\to2}$, and these network channels support all the migration of all the tasks in T. The network precision δ is 1. In the system, a firm real-time function $F = ((T, \prec), 1, 11)$ is waiting to be executed on the processors. The task poset of the function is $(T, \prec)$ which consists of six tasks. Task dependency relations are described in a directed acyclic graph. An edge starting from task τ_i to task τ_j represented by a dotted line denotes a dependency relation $\tau_i \prec \tau_j$.

IV. System Constraints

This subsection describes all the system constraints expressed in the sat model for the defined multiprocessor systems.

A. Constraint on start execution time of functions

Task set T_i corresponding to function F_i can start to run only after the function is triggered. That is, the start execution time of the start task of the poset $(T_i, \prec)$ should be larger than the triggered time of function F_i.

$$\begin{aligned} &\forall F_i \in \mathcal{F}, \forall p_a \in \mathcal{P} \\ &s^{\tau s_i}_a \geq r_i \end{aligned} \tag{4}$$

where symbol $s^{\tau s_i}_a$ denotes the start execution time of task τs_i on processor p_a.

B. Constraint on start time of task migration

If a task τ_i migrates from processor p_a to processor p_b through network channel $n_{a\to b}$, it means *i):* task τ_i has been completed by processor p_a; or *ii):* τ_i has migrated to processor p_a from another processor. For the first case, task τ_i can start to migrate after it has been completed, and for the second case, task τ_i can start to migrate after it has already migrated to processor p_a.

$$\begin{aligned} &\forall \tau_i \in \mathcal{T}, \forall n_{a\to b} \in \mathcal{N}, \exists n_{c\to a} \in \mathcal{N} \\ &(s^i_{a\to b} \geq s^i_a + tc^i_a) \vee (s^i_{a\to b} \geq r^i_{c\to a}) \end{aligned} \tag{5}$$

where symbol $s^i_{a\to b}$ denotes the start time of task τ_i migrating through network channel $n_{a\to b}$, s^i_a denotes the start execution time of task τ_i on processor p_a.

C. Constraint on task dependency

For processor p_a, if $\tau_i \prec \tau_j$, task τ_j can start to run only after τ_i has been completed. Similar to the constraints on start time of task migration, there are two cases. *i):* task τ_i has been completed by processor p_a; *ii):* task τ_i has migrated to processor p_a from another processor. For the first case, τ_j can start to run after τ_i has been completed, and for the second case, τ_j can start to run after τ_i has already migrated to processor p_a.

$$\begin{aligned} &\forall \tau_i, \tau_j \in \mathcal{T}, \forall p_a \in \mathcal{P}, \exists n_{b\to a} \in \mathcal{N} \\ &\tau_i \prec \tau_j \implies (s^j_a \geq s^i_a + tc^i_a) \vee (s^j_a \geq r^i_{b\to a}) \end{aligned} \tag{6}$$

D. Constraint on execution of processors

A processor can execute only one task at a time. This is interpreted as: there is no overlap of the execution time of any two tasks.

$$\begin{aligned} &\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall p_a \in \mathcal{P} \\ &(s^i_a \geq s^j_a + tc^j_a) \vee (s^j_a \geq s^i_a + tc^i_a) \end{aligned} \tag{7}$$

E. Constraint on network channels

A network channel can transfer data of only one task at a time. That is, there is no overlap of the migration time of any two tasks on a network channel.

$$\begin{aligned} &\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall n_{a\to b} \in \mathcal{N} \\ &(s^i_{a\to b} \geq s^j_{a\to b} + tm^j_{a\to b}) \vee (s^j_{a\to b} \geq s^i_{a\to b} + tm^i_{a\to b}) \end{aligned} \tag{8}$$

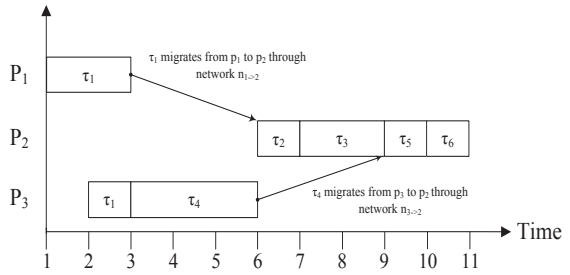


Fig. 4. The scheduling result for example shown in Fig. 3 by using the proposed SMT-based scheduling

F. Constraint on heterogeneous processors

In heterogeneous systems, processors have different architectures, some tasks can only be executed on some specific processors. For tasks that cannot be executed on some processors, the start execution time of the tasks in such processors are set to $+\infty$, which means the tasks will never start to run on these specific processors.

$$\begin{aligned} &\forall p_a \in \mathcal{P}, \forall \tau_i \in \mathcal{T} - TS_a \\ &s_a^i = +\infty \end{aligned} \tag{9}$$

G. Constraint on heterogeneous network channels

For a distributed real-time system, different processors are connected through network channels which are built by routers. As different routers support different network protocols, some tasks may not be migrated through some network channels. Similar as the constraint on heterogeneous processors, for tasks that cannot migrate on some network channels, the start migration time of the tasks in such network channels are set to $+\infty$, which means the tasks will never start to migrate on these specific network channels.

$$\begin{aligned} &\forall n_{a\to b} \in \mathcal{N}, \forall \tau_i \in \mathcal{T} - TS_{a\to b} \\ &s_{a\to b}^i = +\infty \end{aligned} \tag{10}$$

V. Target Constraints

There are many targets can be considered when we design scheduling for real-time systems. Which objectives are appropriate in a given situation depends, of course, upon the application. In this section, we give design guidelines for different scheduling targets.

A. Making all the functions meet their deadlines

Under normal workload conditions, the desired schedule should make sure that every triggered function can be completed before its deadline.

$$\begin{aligned} &\forall F_i \in \mathcal{F}, \exists p_a \in \mathcal{P} \\ &s_a^{\tau e_i} + tc_a^{\tau e_i} \le d_i \end{aligned} \tag{11}$$

where symbol $s_a^{\tau e_i}$ is the start execution time of task τe_i on processor p_a, and $tc_a^{\tau e_i}$ is the number of time slots needed for processor p_a to complete task τe_i.

Based on this scheduling target, recall the example shown in Fig. 3, we can get the solution model $\mathcal{M}$ which defines the values of the start time of task execution on processor, s_a^j, and the start time of task migration through network, $s_{b\to c}^j$, for $\forall f_i \in \mathcal{F}, \forall \tau_j \in T_i, \forall p_a \in \mathcal{P}, \forall n_{b\to c} \in \mathcal{N}$. Based on the model $\mathcal{M}$, we can get the scheduling results as shown in Fig. 4. This scheduling sequence can make the function F in Fig. 3 meet its deadline. Some characteristics of this scheduling sequence should be noticed:

- Task τ_1 has been executed on processor p_1 from system time $t = 1$ to $t = 3$, and it has also been executed on processor p_3 from system time $t = 2$ to $t = 3$. This means, the SMT-based scheduling framework can handle the parallel execution of tasks, and can make a task repeatedly run on different processors when such repeated execution is necessary.
- Task τ_2 runs on processor p_2 from $t = 6$ to $t = 7$. Although task τ_2 needs the computed results from completing task τ_1, such computed results can not only be obtained by completing task τ_1 on processor p_2 itself, but also can be obtained by transferring the computed results from other processor that has completed task τ_1. Specified to this example, at system time $t = 6$, processor p_2 gets the computed results of task τ_1 from processor p_1.

B. Maximizing obtained values of completed functions

Under normal workload conditions, there exist a schedule can make all the triggered functions meet their deadlines. However, in practical environment, system workload may vary widely because of dynamic changes of work environment. Once system workload becomes too heavy so that there does not exist a feasible schedule can make all the functions meet their deadlines, we say the system is *overloaded*. When system is overload, one reasonable scheduling target is to maximize the obtained values of the completed functions.

Let symbol v be the obtained values of the completed functions, and its initial value is set to be 0. For functions with firm deadlines, system can obtain their values only when such functions have been completed before their deadlines.

$$\begin{aligned} &\forall F_i \in \mathcal{FH} \\ &\texttt{if } \exists p_a \in \mathcal{P}, s_a^{\tau e_i} + tc_a^{\tau e_i} \le d_i \\ &\quad v := v + v_i \\ &\texttt{end} \end{aligned} \tag{12}$$

For completing functions F_i with soft deadlines, the value that the system can obtain is according to the coefficient functions $f_i(t)$, where t is the time when the system completes the function, and it can be calculated as follows.

$$\begin{aligned} &\forall F_i \in \mathcal{FS} \\ &\texttt{if } \exists p_a \in \mathcal{P}, s_a^{\tau e_i} + tc_a^{\tau e_i} \le d_i \\ &\quad v := v + f_i(s_a^{\tau e_i} + tc_a^{\tau e_i}) \\ &\texttt{end} \end{aligned} \tag{13}$$

Let symbol sv denote the maximum obtained values of the completed functions, and obviously, sv is no less than 0 and no larger than the sum of the values of the firm deadline functions ($\sum v_i$ for $\forall F_i \in \mathcal{FH}$) and the values of the soft deadline functions when all the soft deadline tasks are completed before deadlines ($\sum f_i(t_i)$, for $\forall F_i \in \mathcal{FS}$, where t_i is the completed

time instant of task τ_i, and $t_i < d_i$), represented as `max`. The constraints on the scheduling target can be expressed as:

$$v = sv \tag{14}$$

C. Making firm deadline functions meet deadlines while maximizing obtained values of the completed soft deadline functions

Since firm deadline functions usually play important roles in a real-time system, when system is under overload condition, a reasonable scheduling target is to first make sure that all the firm deadline functions meet their deadlines, meanwhile, maximizing obtained values of the completed soft deadline functions. To make firm deadline functions meet deadlines, we can get

$$\begin{aligned} &\forall F_i \in \mathcal{FH}, \exists p_a \in \mathcal{P} \\ &s_a^{\tau e_i} + tc_a^{\tau e_i} \leq d_i \end{aligned} \tag{15}$$

To maximize the obtained value of the completed soft deadline functions, the formula is similar as it for the previous scheduling target. Let symbol v be the obtained values of the completed functions, and its initial value is set to be 0. The values system can obtain by completing the soft deadline functions can be calculated as follows.

$$\begin{aligned} &\forall F_i \in \mathcal{FS} \\ &\texttt{if } \exists p_a \in \mathcal{P}, s_a^{\tau e_i} + tc_a^{\tau e_i} \leq d_i \\ &\quad v := v + f_i(s_a^{\tau e_i} + tc_a^{\tau e_i}) \\ &\texttt{end} \end{aligned} \tag{16}$$

Let symbol sv denote the maximum obtained values of the completed soft deadline functions, and obviously, sv is no less than 0 and no larger than the values of the soft deadline functions when all the soft deadline tasks are completed before deadlines ($\sum f_i(t_i)$, for $\forall F_i \in \mathcal{FS}$, where t_i is the completed time instant of task τ_i, and $t_i < d_i$). The constraints on scheduling target can be expressed as:

$$v = sv \tag{17}$$

VI. Related Work

The research on real-time scheduling has lasted for decades, many research have been conducted on this area. For research on designing scheduling for multiprocessor systems, a comprehensive survey can be found in [5]. In [8], the *Proportionate Fair* (Pfair) algorithm was introduced. Pfair is a schedule generation algorithm which is applicable to periodic tasksets with implicit deadlines. It is based on the idea of fluid scheduling, where each task makes progress proportionate to its utilization. Pfair scheduling divides the timeline into equal length quanta or slots. Authors in [8] showed that the Pfair algorithm is optimal for periodic tasksets with implicit deadlines. In [9], authors extended the PFair approach to sporadic tasksets, showing that the EPDF (earliest pseudodeadline first) algorithm, a variant of Pfair, is optimal for sporadic tasksets with implicit deadlines executing on two processors, but is not optimal for more than two processors.

Some approaches focus on studying task and messages schedule co-synthesis in switched time-triggered networks. In [10], authors studied time-triggered distributed systems where periodic application tasks are mapped onto different end stations (processing units) communicating over a switched Ethernet network. They try to solve the scheduling problem using a MIP multi-objective optimization formulation. In [11], authors studied the system consisting of communicating event- and time-triggered tasks running on distributed nodes. These tasks are scheduled in conjunction with the associated bus messages by using dynamic and static scheduling methods, respectively.

Hitherto, most of the presented methods are either limited to specific task model (e.g., [8, 10] limited to periodic tasksets) or simple system architecture (e.g., [9] limited to two processors, [11] simple bus network topologies). Compared with these works, our proposed framework is flexible and sufficiently general to apply to various kinds of real-time systems and various scheduling targets, which makes that our framework applies much more widely.

VII. Conclusion

In this paper, based on satisfiability modulo theories (SMT), we provide a framework to design scheduling for real-time systems. In the framework, the problem of scheduling is treated as a satisfiability problem. After using first-order language to formalize the satisfiability problem, a SMT solver is employed to solver such a problem. An optimal schedule can be generated based on a solution model returned by the SMT solver. To demonstrate the practicality of the framework, we give design guidelines for real-time systems with multiprocessor. Through the demonstration, the framework is found flexible and sufficiently general to apply to different kinds of real-time systems. By giving the practical design guidelines, we believe that our framework can benefit system designers to efficiently design scheduling.

References

[1] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Apr. 2009.

[2] C. Barrett, et al., "Satisfiability modulo theories," Handbook of Satisfiability, vol. 185. IOS Press, 2009.

[3] L.d. Moura and N. Bjorner, "Satisfiability Modulo Theories: An Appetizer," *Formal Methods: Foundations and Applications*, vol. 5902, pp. 23–26, 2009.

[4] S.S. Craciunas and R.S. Oliver, "SMT-based Task- and Network-level Static Schedule Generation for Time-Triggered Networked Systems," *Proc. 22th Int. Conf. on Real-Time Networks and Systems*, NY, USA, pp. 45–54, October, 2014.

[5] R.I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 5, pp. 35:1–35:44, Oct. 2011.

[6] L. Moura and N. Bjrner, "Z3: an efficient SMT solver," *Proc. 14th Int. Conf. on Tools and Algorithms for the Construction and Anal. of Syst.*, Budapest, Hungary, LNCS 4963, pp. 337–340, Springer-Verlag, 2008.

[7] B. Dutertre, "Yices 2.2," *Proc. 26th Int. Conf. on Comput. Aided Verification*, Vienna, Austria, LNCS 8559, pp. 737–744, Springer International Publishing, 2014.

[8] S.K. Baruah, et al., "A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[9] J. Anderson and A. Srinivasan, "Early-release fair scheduling," *Proc. of the Euromicro Conference on Real-Time Systems*, 2000.

[10] L. Zhang, et al., "Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems," *Proc. of ASP-DAC*, 2014.

[11] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," *Proc. of CODES*, 2002.

Manipulation of Atmospheric Data under Apache Hadoop

E. L. Rasch, A. S. Charão, and P. P. Barcelos
Department of Languages and Computing Systems
Federal University of Santa Maria, Santa Maria, RS, Brazil
{erasch, andrea, pitthan}@inf.ufsm.br

Abstract— *This paper explores the use of Hadoop to analyze large amounts of atmospheric data. First, we present a brief explanation of Apache Hadoop, along with an description of the data we handle, which are produced by satellites. Next, we present some steps of our algorithm to compute statistical measures, in particular standard deviation, and some problems raised on the accomplishment of the task as well as the solutions we found. We finalize the paper with some preliminar results and a perspective from the future of the investigation.*

Keywords: Hadoop, Big Data, MapReduce, Parallel Computing

1. Introduction

Nowadays we are experiencing the dawn of massive data sources, in a rhythm tending to rise with the popularization and spread of the internet. Having this in mind, it makes sense that the development of tools – powerful enough to treat and interact with such an amount of data (known as big data) – would rise along. One of these tools is the subject of the present study: Apache Hadoop. There are many others like it, with more or less the same features, like Storm and Spark [1], operating in different ways. While Spark and Storm stores the processed data on the primary memory, Hadoop uses the hard drive, both for the outputs of the Map and Reduce tasks, which makes it advantageous in cases where the amount of memory is less than the amount of data [2], [3]. Other differentials include the easy way of developing jobs, mainly using Java, and the possibility of batch processing.

Apache Hadoop makes use of the Map/Reduce model for processing huge amounts of data (in our case, a huge amount typically means some gigabytes), and operates by dividing the computer resources in a way that the tasks can be executed in parallel and a distributed way.

So, what we do in this article is to show and overview of the Map/Reduce paradigm immersed on Apache Hadoop, by analyzing climate data produced by satellites. To accomplish this task, we divided the study in the following sections: section 2, in which we describe the Apache Hadoop and the Map/Reduce model; section 3, in which we show how we used Apache Hadoop to manipulate the data, make some comparisons on different ways of executing jobs on pseudo-distributed mode, and expose some preliminary results based on the tests; and section 4, where we present some final considerations about this work.

2. Apache Hadoop

The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. The Apache Hadoop project consists several projects such as Hadoop Commom, Hadoop Distributed File System (HDFS), Hadoop Yarn and Hadoop MapReduce [2]. This paper focuses on Hadoop Distributed File System (HDFS) and Hadoop MapReduce. HDFS is a distributed file system that provides high-throughput access to application data, while MapReduce is a programming model for parallel processing of large data sets.

2.1 HDFS - Hadoop Distributed File System

HDFS is the primary distributed storage used by Hadoop applications. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data.

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that 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.

2.2 MapReduce

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the

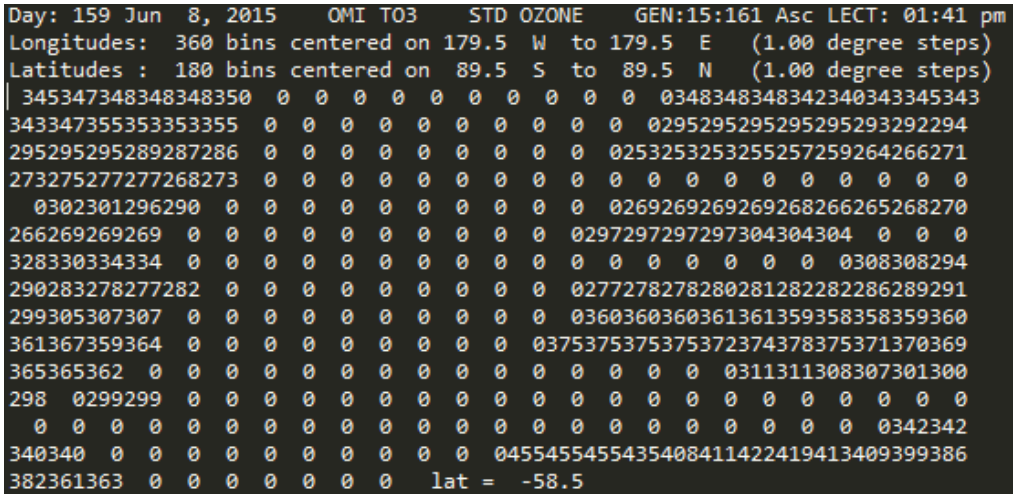


Fig. 1: Raw Data

failed tasks. The slaves execute the tasks as directed by the master.

The MapReduce programming model permits to express simple computing to be performed in parallel on a large amount of data [4]. The input data are divided in tuples (key, value), which are submitted to a mapping function (Map) that generates an intermediate set of pairs (key, value). The intermediate set of pairs are in turn submitted to a reduction function (Reduce), responsible for processing all values of the same key, resulting in a final set of pairs (key, value) [5]. The architecture of the Apache Hadoop platform is based on two major components: the Hadoop Distributed File System (HDFS) [3], which provides a distributed storage layer, and the Hadoop MapReduce [5], which provides a distributed implementation layer.

3. Using Hadoop to climate monitoring

This work describes the use of MapReduce to implement algorithms for manipulating data from satellites of climate monitoring, whose challenge is to deal with big data, extracting and managing it. The main goal of the work is to implement metrics such as average, maximum and minimum values and standard deviation. The algorithms implemented require the selection of the range time and the space to extract the pattern and/or the specific data of a zone.

3.1 Data Manipulation

The data to be manipulated are from the satellites[1] and they are in raw form as shown in Figure 1.

Figure 2 shows the data interpretation. As we see displayed in Figure 1, there are 180 blocks, with 15 lines each, totaling blocks with 360 measures. Each block represents a latitude, varying from 89.5 (-89.5) South to 89.5 (89.5) North. The longitudes are represented by each element from the blocks, varying from 179.5 West to 179.5 East. However, these files of data also contain the title columns, like date and time, location and so on, beyond the relevant information under which the metrics are done. Thus, it is needed to make a parser to find the relevant data. In our case, we set up certain intervals of map areas. Thereby, "lulat" (Left Upper Lattitude) , "lulon" (Left Upper Longitude) and "rllat" (Right Low Latitude), "rllon" (Right Low Longitude) match with latitude and longitude from top left point and latitude and longitude from low right point, respectively.

As for the space of time, we just need to make a simple check on the first row of the data file, and if it doesn't fit our lapse, we can ignore it.

3.2 Map and Reduce Tasks

To compute the statistics that require that we read the data only once (e. g. average), we simply use the Map/Reduce model, so that in the Map task we detach and join the values and calculate its occurrences; and in the Reduce task we compute the result.

However, measures like standard deviation require the knowledge of the average, demanding two or more iterations over the data. The Hadoop tasks, both Map as Reduce, read the data in a streaming form, with the values divided line by line, making it impossible to iterate through the data twice, as a read of a streaming data is also a "consuming" of the data. Therefore, we must choose between (a) make two iterations over the data (the first to calculate the average and store it on a variable, and another to calculate the standard deviation), or (b) to store the values to reuse (using, for example, an array or linked list, which can be iterated as many times as we need). With (a) we raise the processing time, while with (b) we have an intensive use of memory.

[1]The files with the data can be obtained on <https://ozoneaq.gsfc.nasa.gov/data/ozone/>, by choosing "OMI - Aura - Global Ozone Data".

```
public static ArrayList<Double> selectNumeros(String file, Double lulat, Double lulon,
            Double rllat, Double rllon) {
    ArrayList<String> blocks = blockGroup(file.split("\\r?\\n"));
    ArrayList<Double> lista = new ArrayList<Double>();
    for(int i=0; i<blocks.size(); i++){
        String[] aux = blocks.get(i).split("lat=");
        Double lat = Double.valueOf(aux[1]);
        System.out.println("Lat: "+lat);
        System.out.println("lulat: "+lulat);
        System.out.println("rllat: "+rllat);
        if(lat <= lulat && lat >= rllat){
            ArrayList<Double> lista_temp = quebraNumeros(aux[0]);
            System.out.println("lulon: "+(179.5+lulon));
            System.out.println("rllon: "+(179.5+rllon));
            for(int j= (int) (Math.round(rllon+179.5)); j <
                    (1 + (int) Math.round(lulon+179.5)); j++){
                lista.add(lista_temp.get(j));
            }
        }
    }
    return lista;
}
```

Fig. 2: Parser used to filter data

To obtain results considering all of the data, these solutions are enough. But, if we consider some data range, like the exclusively data between the lapse of two points which forms a square, knowing that the data is read line by line, we again came across to an analogous problem: how to know which values from a data line of a block we must consider, since the latitude is shown only in the final line of each block?

The solution could be, as in the previous problem, to store the date on the memory, or even to read it, mark the data we need with a parallel array, and iterate it again to obtain the meaning values.

But for this case we found a more reasonable solution: to reorganize the granularity from the Map task, shown in Figure 3. In a nutshell, we make an override of the InputFormatClass class, whose goal is to break the input files into lines and to forward the data to Map class again. So, it is possible to set up the cardinality of the data in which a Map task will operate, varying from only one character to an entire file. In our case, the files are completely read and then they are broken into blocks of lines, i.e, they suffer a split right on the line that contains the latitude information, allowing us to extract the latitude and determine whether the the block of data is meaningful or not. To do this, we must also update the job to use our custom class, as shown in Figure 4. Figure 5 and Figure 6 show the Map class and the Reduce class, respectively, that illustrate the operation described above.

```
public static ArrayList<String>
blockGroup(String[] lines){
    ArrayList<String> lista=new
        ArrayList<String>();
    String s = "";
    for(int i=0; i<lines.length; i++){
        if (!lines[i].substring(0,1)
        .matches("^[A-Z].*$")){
            s += lines[i];
            if( (1+i)%15 == 0 ){
                lista.add(s
                .replace(" ", ""));
                s = "";
            }
        }
    }
    return lista;
}
```

Fig. 3: Organizing the data reading

3.3 Tests

As an preliminary test, we run our algorithms on an pseudo-distributed environment. We consider the following data set: 2633 files of the raw satellite data (approximately 533 MB). The aim was to see how the memory consumption and the processing time was affected, under the perspectives

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Media");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(DoubleWritable.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(WholeFileInputForma.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    if(args.length > 4){
        conf.set("lulat",args[2]); // left-upper latitude
        conf.set("lulon",args[3]); // left-upper longitude
        conf.set("rllat",args[4]); // right-lower latitude
        conf.set("rllon",args[5]); // right-lower longitude
    }
    job.waitForCompletion(true);
}
```

Fig. 4: Using the "WholeFileInputForm" class

```
public static class Map extends Mapper<
        NullWritable, Text, Text, DoubleWritable> {
    public void map(NullWritable key,
            Text value, Context context) throws
            IOException, InterruptedException {
        String line = value.toString();
        Configuration conf=context.getConfiguration();
        ArrayList<Double> lista;
        if(conf.getDouble("lulat", 99999) == 99999){
            lista = Util.selectNumeros(line,
                conf.getDouble("lulat", 89.5),
                conf.getDouble("lulon", 179.5),
                conf.getDouble("rllat", -89.5),
                conf.getDouble("rllon", -179.5));
        }else{
            lista = Util.selectNumeros(new
                StringTokenizer(line));
        }
        for (int i = 0; i < lista.size(); i++){
            context.write(new Text("Número"),
                new DoubleWritable(lista.get(i)));
        }
    }
}
```

Fig. 5: Map class

```
public static class Reduce extends Reducer<Text,
DoubleWritable, Text, DoubleWritable> {
    public void reduce(Text key,
    Iterable<DoubleWritable> values,
    Context context) throws IOException,
    InterruptedException {
        double sum = 0.0d;
        int count = 0;
        for (DoubleWritable val : values) {
            sum += val.get();
            count++;
        }
        context.write(new Text("Média: "),
            new DoubleWritable(sum / count));
    }
}
```

Fig. 6: Reduce class

of the problems raised from the double read the values on the standard deviation measure. It's important to mention that this tests are merely experimental, as tests on real clusters are scheduled for the future of the investigation.

As for the tests with the different input formats, we conducted a simple test to obtain the average value, and reach an elapsed time of 115.6 seconds when using the custom WholeInputFileFormat to process the data (considering the entire job), while using the default TextInputFormat we obtained a time of 141.7 seconds. This values are the average of a series of 5 tests on each configuration. On the memory analysis, we reach a total committed heap usage of 4008866480128 bytes (average). As for the memory of the tests with entire file reading, the total committed heap usage totaled 5032410349568 bytes (average).

Also a test with a reduced data set was conducted (967 files with 196.1 MB) to obtain the variance. The time elapsed to accomplish the entire task, on the custom input format environment was 134.1 seconds. On the other hand, with the standard input and using a vector to store the values, the time was 140.4 seconds.

4. Conclusions

The present work arises the possibility of knowing and experiencing of a parallel and distributed tool to process mass data, and in our case, atmospheric data. In fact, it's a experimental work, which explores the usage of the Map/Reduce model in manipulation and treatment of huge masses of atmospheric data, produced by satellites used on climate researches.

Until now, all measurements were carried out on pseudo-distributed configurations and using relatively small amounts of data. It is necessary to extend this process to a real cluster. Thus, measurements can be carried out on the actual costs of each solution found, such as costs for memory and performance. In this sense, it is expected that at the end of the project, we can have good sense on the time and resources required to perform each operation, so that they can compare and indicate more clearly what are the best outputs for each context.

References

[1] K. Ballou, "Apache storm vs. apache spark," http://zdatainc.com/2014/09/apache-storm-apache-spark/.

[2] A. S. Foundation, "Welcome to hadoop!" available in: http://hadoop.apache.org.

[3] ——, "Hdfs architecture guide," available in: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[4] D. Wegener, M. Mock, D. Adranale, and S. Wrobel, "Toolkit-based high-performance data mining of large data on mapreduce clusters," *2013 IEEE 13th International Conference on Data Mining Workshops*, vol. 0, pp. 296–301, 2009.

[5] 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

A Hybrid Distributed Framework for SNP Selections

Pengfei Liu[1], **Shuai Li**[1], **Weiying Yi** [1] **and Kwong Sak Leung**[1]
[1]Department of Computer Science and Engeering
The Chinese University of Hong Kong, Hong Kong

Abstract— *With the development of next generation sequencing technology, researchers are able to obtain extremely high-dimensional data. However, only a fraction of the data is related to diseases and the computational time on processing the whole sequences is tremendous. Moreover, using the high-dimensional data directly will greatly reduce the accuracy of the machine learning and data mining algorithms. Single nucleotide polymorphism (SNP) selections is critical for addressing these problems in genome wide association study (GWAS). Typically, it needs days to perform SNP selections even though the relationship between SNPs and diseases is assumed to be linear. More time is needed when the relationship is nonlinear. In order to speed up the SNP selection processes, a CPU-GPU hybrid distributed framework (HDF) specifically for SNP selection algorithms is introduced in this paper. The HDF fully utilizes the computational power of machines. And the interfaces are also provided, which help researchers to extend their SNP selection algorithms into distributed version. The results demonstrate that the acceleration by HDF is about hundreds times on SNP selections with synthetic and real data, compared to single machine.*

Keywords: SNP selection, CPU, GPU, distributed

1. Introduction

Genome-wide association study (GWAS) is an analysis to identify which part of the human genomes are associated to a certain trait. In GWAS, statistical and computational analyses are applied to compare the DNA sequences of the controls (healthy samples) and the cases (patients with the specific genetic disease) in order to identify the related single nucleotide polymorphisms (SNPs) [1][2][3]. With the technology of next generation sequencing, researchers are able to obtain millions of SNPs in DNA sequences. However, only a small fraction of the SNPs are related to diseases, and the rest are irrelevant and regarded as noises. These noises will severely reduce the accuracy and reliability of the GWAS algorithms [4]. Hence, identifying the useful SNPs before analyzing their relationship with diseases is a critical issue in GWAS.

There are about 4 million SNPs in human DNA sequences, and many SNPs work in coordination to manifest a disease [5]. Analyzing such a high dimensional combinatorial relationship increases the computational complexity a lot. To speed up the process of SNP selection, parallel computing is adopted in this work.

Nowadays, most of the computers are equipped with both CPUs and GPUs. In order to maximize the utilization of the computing resources, a CPU-GPU hybrid distributed framework (HDF) specifically for SNP selections is proposed and implemented. The HDF provides interfaces that help researchers extend their SNP selection algorithms into distributed versions. To the authors' best knowledge, the proposed HDF is the first CPU-GPU based distributed system specifically designed for speeding up the SNP selection processes.

The HDF consists of three components:

- **Controller**
  The Controller decomposes the original computational mission from a user into many small tasks, and distributes them to the CPU clients and GPU clients described below. After receiving the progress report from the clients, the Controller merges the progresses and find a new task to distribute, which can be manipulated using the provided interfaces.
- **CPU client**
  The CPU clients use multi-thread architecture to handle the tasks distributed by the Controller and return the results back to the Controller after finishing the task.
- **GPU client**
  The GPU clients use Nvidia CUDA API to process the tasks distributed by the Controller and return the results back to the Controller after finishing the task.

To test the performance of the HDF, we implement an SNP selection algorithm ReliefF on the HDF using the provided interfaces. The experimental results show that the distributed version is about hundreds times faster than the a single thread CPU version.

The rest of the paper is organized as follows. Literature reviews on SNP selection algorithms and distributed systems are introduced in Section 3. Section 4 and 5 are the architecture design and the implementation of the HDF. Experiments are performed in Section 6 and the results are analyzed in Section 7. Section 8 is the conclusion and discussion.

2. Definition of SNP Selections

Each SNP in human genome is either an A-T pair or a C-G pair. For each SNP, the pair with higher probability is called the major allele, otherwise it is called the minor allele.

For each pair of chromosomes, there are three different kinds of SNP combinations, major-major, major-minor, and minor-minor.

Input: A dataset that contains the cases and the controls. Each sample stores an array of human SNP genotypes, which are encoded in Table 1.

Genotype	Value
missing	0
major allele–minor allele	1
major allele–minor allele	2
minor allele–minor allele	3

Table 1: SNPs encoding scheme

Output: The association weight between each SNP, or a set of SNPs, and the target disease.

3. Related Work

With the increasing dimension of data, feature (that is SNP in this paper) selection becomes more and more important in various research areas [6][7][8]. Generally speaking, there are three types of feature selection algorithms. *Filters* utilizes one or more statistical properties of each attribute, such as information entropy and t-test value, to identify useful features [9]. *Wrappers* evaluate the association between labels and groups of features. *Embedded methods* treat features selection as part of main algorithms.

Bacause SNPs may work in coordination to manifest a disease and the interactions between SNPs probably are nonlinear, it takes days to perform SNP selections [10].

The ReliefF [11] and its variations [12] [13][9] are widely used in SNP selections. It evaluates every SNP in the following steps. First, it compute the distance matrix among all the samples. Second, for each sample, ReliefF finds its nearest neighbor with the same phenotype (called nearest hit) and the nearest neighbor with different phenotype (called nearest miss). Third, for each SNP, its value in the selected sample is compared with the nearest hit and nearest miss. The weight of each SNP will be updated based on the comparison result.

Meanwhile, many parallel and distributed studies have been reported to accelerate the SNP selection algorithms [14]. Some of them are multi-core CPU based architecture [15], and some of them are GPU based architecture [16][17]. Although there are many distributed machine learning packages, such as the TensorFlow [18] from Google and the DMTK from Microsoft. There are no packages specially designed for SNP selections.

4. System Design

The CPU-GPU hybrid distributed framework (HDF) adopts a central control structure. In the proposed HDF, there is a master, called the Controller, which controls the work flow of the whole system. The Controller decomposes the computing mission, uploaded by a user, into many small tasks and distributes the small tasks to the CPU and GPU clients, where lots of CPU clients and GPU clients can be activated in this HDF. Those clients receive tasks from the Controller and return the computed results back. The architecture of the HDF is illustrated in Fig. 1.

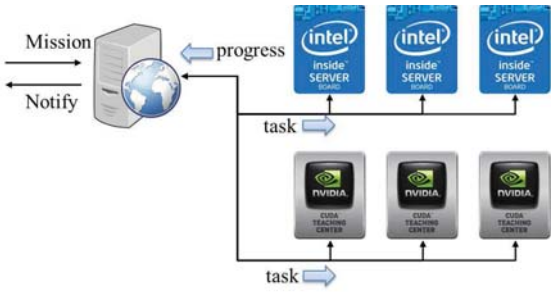


Fig. 1: The architecture of the CPU-GPU hybrid distributed framework.

4.1 Two levels acceleration

Our HDF uses two levels of acceleration to speed up the computing process. The first level is in the Controller-clients cooperation. The clients are independent from each other and they can work concurrently. By decomposing the mission and distributing the tasks in the Controller, and executing the tasks concurrently in the clients, the HDF achieves the first level of acceleration. The second level of acceleration is in the clients. CPU clients can use the multi-thread scheme to further speedup the execution. And the advantage in GPU clients lies in vector operations, which can greatly accelerate the process of SNP selections.

4.1.1 Acceleration by distributing

For most SNP selection algorithms, the evaluation of an individual SNP, or a set of SNPs, is independent from other SNPs, or other sets of SNPs. This enables the clients to work concurrently. So the Controller maintains two lists. The first list is all clients that are in connection with the Controller, and the second list is all small tasks which are decomposed by the mission. Each task keeps a record of its destination client. When there is a progress report from a client, the Controller searches this task from the task list and merges this progress with the progresses obtained so far. The Controller can distribute small tasks to different clients one by one or distribute randomly to make the clients have balanced load. The Controller keeps distributing and merging until all tasks are finished. In addition, if no progress report of a task after distributing is heard for too long time, the HDF will redistribute this task.

4.1.2 Acceleration by multi-threads

After receiving the tasks from the Controller, CPU and GPU clients can further speedup the SNP selection processes. Although they both use the multi-thread scheme, their architectures are different from each other.

CPU Clients When receiving a task from the Controller, each CPU client divides it into smaller ones and assigns a thread from its thread pool to handle the smaller tasks. Previous studies show that the performance of CPU multi-thread parallelization is highly related to CPU thread scheduling, and there is little increase when there are more threads than there are more cores in CPU [19]. (This is validated by our experiments in Section 7). In order to achieve an optimal trade-off between the acceleration and CPU thread switch cost, the CPU clients keep a thread pool with the number of threads a little bit more than the number of physical cores in the CPU.

GPU Client The hardware structure of GPU is different from that of CPU. GPU has many pipelines which can process multiple data under one instruction (SIMD). When doing vector operations, these pipelines can process different elements in the vectors simultaneously. This makes GPU much faster than CPU in many scientific computations involving vectors and matrices. To utilize the advantages of GPU, researchers need to transform the task data into matrices with rows standing for samples and columns representing SNPs, and rewrite the SNP selection algorithms using the computing API of GPU, such as CUDA.

4.2 Scheduling

The main challenge for the distributed system is the communication and synchronization among all the machines. In the HDF, the Controller is responsible for data splitting, distributing and the synchronization of all the computing clients. When researchers use this HDF to develop the distributed version of their SNP selection algorithms, they can use the defaulted interfaces of the HDF to handle these scheduling problems, or that they can define their own scheduling methods.

4.2.1 Splitting scheme

One important difference of bioinformatics data from data of other fields is that there are extreme high number of features (10^6) but relatively low number of samples(10^3). To split this imbalanced data, we provide four defaulted interfaces, which are features oriented and samples oriented splitting, with or without overlapping. Users can use any one of them, or define their own splitting methods based on their algorithm.

There are some cases that the users want to define their own splitting methods. For example, the processing power of CPU and GPU clients are imbalanced, and users may want to distribute them with different kinds of tasks. Note that some algorithms may need a special method for splitting and scheduling. For instance, VLSReliefF [12] samples features with replacement.

4.2.2 Distributing scheme

Since the acceleration comparisons of GPU and CPU vary on different algorithms, our HDF provides several distributing interfaces to optimize the performance. It can distribute tasks to CPU or GPU clients in priority, where it will not consider the other type of clients unless all the same types clients have been used. It can just distribute to GPU clients only or CPU clients only. It can also choose any of the two types of clients randomly. The interfaces are also provided for users to define their own methods of distributing.

4.2.3 Synchronization

By defining three states for the tasks, *Unprocessed*, *Waiting* (sent but without progress reported) and *Finished*, the Controller synchronize the HDF in the following steps. First, when a task is created, its state is set to be *Unprocessed*, and it is assigned with a unique id to identify from other tasks. Second, when a task is sent to a client, the task will remember the ip address of the client and update its state to be *Waiting*. Third, when the Controller receives a progress report from a client, which contains the id of the corresponding task and the ip address of the client, the Controller will check whether it is a legal progress (by comparing the task id and the ip address). If legal, the Controller merges this progress result with others' (by task id), and updates the task state to be *Finished*, and then find an *Unprocessed* task to distribute. For tasks which we haven't received their progress reports for a long time, the Controller will assume there is something wrong and will redistribute the task. When all tasks in the list have be updated to *Finished*, the Controller will notify the result to the users.

5. Implementation

As illustrated in Fig. 2, the proposed HDF is divided into three layers, the network layer, the middle layer, and the algorithm layer. Each layer focuses on its own target and provides service to its upper layer.

5.1 Network layer

Network layer handles the packages transmitting and receiving between the Controller and clients, and focus on providing stable communication services to its upper layer, the middle layer. The following kinds of packages are defined in the network layer.

- **Register package**, which is sent from the client to the Controller, and contains the IP address and the hostname of the client. The Controller will add the

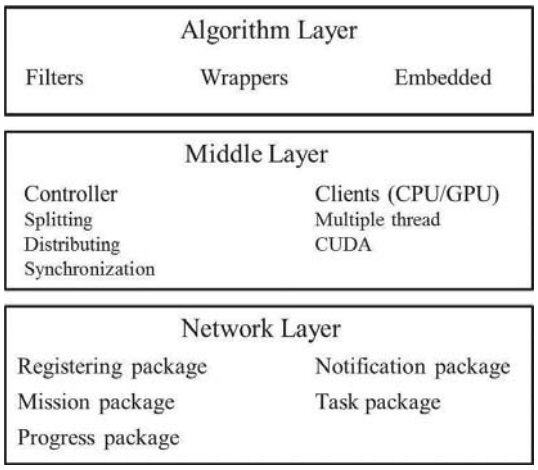


Fig. 2: Three-layer structure

socket ID to the package after receiving it from the client.

- **Mission package**, which is sent from the user to the Controller, and contains the location of the input file and the name of the algorithm.
- **Task package**, which is sent from the Controller to the client, and contains the location of the input file, the name of the algorithm, the task ID, the mission ID that the task belongs to, and a hash table that contains a subset of SNPs in the input file. The task package contains the file path instead of the content of the input file to reduce the communication overhead for the Controller.
- **Progress package**, which is sent from the client to the Controller, and contains the task ID, the mission ID that the task belongs to, and an array of SNP scores. This array of SNP scores is associated with the hash table in the task package, and the Controller will merge it with score arrays of other progress packages.
- **Notification package**, which is sent from the Controller to the user, and contains the location of the result file.

These packages are structured information. The network layer provides methods to serialize them into streams and to deserialize them from streams for transmitting and receiving.

5.2 Middle layer

The work flow of the proposed HDF is based on the event driven model. The Controller and clients start to work after some packages arrived. For the Controller, it may receive the following three packages:

a) Registration from clients: After receiving register package from the client, the Controller checks whether this client is on the clients list based on the ip address. If it is a new client, its IP address will be added into the clients list, which are maintained by the Controller. If not, the client's information will be updated.

b) Mission from user: After receiving mission package from the user, the Controller looks for the algorithm specified in the mission task, and then splits and distributes the data using the interfaces specified by the user. Different interfaces are provided to handle the splitting and distributing processes.

c) Progress from clients: The progress package's response from the Controller has been specified in Section 4.2.3.

For clients, it can only receive the task package from the Controller.

d) Task from the Controller: After receiving task package from the Controller, the client reads the data file from the file path provided by the task package. Then the indicated algorithm is called to handle the subset of data specified in the hash table stored in the task package. When the task is completed and a progress report to the Controller is generated, the client will copy the task ID of the task package to help the Controller synchronize.

5.3 Algorithm layer

Enabled by the network layer and middle layer, all kinds of SNP selection algorithms (described in Section 3) that evaluate a subset of SNPs independently from the rest of SNPs can be extended to distributed versions using the proposed HDF. An algorithm manager is also implemented in the proposed HDF to manage the algorithms when multiple SNP selection algorithms are adopted into the HDF.

6. Experimental Designs

6.1 ReleifF algorithm

According to the proposed system design in Section 4, the HDF has been implemented using C++ 11. Also, the ReliefF algorithm has been integrated into the HDF in order to compare its performance with that of a single PC. The pseudo-code of the ReliefF algorithm is illustrated in Fig. 3.

6.2 Hardware configurations

In the following experiments, 16 CPU clients (Debian 6.06 workstations with Intel Xeon E5 processors) are adopted. Among these 16 CPU clients, client 1 to client 4 are single-core machines; client 5 to client 8 are 4-core machines; client 9 to client 12 are 5-core machines; and client 13 to client 16 are 16-core machines. 6 GPU clients that equipped with NVIDIA Kepler (GK110GL) display board are also adopted. The memory capacity of each one of these 22 machines is larger than 100 GB.

Input: for each training instance a vector of attribute values and the class value
Output: the vector W of estimations of the qualities of attributes

1. set all weights $W[A] := 0.0$;
2. **for** i := 1 **to** m **do begin**
3. randomly select an instance R_i;
4. find nearest hit H and nearest miss M;
5. **for** $A := 1$ **to** a **do**
6. $W[A] := W[A] - \text{diff}(A, R_i, H)/m + \text{diff}(A, R_i, M)/m$;
7. **end**;

Fig. 3: Pseudocode of ReliefF algorithm.

	Group 1	Group 2
Heritibiltiy (folders)	0.1, 0.2, 0.3, 0.4	0.1, 0.2, 0.3, 0.4
Dataset in each folder	100	100
Samples in each dataset	1000 (500:500)	1000 (500:500)
Features(SNPs) in each dataset	1000	10000
Total Size	700MB	9.8 GB

Table 2: Details for the synthetic data.

6.3 Data sources

a) Synthetic data: The synthetic data used in the experiments are generated by GAMETES[20], which is a specified tool with high accuracy GWAS data generation ability. Two groups of synthetic data with different sizes are generated. The first group contains datasets with 1k features, and the second group contains datasets with 10k features. For each group, there are 4 folders representing different heritability (0.1, 0.2, 0.3, and 0.4). Each of these folders contains 100 files in it and each file is an independent dataset containing 1000 samples (500 positive samples and 500 negative samples). Detailed information of the synthetic data is listed in Table 2

b) Real data: A real dataset *phs000019v1* for psoriasis study from the Genotypes and Phenotypes datebase(dbGaP) is used in the experiments. This dataset contains 1659 samples, in which there are 950 cases, and 709 controls. For each sample, there are 448955 SNPs. The size of the *phs000019v1* is 2.1GB.

In the experiments, we design 3 tasks for the proposed HDF. task No.1 is to run the ReliefF algorithm on the synthetic dataset with 1k SNps. task No.2 is to run the ReliefF algorithm on the synthetic dataset with 10k SNPs. And task No.3 is to run the ReliefF algorithm on the real dataset. For each task, the speedup achieved by the proposed HDF, compared to one single-thread PC, is recorded.

7. Results

Previous experiments have indicated that single-thread CPU client can accomplish tasks No.1-3 in 10825s (3.1h), 107982s (32h), and 29810s(8.3h) respectively. In the following experiments, these three results will be considered as benchmarks and all the accelerations are calculated with respect to them.

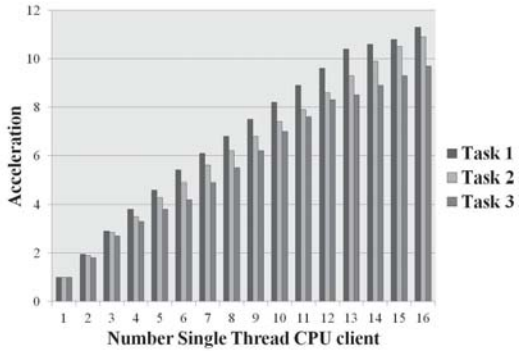


Fig. 4: Acceleration achieved by multiple CPU clients.

In the following experiments, the two levels of acceleration techniques of the HDF will be tested, and the results illustrate that significant acceleration rate can be achieved for both levels.

7.1 First level acceleration

a) Acceleration of multiple CPU clients: In this experiment, the number of CPU clients is increasing from 1 to 16, where each client has only one single-thread. The result is shown in Fig. 4.

In Fig. 4, the horizontal axis is the number of adopted CPU clients, and the vertical axis is the acceleration achieved compared to the benchmarks described in Section 7.1. A linear relationship with coefficient < 1 between the number of clients and the acceleration can be obtained in this figure. When the number of clients adopted is small (< 5), the linear coefficient is above 0.9. It drops to around 0.7 when the number of clients adopted increases. The reason that task No.1 achieves better acceleration than tasks No.2-3 is that the size of task No.1's dataset is much smaller than that of the other two. Such a small size dataset makes task No.1 suffers less from the communication overhead. The accelerations achieved by the task No.3 is the smallest because the dataset is stored in one single file, while the datasets of tasks No.1-2 are stored in many small files. Hence, when performing tasks No.1-2, different clients read different small size files simultaneously. However, when performing task No.3, different clients read the large size single file independently. Such a read operation dramatically increases the reading time of the hard disk, and further encumbers the acceleration.

b) Acceleration of multiple GPU clients: In this experiment, the number of GPU clients is increasing from 1 to 6. The result is shown in Fig. 5.

In Fig. 5, the horizontal axis is the number of GPU clients adopted, and the vertical axis is the achieved acceleration rate, compared to the benchmarks described in Section 7.1.

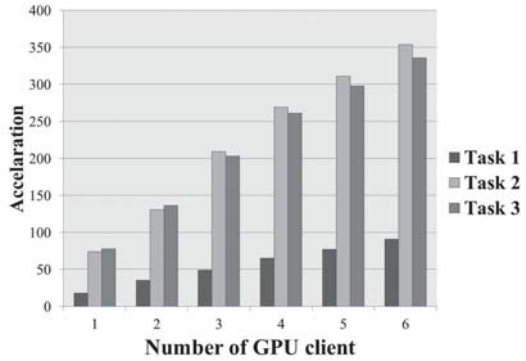


Fig. 5: Acceleration achieved by multiple GPU clients.

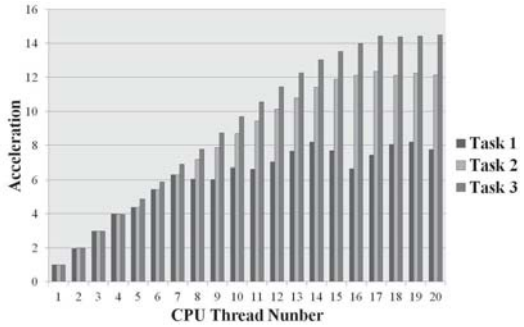


Fig. 6: Acceleration achieved by CPU multi-thread processing

The speedup by GPU is much better than that by CPU. The performance differences among tasks No.1-3 are mainly caused by the interchanging property of GPU computing states. There are two computing states inside one GPU computation process, the working state and the sleeping state. Whenever a GPU operation arrives, the computing state changes from the sleeping state to the working state, and changes back to the sleeping state after accomplishing the tasks. A specific period of time is needed for interchanging states. For task No.1, frequent interchanging of states occurred due to the small size of the dataset files. For tasks No.2-3, the overhead on interchanging states is much smaller than that of task No.1 because the size of the dataset files is larger. The difference between tasks No.2-3 is caused by how the dataset is stored (multiple small-size files or single large-size file).

7.2 Second level acceleration

The performance of the second level acceleration is tested by conducting experiments on a single CPU client, and a single GPU client respectively.

a) Acceleration of CPU multi-thread processing: In this experiment, there are 16 physical cores inside the CPU client. The number of threads is increasing from 1 to 20, and the result is shown in Fig. 6

In Fig 6, all the three tasks can be speeded up by the multi-thread technique. However, scheduling the threads takes time. When the size of the processed dataset by each thread decreases, the thread scheduling overhead becomes important. For task No.1, there are only 1000 features for each sample. When the number of threads increases, each thread will process about one hundred SNPs. The time spent on scheduling threads is then more significant than the time spent on processing the SNPs. For tasks No.2-3, the increasing tendency of acceleration stops when the number of threads exceeds the number of physical cores inside the CPU. Performance of task No.3 is typically better than that of task No.2 because the size of dataset processed by each thread is larger and the time spent on thread scheduling is relatively insignificant.

b) Acceleration of GPU parallel processing: GPU parallel processing is a little bit more complex than CPU multi-threading. In CUDA API, each GPU process contains several blocks, and each block contains many threads. Despite the accessible global memory for all threads, each block also has its own faster memory. Users can modify the memory distribution by adjusting the ratio between the number of blocks and the number of threads in each block. In the experiment, the number of threads in each block is increasing from 1 to 1000. The results illustrate constant acceleration for all the three tasks, which indicates that there may be some optimization inside CUDA.

7.3 Acceleration of the proposed CPU-GPU hybrid distributed framework

In this subsection, the CPU clients and GPU clients have been integrated into one hybrid system called CPU-GPU hybrid distributed framework (CPU-GPU HDF) and its performance on SNP selections has also been tested. For the CPU clients, 20 threads are created in each client. For the GPU clients, the number of blocks in GPU doesn't matter as discussed in Section 7.2. The results are shown in Fig. 7.

In Fig. 7, the acceleration increases when the number of CPU and GPU clients increase. The acceleration power of CPU clients is almost linear to the number of physical cores in it, and the acceleration power of GPU client is algorithm dependent. For ReliefF algorithm, GPU client can achieve 30 to 70 times speedup compared to benchmarks. When there are 16 multi-thread CPU clients and 6 GPU clients, the acceleration for task No.1-3 are 151, 423, 419 respectively.

8. Conclusion

SNP selections is critical in genome wide association study (GWASs). However, SNP selections is really time consuming, so the acceleration on performance is essential. In

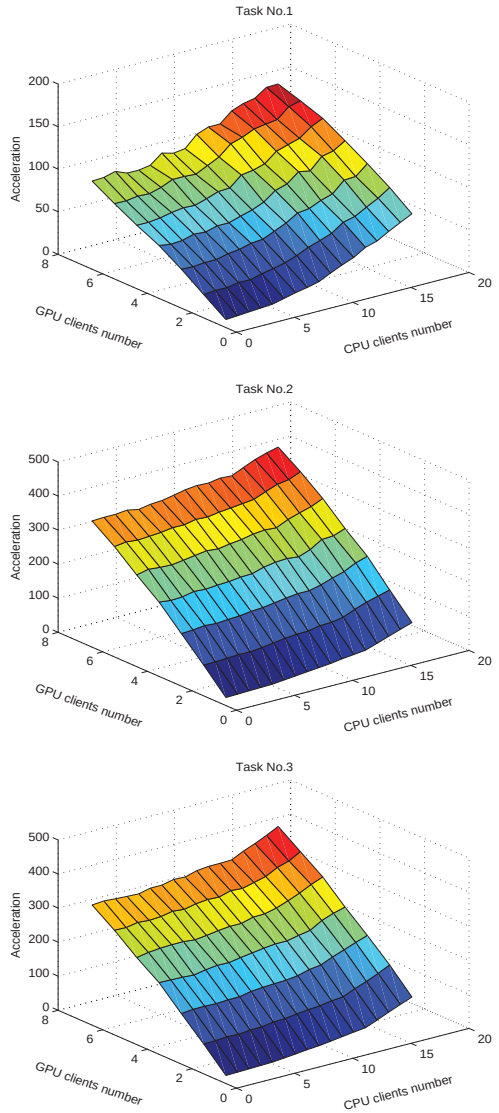


Fig. 7: Acceleration achieved by the HDF.

this paper, a CPU-GPU hybrid distributed framework specifically for SNP selection algorithm is proposed, designed, implemented, and tested. The HDF provides interfaces for data splitting and communication synchronization. It can support many different SNP selection algorithms using the interfaces provided. In the experiments, the algorithm of ReliefF SNP selection has been integrated into the HDF to test its performance on different datasets. The experiments indicate that the proposed HDF is able to achieve significant acceleration on different datasets.

References

[1] P. M. Visscher, M. A. Brown, M. I. McCarthy, and J. Yang, "Five years of gwas discovery," *The American Journal of Human Genetics*, vol. 90, no. 1, pp. 7–24, 2012.

[2] M. A. Rivas, M. Beaudoin, A. Gardet, C. Stevens, Y. Sharma, C. K. Zhang, G. Boucher, S. Ripke, D. Ellinghaus, N. Burtt *et al.*, "Deep resequencing of gwas loci identifies independent rare variants associated with inflammatory bowel disease," *Nature genetics*, vol. 43, no. 11, pp. 1066–1073, 2011.

[3] D. L. Nicolae, E. Gamazon, W. Zhang, S. Duan, M. E. Dolan, and N. J. Cox, "Trait-associated snps are more likely to be eqtls: annotation to enhance discovery from gwas," *PLoS Genet*, vol. 6, no. 4, p. e1000888, 2010.

[4] J. Bedő, D. Rawlinson, B. Goudey, and C. S. Ong, "Stability of bivariate gwas biomarker detection," *PloS one*, vol. 9, no. 4, p. e93319, 2014.

[5] A. Dahl, V. Iotchkova, A. Baud, Å. Johansson, U. Gyllensten, N. Soranzo, R. Mott, A. Kranis, and J. Marchini, "A multiple-phenotype imputation method for genetic studies," *Nature genetics*, 2016.

[6] M. Dash and H. Liu, "Feature selection for classification," *Intelligent data analysis*, vol. 1, no. 3, pp. 131–156, 1997.

[7] ——, "Feature selection for clustering," in *Knowledge Discovery and Data Mining. Current Issues and New Applications*. Springer, 2000, pp. 110–121.

[8] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[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] K.-Y. Lee, P. Liu, K.-S. Leung, and M.-H. Wong, "Very large scale relieff algorithm on gpu for genome-wide association study," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015, p. 78.

[11] 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.

[12] 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.

[13] 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.

[14] M. Cadzow, J. Boocock, H. T. Nguyen, P. Wilcox, T. R. Merriman, and M. A. Black, "A bioinformatics workflow for detecting signatures of selection in genomic data," *Frontiers in genetics*, vol. 5, 2014.

[15] X. Zheng, D. Levine, J. Shen, S. M. Gogarten, C. Laurie, and B. S. Weir, "A high-performance computing toolset for relatedness and principal component analysis of snp data," *Bioinformatics*, vol. 28, no. 24, pp. 3326–3328, 2012.

[16] X. Hu, Q. Liu, Z. Zhang, Z. Li, S. Wang, L. He, and Y. Shi, "Shesisepi, a gpu-enhanced genome-wide snp-snp interaction scanning algorithm, efficiently reveals the risk genetic epistasis in bipolar disorder," *Cell research*, vol. 20, no. 7, pp. 854–857, 2010.

[17] L. S. Yung, C. Yang, X. Wan, and W. Yu, "Gboost: a gpu-based tool for detecting gene–gene interactions in genome–wide case control studies," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, 2011.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems, 2015," *Software available from tensorflow. org*.

[19] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 2.

[20] 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, p. 1, 2012.

Toward a Smart Adaptive Scheduling using Lua Programming Language

Felipe Santos da Silva and Marcia Pasin
Departamento de Linguagens e Sistemas de Computação
Universidade Federal de Santa Maria, Santa Maria, Rio Grande do Sul, Brazil

Abstract— *Process scheduling can define the performance of a computer system. There are different algorithms to deal with it (such as FIFO, shortest remaining time, and round-robin). However, due to the dynamic nature of distributed applications, the choice of a scheduling policy is not trivial. Indeed, choosing an appropriate policy requires take into account the scenario and current demand. If the scenario and demand change, an adaptation may be necessary. Thus, in this work we describe an approach based on the application of scheduling policies and adaptation to deal with changes in demand. This is a step in the direction of smart scheduling. The adaptation of the scheduler behavior occurs through the use of heuristics. It trades at a random policy. If the new policy gets better results than the former, execution continues with the new policy until a new evaluation occurs. Experiments were conducted using Lua programming language due to the suitable support to the development of prototype in distributed systems.*

Keywords: scheduling, adaptation, client-server architecture.

1. Introduction

Distributed applications are subject to heterogeneous demand, peak load and idleness. This scenario leads to the need to manage these occurrences, which is a complex activity. Current computing infrastructure solutions are not designed to automatically adapt to dynamic conditions and complexity of the system. Typically, existing solutions require manual operation, leaving the system to be subjected to unavailability and inappropriate response time. There is a shortage of self-adaptive computational infrastructures, which are able to withstand changes in demand and are able to deal with different adversities simultaneously.

More specifically, current distributed applications need more adaptable solutions that allow facilities to the evolution such as tuning of of scheduling policy. Adaptive solutions for scheduling with a focus on adaptation are better suited to the dynamic demands of today's distributed applications and consider requirements such as overload servers [10], information contained in the client request [7], such as response time, or even priority of client processes [4].

Thus, in paper we propose a scheduler processes that implementing different scheduling policies, flowing dynamic behavior and that seeks to improve the system's efficiency. This is an step toward the building of a smart scheduler, suitable to distributed systems with dynamic load.

Scheduling policies executed by the scheduler are configurable and include FIFO (first in, first out), SJF (the shortest job first) and lottery scheduling [9]. Basically, the scheduler operates in a continuous looping and the behavior modification, i.e. the switch from a policy to another, occurs through the use heuristics. The system continuously switchs from a policy to a new policy in a random fashion. If the new policy achieves better results, execution continues with the new policy until a new assessment.

To evaluate the our proposed solution, we implemented a prototype using Lua programming language [1] [5]. Lua is a powerful language to build prototypes.

Both scheduling is recurrent research subject. However, there is still no consensus among authors on the use of scheduling policies. Choosing the suitable policy depends on application characteristics, demand and the computer system issues.

In this sense, this work investigates the use of process scheduling policies in heterogeneous scenarios, trying to evaluate them as to optimize response time to client request. As it is difficult to choose a single policy that offers excellent results in all possible combinations of hardware and software, an adaptive mechanism is an interesting idea, as the scheduling process adapts as needed by the computer system.

Through knowledge of the application needs to be performed, the current loads and the behavior of different scheduling policies in accordance with given scenario may be performed choosing the best scheduling policy at that time or at least can be used as a solution that gives the performance values that are close to the best solution.

As a restriction, in this work we assumed that the system never fails and has no answer delays, for purposes of estimating the best policy given the target scenario. It is expected that a policy with a good performance in the ideal situation also is the actual situation. Policies that do not show in this state will be out of choice for the adaptive scheduler.

This paper is organized as following. Section 2 presents the related works. Section 3 presents the design of the adaptive scheduler, describing the overall architecture and

[1] https://www.lua.org

adaptation policies. Section 4 shows the experimental evaluation of the proposed adaptive scheduler and indicates how adaptation can improve the performance of a computer system. Finally, section 5 presents the conclusions and final considerations.

2. Related works

Adaptation in computer systems is not a new subject. It has been studied in the context of centralized and distributed architectures, aiming at the implementation of the concept of autonomic systems [6].

Even before the development of autonomic computing, Cervieri [3] proposes the use of adaptive scheduling for real-time CORBA applications. The period in which each tasks are executed is controlled, varying within a predetermined range. The main objective of the approach is to reduce the average delay of running tasks.

In Ewing & Menascé [4], the scheduling is handled the load balancing level for multiple servers in an auction system using web pages. The goal of this approach is to prioritize groups of users who are running the largest amounts of bids. Scaling is performed by classification of requests, allocating priority requests to a cluster with more processing availability.

As a common feature with our proposed work, the work of Ewing & Menascé also uses the response time to user requests' as metric adaptation in the computer system. In contrast, the adaptive scheduler here proposed focuses on appropriate policy choice not on the choice of suitable machine as Ewing & Menascé do.

Badonnel & Burgess [1] use metrics of time and abandonment of requests to perform calibrations in a distributed computer system. Response time measurements of customer orders are collected. Statistics are generated with the number of applications being executed, and the average response time is compared using pull-based and push-based methods.

In contrast to our work, in Badonnel & Burgess, the focus is on communication between clients and the scheduling module. In the scheduler, and each server exists a finite queue of requests that may suffer dropouts when the queue is full. A statistic about the number of abandoned order is used as a basis for decision between the two methods (push or pull). It is used FCFS (first come first serve) policy or FIFO for the distribution of tasks.

In Bouchenak *et al.* [2], sensors are used to check the CPU usage in order to manage a system made up of components. This management is able to deploy distributed applications to get a reconfiguration autonomously when required. Adaptation is executed by a centralized module that collects information about CPU in local components. If the setting is not adequate, which is verified by defining a threshold, the system triggers a mechanism that adapts the system configuration. In contrast, in this work the adjustment is applied to the scheduling of processes and not in load balancing or neither in the system configuration. The approach adopted here has the focus on the scheduling policies themselves.

3. Architecture overview

Our adaptive scheduling service follows a client-server architecture. This architecture is composed of n clients and a server. Basically, clients request the execution of tasks to the server. The server receives the request and uses a queue to store them. The schedule executes a scheduling policy (FIFO, lottery or SJF).

Continually, the server switches from a policy to another looking for better respond time results. Thus, the server service was fractioned in three parts: task executor itself (i.e. the scheduler), adapter and queue manager. An schema with this architecture is shown in Figure 1.

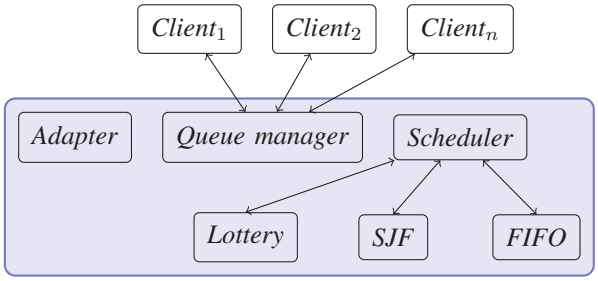


Fig. 1: Overview of the client-server architecture.

In the following, we describe in details the client-server architecture. Initially, we focus on the client side, then we detailed the server architecture and also the scheduling algorithms. We describe, in particular, the adjustment process based on heuristics. For short, we also given implementation details in the algorithms.

3.1 Client side

A client interacts with the queue manager on the server side. Clients request tasks to the server by running the code outlined by Algorithm 1.

```
1: function runs_the_client (list_of_tasks, server_id)
2: for  i ⇐ 1 until size (list_of_tasks) do
3:     sent ⇐ false
4:     while not sent  do
5:         sent ⇐ send (server_id,
6:         {from ⇐ self(), task ⇐ list_of_tasks[i]})
7:     end while
8:     msg ⇐ receive()
9: end for
```

Algorithm 1: Client processing.

Basically, in Algorithm 1, there is a list of tasks, with size given by $size(list\_of\_tasks)$ that must be executed by a server, specified by the variable $server\_id$. Notice

that, if there is more than a task to be executed in the $list\_of\_tasks$, using the Lua implementation, we can implement the $send$ using the function $concurrent.spawn()$ to emulate $size(list\_of\_tasks)$ concurrent clients. This is given by lines 5-6.

The variable msg, in line 8, receives the confirmation of the execution of the task by the server. While the client process does not receive this confirmation, the client is blocked waiting.

3.2 Server side

As we previously mentioned, the server service has three parts: task executor itself, adapter and queue manager. The tasks' executor executes a scheduling policy. The adapter switches from a scheduling policy to another. The adapter also stores information about the collected response time and sent by clients, which will be used for adaptation. The queue manager performs the communication with the clients.

3.2.1 Adaptation

Basically, the adapter operates through the use heuristics. Exchange the policies is a random approach. If the new policy gives better results, execution continues with the new policy until a new evaluation within the specified time interval.

The client sends, in a interval of time T_c, information about the response time to the adaptive service. The server executes the following tasks, in a interval T_s of time:

1) collects clients' response time information,
2) raffles new policy,
3) runs new policy,
4) collects the new clients' response time information,
5) if the new response times are better than the previous one, then follows by applying the new policy; otherwise, then returns to step 2.

3.2.2 Queue manager

The queue manager interacts with the clients. With regard to the queue manager, the following steps are executed:

- each client process sends a work requests to the server;
- the server, through the queue manager, receives these requests and puts them in a queue. If the queue is full, the client remains in a continuous loop until it reaches blank entry in the queue.
- the task manager, when idle, is constantly calling for a new task to the queue manager,
- when the queue is not empty, the queue manager staggers the arranged processes in the queue and chooses which process should be sent to the tasks' performer,
- the scheduler, when finishing a task, returns to the manager and to the client that the work has been completed.

To implement the queue on the server, we used the mailbox feature offered by Lua programming language. The code is outlined by Algorithm 2.

O Algorithm 2 shows the set of steps executed by the server. Again, we give some Lua details. Initially, the queue is empty (line 2) and the process to perform the task is started (line 3).

Continually, the server receives new messages. If the new message is from the executor process (line 6) and the process queue is not empty (line 7), the scheduler selects a task from the queue (line 8), according to a specific policy, and sends it to the executor (line 9).

If the queue is empty, the queue manager sends to the performer a message indicating that there is no task to be performed at the time (line 11). If the received message is not from the task manager, a client is thus the task requested, and the request for executing the task is inserted in the queue.

```
1:  function server()
2:  queue ⇐ {}
3:  executor_id=spawn (executor, self())
4:  while true do
5:     msg ⇐ receive()
6:     if msg.from=executor then
7:        if queue>0 then
8:           chosen ⇐ choose (queue)
9:           send (executor_id, chosen)
10:       else
11:          send (executor_id, {client=nil, task=0})
12:       end if
13:    else
14:       insert (queue,{client=msg.from, task=msg.task})
15:    end if
16: end while
```

Algorithm 2: Server processing.

3.2.3 Scheduler

The scheduler executes a scheduling policy. The scheduling policies are responsible for defining what should be done to occur the scheduling. The literature describes different policies for scheduling processes. In the scope of this paper, we studied the following policies: SJF, FIFO and lottery algorithm. In sequence, these policies will be briefly described.

SJF (shortest job first). Using this policy, the scheduler performs tasks according to the size of each task. The shorter task has higher priority over the longer tasks. This policy is especially suitable for batch jobs, where the runtime is known in advance. This policy is, therefore, interesting to computing systems with requirements of efficient response time.

As disadvantage, the information about the duration of each task needs to be available beforehand. However, in many applications, such as file transfer between different machines, these values can be easily estimated.

FIFO (first in, first out). In this approach, the first to enter in a queue is the first to leave the queue. This is one of the simplest policies to be implemented, because it does not require special approaches such as previously to compute the information on the duration of each available task.

Basically, this policy put the tasks to be performed in a queue, following the order in which the requests were delivered to the server. The first task that enters the queue is the first task to be performed.

In this policy, for each execution, the order of the tasks may change substantially, because the task execution order is enforced by the delivery order of clients' request messages on the server (that is, no matter the shipping order). This order impacts on the performance as well as in the response time to clients.

The lottery scheduling. This algorithm chooses randomly which is the next task to be executed distributing lottery tickets to the tasks. When the scheduler is available for use, a new ticket is drawn. According to [8], this algorithm may prioritize a process p_i allocating more tickets for this process.

3.3 Metrics' definition

In the scope of scheduling algorithms, different metrics can be used to quantify how much a policy is better than another. Among the different metrics reported in the literature, Tanenbaum [8] stands out:

- fairness to ensure that each process receives a fair share in the allocation of CPU,
- efficiency to keep the CPU busy 100% of the time,
- response time to minimize the response time for interactive users,
- turnaround to minimize the time that batch users must wait for output,
- throughput to maximize the number of jobs processed per hour.

In this work, we choose the response time as metric because we are interested in build a high performance system in terms of clients' feedback. We are looking into the direction of a smart, adaptive scheduler. Policies are evaluated periodically and the scheduler selects the policy with the best response time for a given scenario.

4. Experimental evaluation

4.1 System setup

The experiments were performed on a machine using the Debian 7.2 operating system (wheezy) 32-bit with the Linux Kernel 3.2.0-4-686-pae, and GNOME 3.4.2 desktop. The machine has memory RAM 3.7GiB and four-processor Intel®Core™i3 CPU 540 @ 3.07GH. The version of Lua programming language we used is 5.1, with the system $LuaRocks$ to install the library $concurrentLua$.

In total, we used only 420 lines of code, which include the implementation of the client-server architecture, the testing scenarios and auxiliary programs to assess results of $logs$ generated by the experiments. This only was possible because Lua is a really powerful language to build prototypes.

4.2 Defining the target-scenarios and comparing policies

For the experimental evaluation, we defined four target-scenarios, with different demands. The scenarios and results we achieved are described below.

To measure the server's response time to clients' requests, and to enable evaluate metrics seeking adaptation, we used a clock that increases the time according to the task being performed by the server.

To provide more accurate measurements, we carried out three executions in each scenario and we plot the average of these values in the graphics.

4.2.1 Scenario 1: homogeneous and short-term tasks

In scenario 1, the server receives requests from clients that issue identical tasks. Each client, using the $concurrentLua$ library, issues 100 tasks simultaneously. The tasks are homogeneous and have short duration. Because tasks are short, the demand in the server service is high. The duration of each task is 10 units of assessment measure.

The results obtained by performing experiments using this scenario, and policies lottery, FIFO and SJF are shown in Figure 2. In this first scenario, lottery policy got better response time than FIFO and SJF, which in turn showed similar behavior.

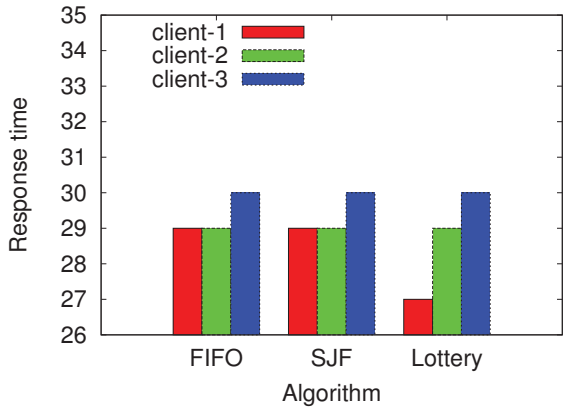


Fig. 2: Performance comparison of policies in scenario 1, using homogeneous and short-term tasks.

In fact, one may claim that the results we got are almost the same, and the performance of the algorithms are equivalent in this case.

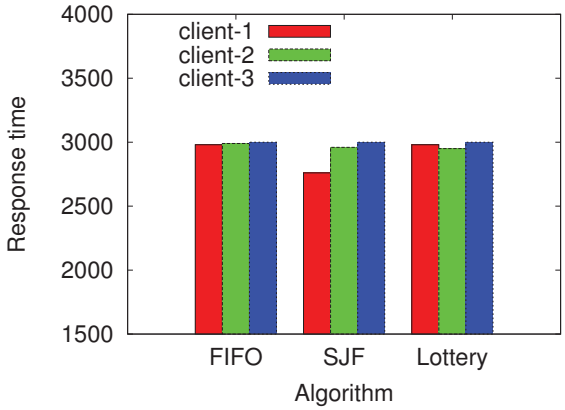


Fig. 3: Performance comparison of policies in scenario 2, using homogeneous and long-term tasks.

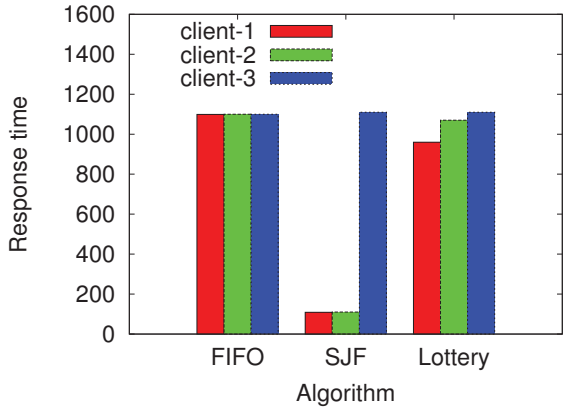


Fig. 4: Performance comparison of policies in scenario 3, using heterogeneous tasks.

4.2.2 Scenario 2: homogeneous and long-term tasks

In scenario 2, the server receives, again, requests from clients with identical tasks. Each client issues concurrently 100 tasks. Tasks are homogeneous but long-lasting to contrast with scenario 1. As the tasks are long-lasting, in the server a new task is performed just after completion of the previous task. In the scenario 2, the duration of each task is $1,000$ units of assessment measure.

In the experiment, this second scenario gave us very similar results to the scenario 1. The results obtained with the execution of experiments using this scenario, with lottery, FIFO and SJF are shown in Figure 3. However, now, SJF got slightly better results than the other algorithms. Again, one can claim that the results are very similar since tasks have the same size and the effort to execute all algorithms is also similar.

4.2.3 Scenario 3: heterogeneous tasks

We tried heterogeneous task in this next experiment. In scenario 3, the server receives requests from three different clients. The first client issues 100 small tasks. The second client issues 100 average size tasks. The third client issues 100 long tasks. Each short task has 10 units of assessment measure. Each mean size task has 100 units of assessment measure. Finally, each long hard task has $1,000$ units of assessment measure.

The results obtained by the performing experiments using this scenario, using lottery, FIFO and SJF policies are shown in Figure 4.

In this third scenario, we observed a lower response time on the client with short and average tasks using the SJF policy. This policy got really better results than the others. The client with only large tasks went through a period of starvation: waiting until all the tasks of the first two customers were executed. Note also that the response time to clients with small and medium tasks is very similar. This is due to the fact that when finalized all small tasks in the queue, the queue server chooses the average task and all the long tasks must wait. Thus, the average task was the lowest at the time, and it was chosen to be the task performed before arriving a request of new small task. It was observed also that small and medium tasks execution were intercalated during scheduling.

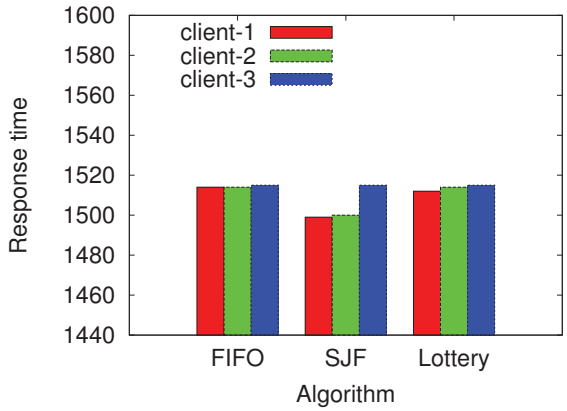


Fig. 5: Performance comparison of policies in scenario 4, using heterogeneous tasks and dynamic demand.

4.2.4 Scenario 4: heterogeneous tasks with dynamic demand

In scenario 4, the server receives one request per client and each request has a huge sequence of large and short tasks. First each client sends k large tasks to the server then each client sends n short tasks to the server.

The results obtained with the execution of experiments using this scenario, and policies lottery, FIFO and SJF, are shown in Figure 5.

In this scenario, despite differences in the results we got, it was observed that starvation also takes place, but not so

Table 1: Results in experiments with adaption.

System charge	Before adaption		After adaption	
	Algorithm	Value	Algorithm	Value
Short tasks	SJF	29	FIFO	28
Long tasks	SJF	2,979	Lottery	2,869
Heterogeneous tasks	FIFO	1,095	SJF	456
Dynamic demand	FIFO	1,484	SJF	1,469

severe as the previous scenario, due to the fact the client 3 has had half of its requests made in the middle of the experiment. Here, 50 small tasks were performed before starting the execution of the large tasks for the clients 1 and 2.

To remedy the situation of starvation, in this case, it could be implemented a more tricky approach, but reducing the weight of requests according to a criteria, such as age.

4.3 Experimental evaluation with adaption

After comparing the policies and finding what policy is most appropriate for each situation, the policies were assessed again. This time, algorithms are compared each other, aiming to adapt the scheduling.

Instead of changing the scenario and waiting for the scheduler to decide the policy, in these new experiments, we changed the current policy to another policy. In general, as expected, the service has improved.

To demonstrate situations of adjustment, experiments were performed in the same four different scenarios (section 4.2). However, this new battery of experiments, the scheduling changes its policy on time it reaches half the customers' demand, seeking to improve the service. Results are summarized in Table 1.

4.3.1 Adaptation in scenario 1

First, we take into account a scenario with small and homogeneous tasks. At the beginning of the experiment, the scheduler uses the SJF policy. However, when the experiment reaches the half of the demand of the three clients, the scheduler changes its policy to lottery. In this experiment, the average response time slightly decreases from 29 to 28 units of assessment measure.

Again, we cannot claim that lottery scheduling is better than FIFO because the values we got are very similar. With this experiment, we just want to test our adaptation service.

4.3.2 Adaptation in scenario 2

Scenario 2 deals with homogeneous and long-term tasks. In this experiment, the scheduler uses, initially, the SJF policy. However, when the experiment reaches the half of the demand of the three clients, the scheduler turns to lottery approach. In this situation, the average response time decreases, slightly, from $2,979$ to $2,869$ units of assessment measure.

Again, we cannot claim that lottery scheduling is better than SJF because the values we got are very similar. With this experiment, we just want to test our adaptation service one more time. Indeed with tasks with the same size, there is no sense in executing SJF. But, as expected, in contrast to the first scenario, we got high values in response time here since tasks in scenario 2 are longer than in scenario 1.

4.3.3 Adaptation in scenario 3

So, let's move to a more realistic scenario with heterogeneous tasks. In fact, the presence of heterogeneous tasks is common in many real applications.

In this experiment, the scheduler uses initially FIFO policy. However, when the experiment reaches the half of the demand of the three clients, the scheduler changes its policy to SJF.

Thus, the average response time decreases considerably more than half, from $1,095$ to 456 units of assessment measure. In the scenario with heterogeneous tasks, it is easier to realize the gain on the exchange policy from FIFO to SJF. However, the drawback of SJF is how to predict the duration on the jobs.

4.3.4 Adaptation in scenario 4

Finally, we take into account adaptation in scenario 4, with heterogeneous tasks and dynamic demand. Now, clients request 25 large tasks, followed by 25 small tasks, followed by another 25 large jobs and 25 small tasks.

At the beginning of the experiment, the scheduler uses the FIFO policy. However, when the experiment reaches the half of the demand of the three clients, the scheduler turns to SJF policy. In this situation, the average response time decreases, very subtly, from $1,484$ to $1,469$ units of assessment measure.

Despite of the large jobs in the queue, when the server turns from FIFO to SJF, the response time is maintained because large jobs are processed only at the end of the experiment.

4.4 Final remarks

Analyzing the results obtained from the experiments in different scenarios, one can notice that the scheduling algorithm choice is not a complex task when the system is subject to constant demand of similar processes.

However, in the presence of heterogeneous tasks, SJF tends to outperform the other evaluated policies. In our proposed approach, observing the clients' behavior, the scheduler can adapt its policy to better serve them. One important restriction of this policy is that computing the time a job needs to be processed a $priori$ is really tricky.

Finally, with regard to the adjustment (i.e. switching from a policy to another), when changing from FIFO or lottery scheduling to SJF, we got the best performance results or, at least, we maintain the performance.

5. Conclusions

In this paper, we presented an adaptive scheduler in the context of computer systems. We implemented and evaluated different scheduling policies (FIFO, SJF and lottery) using the support of the Lua programming language. The scheduler adapts following the response time provided by clients' requests. To enable the evaluation of our approach, experiments were performed considering four different scenarios of tests with different amounts of tasks and different weights for these tasks.

Analyzing the results obtained with the experiments in different scenarios, one may claim that the scheduling approach choice is not so relevant in situations where all clients have similar size tasks. However, in a scenario with tasks with different sizes, SJF achieves the best results. When we take into account adaptation, the experiments have shown that SJF showed the best results in the investigated scenarios.

As a continuation of this work, other policies and metrics could be used to allow the adaptation of the scheduling processes. Thus, further works include the following: (i) investigation of the use of a decentralized architecture with load distribution among different servers, (ii) investigation scenarios where the frequency of issuing requests among customers is different, (iii) evaluation of other scheduling policies and more complex policies, for instance, those that implement requests queues using heaps, and (iv) evaluation of the use of data mining to predict client's behavior and to choose the appropriate policy.

References

[1] R. Badonnel, M. Burgess, "Service load balancing with autonomic servers: reversing the decision making process", in Proceedings of 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS 2008). LNCS 5127. Bremen, Germany. pp. 92-104, 2008.

[2] S. Bouchenak, N. De Palma, D. Hagimont, C. Taton, "Autonomic management of clustered applications", in Proceedings of the IEEE International Conference on Cluster Computing, vol., no., pp.1-11, 25-28 Sept. 2006.

[3] A. Cervieri, R. Silva de Oliveira, C. F. Resin Geyer, "Uma abordagem de escalonamento adaptativo no ambiente Real-Time CORBA", in Proceedings of the XX Simpósio Brasileiro de Redes de Computadores (SBRC'2002). Búzios - RJ, 2002.

[4] J. M. Ewing, and D. A. Menascé, "Business-oriented autonomic load balancing for multitiered Web sites", in Proceedings of the 17th Annual Meeting of the IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2009), London, UK, 2009.

[5] R. Ierusalimschy, L. H. de Figueiredo, W. Celes. "Lua Reference Manual" (Rio de Janeiro: Lua.org), 2006. 103 p.

[6] J. O. Kephart, D. M.Chess, "The vision of autonomic computing", Cover Feature, IEEE Computer Society, January 2003.

[7] W.-S. Li, D. C. Zilio, V. S. Batra, M. Subramanian, C. Zuzarte, and I. Narang,"Load balancing for multi-tiered database systems through autonomic placement of materialized views", in Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), 2006.

[8] A. S. Tanenbaum, "Modern Operating Systems", Prentice Hall Press, Upper Saddle River, NJ, USA, 1992.

[9] C. A. Waldspurger, W. E. Weihl, "Lottery scheduling: flexible proportional-share resource management", in Proceedings of the 1994 Operating Systems Design and Implementation Conference (OSDI '94). Monterey, California. November, 1994.

[10] H. Zhang, X. Qiu, L. Meng, and X. Zhang, "Design of distributed and autonomic load balancing for self-organization LTE", in Proceedings of the IEEE 72nd Vehicular Technology Conference Fall (VTC 2010-Fall), pp. 1-5, 2010.

Complex Query JOIN Optimization in Parallel Distributed Environment

Dr. Sunita M. Mahajan[1], **Ms. Vaishali P. Jadhav** [2]
[1]Principal, Computer Science Dept., Mumbai Education Trust , Bandra , Maharashtra, India
[2]Research Scholar, NMIMS University, Mumbai, Maharashtra, India

Abstract *- The research work covers the query optimization concept in parallel distributed environment. The queries considered are select-project-join (SPJ) queries with large databases. The main query operation considered for research is JOIN operation of the query. For fast execution of a complex query, JOIN operation time needs to be minimized. Different JOIN operation algorithms such as Network Byte Order (NBO) parallel two way semi JOIN with bit array, NBO parallel collision free intelligent bloom JOIN filter, NBO parallel positional encoded reduction filter (PERF) JOIN and NBO parallel distinct encoded reduction filter (DERF) JOIN are implemented and evaluated with existing algorithms*.

Keywords: Query Optimization, JOIN Operation, Network Byte Order, Parallelization, and Distributed Environment.

1 Introduction

A query is defined as content retrieval from database on demand. It can be as easy as "Retrieving name of the person with PAN card number AAQPW2130D" or more complex like "Finding the amount of EMI of all bank customers whose age is between 30-39 years, having more than 5 years of work experience, having loan amount between 20 lacks to 50 lacks and want to make up loan within 5 years having loan period of 20 years with floating interest". After carrying out a JOIN operation between query tables, the query results are produced. The JOIN order of the tables decides the performance of the query. Query optimizer determines JOIN order via different JOIN algorithms in diverse environments. Depending upon the algorithmic change in each JOIN algorithm, query JOIN optimization time may vary. The research focus is to reduce the optimization time and network cost (in terms of amount of data to be transferred on network) of JOIN operation of a complex query in large databases.

The following Fig.1 gives an example of a complex query consists of many relations R1-R15. To produce the final JOIN operation result in minimum time, we need to reduce the time required by intermediate JOIN operations. To get the fast result, we parallelize a set of query optimization procedures for improving the performance of JOIN operation in a complex query on large databases in parallel distributed environment [1-17] [18-23] [29-31] [48].

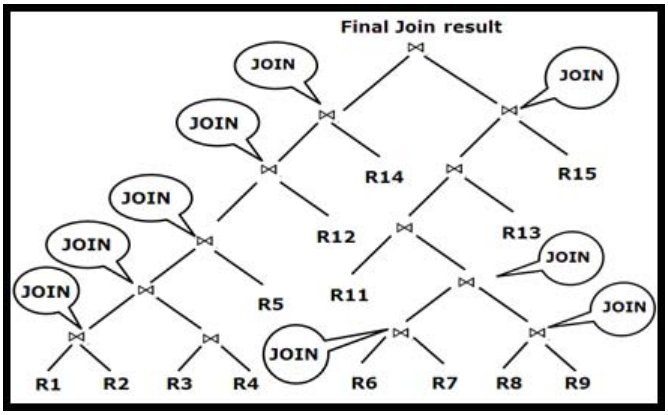


Fig. 1 Distributed Complex Query with Multiple JOINs

For implementing parallel JOIN algorithms, intra-operator parallelism and inter-query parallelism with shared nothing architecture is used. In intra-operator parallelism, single query JOIN operation is executed and in inter-query parallelism, multiple queries are executed on multiple nodes in distributed environment [18-23] [29-32] [48].

In parallelization, various load balancing schemes are used. The load balancing schemes that we used are round-robin, total-sum, equi-depth and stratified-allocation. The cost of parallel execution of JOIN operation includes the cost of data partitioning, data assembling and maximum execution cost of JOIN operation on multiple nodes. For achieving better results of parallelism, the combination of independent and pipelined parallelism is used. In independent parallelism, each node works independently with the data allotted to it. The pipelined parallelism gathers the results in pipeline and gives the final result [9-11] [42-45].

Our research work gives extra facilities during optimization phase. Query JOIN optimizer converts actual JOIN attribute values into a compressed binary form. But the problem is different CPU platforms may store compressed binary data types differently [17].i.e. Little Endian or Big Endian representation. To solve this problem, this compressed binary data is again converted into ***network byte order*** representation

which will be an intermediate storage of binary data to be transmitted across network [17] [40] [42]. This NBO data again is encrypted and provides the security during transmission of data on network.

The research work focused on various challenges occurred during optimization of query in different environment such as centralized environment, distributed environment and parallel environment. We focused on parallel environment. Producing the query result in less execution time, reducing communication cost when data is distributed among different sites, preventing data loss during JOIN operation, eliminating duplicate data during data transfer, reducing the amount of data to be transferred using data compression techniques, securing the data during transmission are some of the challenges in our research work[9-11] [42-15].

Based on certain parameters such as execution time, memory utilization, amount of data to be transferred on network, speed and efficiency, the query JOIN optimization algorithms are evaluated [40-48].

2 Related Work

The existing JOIN algorithms that we considered for our research are as follows:

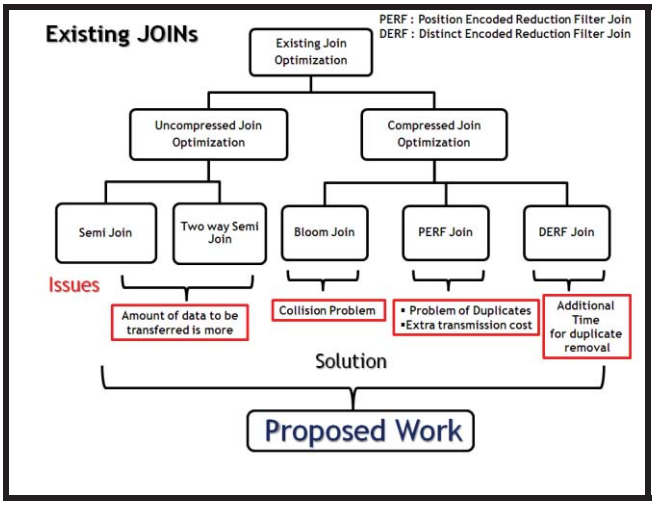


Fig. 2 Existing JOINs

For research in optimization of complex queries, we studied the existing JOIN optimization algorithms. Existing JOIN optimization algorithms are classified as Uncompressed and Compressed JOIN optimization. We studied semi JOIN and two way semi JOIN optimization techniques [1- 5] [19-23] [42-48] [30-35]. In Uncompressed JOIN optimization, the actual JOIN attribute values are transmitted so it increases the transmission time as well as network data. In two way semi JOIN optimization, the common and uncommon JOIN attribute values are compared but data is still in uncompressed form. We then studied some compressed techniques such as Bloom JOIN [6], Position Encoded Reduction Filter (PERF) JOIN [7, 8] [41-47] and Distinct Encoded Reduction Filter (DERF) JOIN [7,8][41-47]. Bloom JOIN faces the collision problem i.e. more than one elements point to the same address. PERF JOIN uses position encoding which is not suitable for large databases and DERF JOIN requires more time to remove duplicates. So our research fills these gaps by using parallelization to improve the performance of JOIN operation [24-28] [36-39] [47]

3 Research Work

We develop four JOIN optimizers in parallel distributed databases which focus on network data reduction, time-memory reduction and security during data transmission on network.

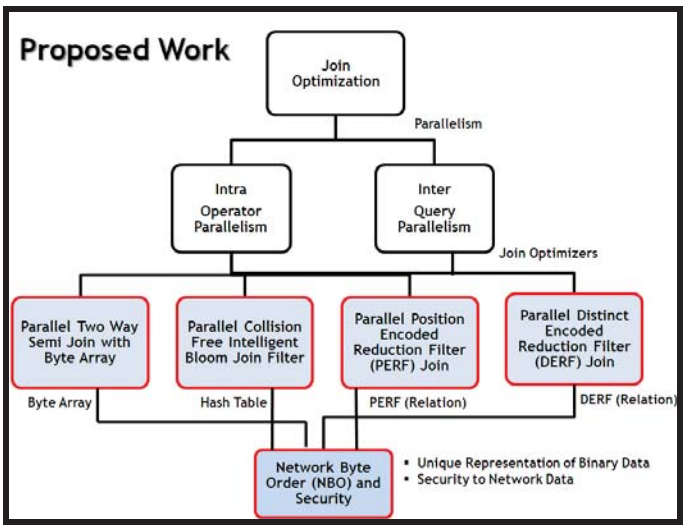


Fig. 3 Proposed Work

Instead of sending a data in its actual format, data is converted into binary format i.e. data is compressed and cost in terms of data transfer is reduced [30-35]. The main issue with binary data transmission is its compatibility with binary data representation on different machines [40]. Different machines can have different binary representations such as little endian or big endian i.e. the way to read binary data may be different. Machines with little endian binary representation read Least Significant Byte (LSB) first and machines with big endian binary representation read Most Significant Byte (MSB) first. So while binary data is transferred between different machines with different representations, data can be interpreted wrongly. To avoid this problem, our research converts this ordinary binary data into network byte order (NBO) which is transparent to any binary representation [40][42].

Another issue that we consider in our research is security of data during transmission on network. The next step after conversion of binary data into NBO is providing data security by encrypting NBO using Advanced Encryption Security

(AES) algorithm. The encrypted data is transferred on network and provides security to original data [40] [42-47] [50]. Thus the problems of data reduction, binary data representation and security issues are solved with above solutions (Data Compression, NBO and AES). The time and memory reduction is achieved by using parallelization concept in query optimization [9-11] [42-45].

The detailed block diagram of our system is given below in Fig. 4

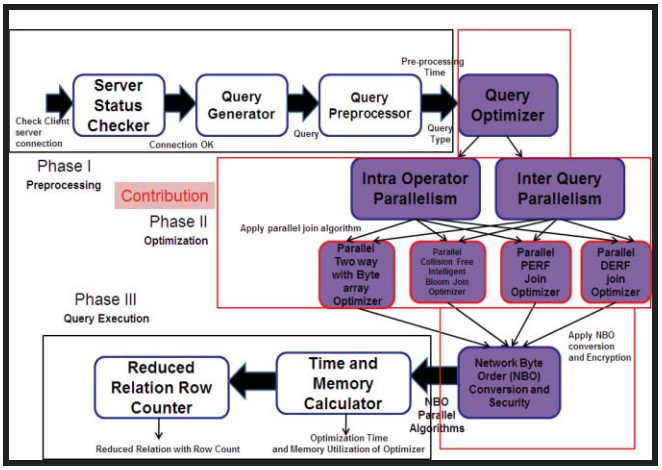


Fig. 4 Block Diagram and Contribution

The basic block diagram of a system is consists of 3 phases: preprocessing, optimization and query execution as shown in Fig. 3. Pre-processing consists of server status checker module which checks the connection between server and client in distributed environment. If the connection is OK then query generator module generates the query. According to the number of JOIN attributes of query tables, adjacency matrix is created in query pre-processing module. It gives the pre-processing time and query type as its output. Query type defines the number of relations in a query with number of JOIN attributes. For example query type 4-#3 means number of relations is 4 and number of JOIN attributes is 3 in a query.

The optimization phase consists of intra-operator and inter-query parallelism. Intra-operator parallelism consists of parallelization of single operator i.e. JOIN operator on multiple machines. Inter-query parallelism consists of multiple queries executed simultaneously on multiple machines [9-11] [45].Our four parallel JOIN optimizers are available in both environment i.e. intra- operator and inter-query parallelism. After the data compression and data reduction is over in all parallel JOIN optimizers, compressed data is converted into network byte order (NBO). NBO data is then encrypted with the help of AES algorithm and provides the security during transmission of data on network [43-47] [50].
In execution phase, the query is executed with calculation of time and memory requirement. The row count of reduced relation is also calculated in execution phase [1-5] [19-23]. The queries considered for research are star, chain, cycle and clique. The five databases used for evaluation are TPC-H, Northwind, Pub, AdventureWorks and Query Optimization (customized database used for testing) [13-15]. Databases can be scaled with proper scale factors.

The four JOIN optimizers developed are

- NBO parallel two way semi JOIN with byte array
- NBO parallel collision free intelligent bloom JOIN
- NBO parallel positional encoded reduction filter JOIN
- NBO parallel distinct encoded reduction filter JOIN

3.1 NBO Parallel Two Way Semi JOIN with Byte Array

Original two way semi JOIN calculates the common and uncommon JOIN attribute values and compares the count of both the values. The less count of common or uncommon values are transmitted on network and used further for reduction at resultant site [1-5] [19-23] [42-44] [50] [30-35].

The detailed algorithmic contribution of our first NBO parallel algorithm is given below in Fig 5.

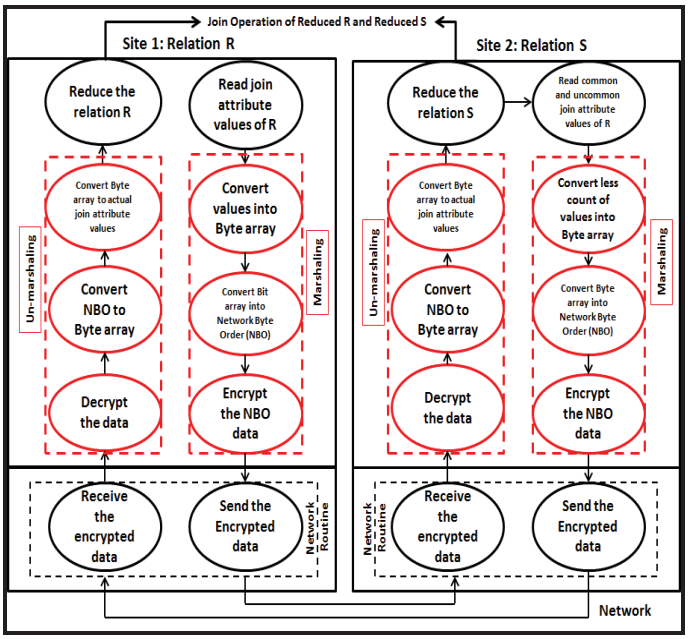


Fig. 5 Algorithmic Contribution of NBO Parallel Two Way Semi Join with Byte Array

Two way semi JOIN optimizer can also be executed in parallel environment. Instead of sending actual JOIN attribute values, our optimizer encodes the lesser count of either common or uncommon JOIN attribute values into byte array to save transmission cost. The byte array is converted into network byte order (NBO) to solve the problem of compatibility of binary representations on multiple machines [40]. Then NBO data is encrypted using AES algorithm for providing security

during data transmission [12] [42-47] [50]. Parallelization of above optimizer improves the performance of original two way semi JOIN optimizer [9-12] [17] [30-35][38]. We have used intra-operator and inter-query parallelization in our research.

3.2 NBO Parallel Collision Free Intelligent Bloom JOIN Filter

Original bloom JOIN filter uses hash function to map JOIN attribute values of relations. The main problem of hash function is collision problem. More than one JOIN attribute values can be mapped to the same address in collision problem [6] [24-28] [36-39].

Instead of using hash functions in bloom JOIN, our algorithm used C# data structure i.e. Hash Table. The problem of collision due to hash functions is avoided by using hash table. Also hash table saves only the distinct values so parallel collision free intelligent bloom JOIN with hash table is useful in large databases and improves the performance of original bloom JOIN optimizer[9-12] [17].

The main contribution of this algorithm is conversion of JOIN attribute values into hash table in forward reduction of JOIN operation. The hash table values are converted into network byte order for solving the problem of binary representation. Security to transmitted data is provided by using AES algorithm. In backward reduction, the less count of common and uncommon values are converted into hash table, then network byte order conversion and then AES algorithm is applied for security purpose.

3.3 NBO Parallel Position Encoded Reduction Filter JOIN

Original positional encoded reduction filter (PERF) encodes the position of JOIN attribute value into byte array. Considering JOIN operation between two relations, PERF JOIN sets byte array value equal to 1 for the common JOIN attribute value and sets 0 for the uncommon JOIN attribute value [7-8] [41-47].

The problem with original PERF JOIN is it encodes position in byte array, as the JOIN attribute values increases, the size of byte array increases. This problem is solved by NBO parallel PERF JOIN which compares common and uncommon JOIN attribute values and instead of encoding all JOIN attribute values, less count of common and uncommon values are used for encoding.

The conversion process of compressed data to NBO data and then NBO data to encrypted data remains same for all optimizers. The PERF JOIN encodes all the values including duplicates.

3.4 NBO Parallel Distinct Encoded Reduction Filter JOIN

In PERF join, duplicate JOIN attribute values are also encoded. Duplication increases the cost in terms of response time, transmission time as well as it increases the amount of data to be transferred on network. To solve this problem, original distinct encoded reduction filter (DERF) JOIN selects distinct JOIN attribute values while transmitting data on network. It solves the problem of duplication [7-8] [41].

Our NBO parallel DERF JOIN further reduces the cost in backward reduction phase of JOIN operation. NBO parallel DERF provides security during transmission of data as well as reduces transmission overhead. In backward reduction phase, the distinct common and uncommon JOIN attribute values are compared and less count of these values is represented into byte array for further reduction.

4 Experimental Set Up

For experimentation, we consider 3 servers (Configuration: Intel® Core™ 2 Duo Processor E7300, 2.66 GHZ, 3 MB Cache) and 20 Clients (Configuration: 2.9GHz Intel Core i5 processor,4GB DDR3 RAM, 500GB hard drive) in one data center. During experimentation, database of size 500GB is considered and 1500 queries are evaluated. The queries considered for research are star, chain, cycle and clique. The five databases used for evaluation are TPC-H, Northwind, Pub, AdventureWorks and Query Optimization (customized database used for testing) [13-15] [16-17]. Databases can be scaled with proper scale factors.

Different load balancing schemes are used during experimentation such as round-robin, total-sum, equi-depth and stratified-allocation [12]. We considered all the test cases (Best, Average and Worst) during our experimentation. This research paper includes the results of 'equi-depth' load balancing scheme with all types of queries in intra-operator and inter-query parallelism.

We have used following parameters for evaluation of our optimizers.

- Execution time
- Memory utilization
- Amount of data to be transferred
- Speed –Up
- Efficiency

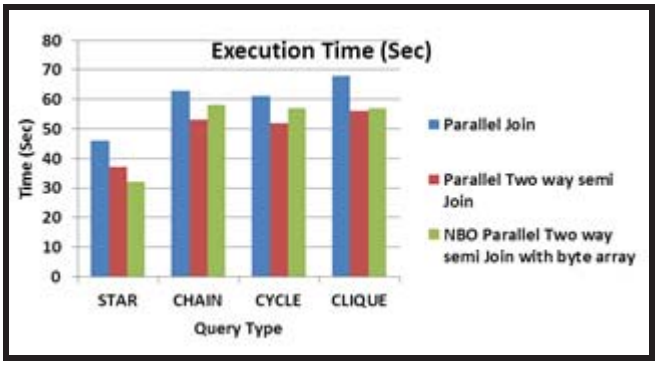
With the help of these evaluation parameters, optimizer's performance is evaluated. All NBO parallel optimizers with all types of queries in intra-operator and inter-query parallelism are shown in results.

5 Results and Discussion

This section includes the results of our first NBO Parallel JOIN optimizer in worst case intra-operator parallel environment. Our algorithms work well with best case and average case for all types of queries.

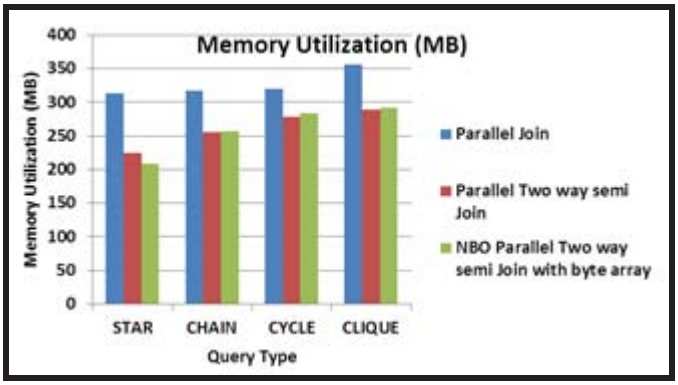
Our first optimizer i.e. NBO Parallel Two Way Semi JOIN with Byte Array is compared with two existing JOIN optimizers i.e. Parallel JOIN and Parallel Two Way Semi JOIN. This optimizer is evaluated on basis of all evaluation parameters such as execution time, memory utilization, amount of data to be transferred, speed up and efficiency. Table 1 shows the evaluation of our optimizer using execution time.

Table 1 Execution Time



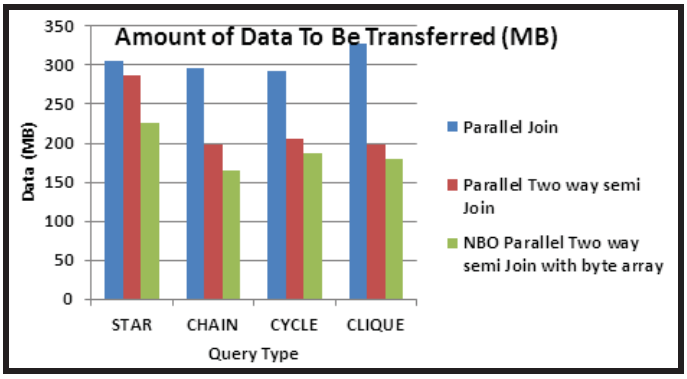
NBO Parallel Two way Semi JOIN Optimizer works well with all queries in best and average case. The time required to execute Chain, Cycle and Clique queries is more than the time required to execute existing optimizers in worst case. So our optimizer works well for star queries in worst case scenario. The time required to count common and uncommon JOIN attribute values may take extra time for more complex queries in worst case scenario.

Table 2 Memory Utilization



The memory utilization is also less for Star queries in worst case scenario. Memory required by Chain, Cycle and Clique is somewhat more than the existing optimizers.

Table 3 Amount of Data to be Transferred



Our NBO parallel optimizers used compressed data during transmission on network. So in all test cases, the amount of data to be transferred is less than existing JOIN optimizers. In worst case also, the amount of data to be transferred on network is less for all types of queries.

We compared all our NBO parallel optimizers with above evaluation parameters. For other two evaluation parameters such as Speed-Up and Efficiency, we compared all four NBO parallel optimizers with existing parallel optimizers.

Table 4 Speed Up

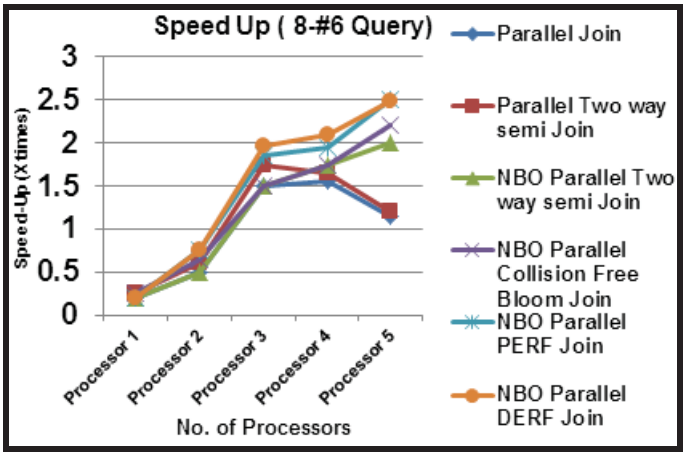
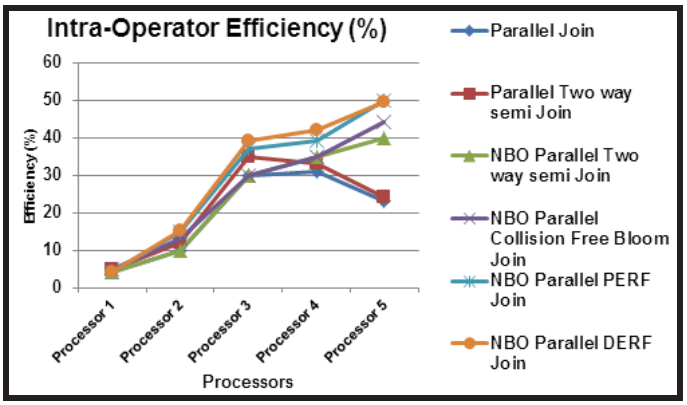


Table 5 Efficiency



Our four NBO parallel optimizers are compared with existing two optimizers.Speed Up and efficiency of all optimizers is measured. Among all four optimizers , NBO Parallel Distinct Encoded Reduction Filter gives more speed and is more efficient than other 3 NBO parallel optimizers.

6 Conclusion

NBO parallel optimizers significantly improve the performance of JOIN operation in parallel distributed environment along with reduction in the execution time and amount of data to be transferred on network. The compression techniques such as Byte Array conversion, Hash Table conversion, PERF(Relation) conversion and DERF(Relation) conversion solves the existing issues in JOIN operation. We found 'Equi-Depth' is better load balancing scheme among other load balancing schemes. Our evaluation parameter, 'amount of data to be transferred' is always acceptable for all queries in all cases.

NBO Parallel Two Way Semi JOIN with Byte Array and NBO parallel PERF JOIN is suggested for star queries. NBO Parallel Collision Free Intelligent Bloom JOIN Filter is suggested for star and chain queries. NBO Parallel DERF JOIN is suggested for all types of queries i.e. star, chain, cycle and clique. NBO Parallel DERF is the best optimizer among all other NBO parallel optimizers.

In future these NBO parallel optimizers can be tested on unstructured databases or column oriented databases.

7 References

[1]Abraham Silberschatz, Hank Korth and S. Sudarshan. Database System Concepts, 5th Edition, McGraw-Hill, 2006

[2]"SQL Statement Processing" http://technet.microsoft.com/ens/library/ms190623(v=sql.105).aspx

[3]Goetz Graefe, Query Evaluation Techniques for Large Databases, *ACM Computing Surveys*,Vol. 25, No. 2, June 1993.

[4]"Application areas of Databases", http://my.safaribooksonline.com/book/databases/9788131731925/databasesystem/ch01lev1sec4

[5]Anand V. Hudli, "Distributed Query Processing", M.Tech Dissertation IIT Bombay, 1984

[6]J.M.Morrissey and W.Osborn,"Experiments with the use of reduction filters in distributed query optimization" , in *proceedings of the 9th International Conference on Parallel and Distributed Computing and Systems*,(pp.327- 330).

[7]Ahmet Cumhur ÖZTÜRK, "Distinct Encoded Records Join Operator for Distributed Query Processing", Thesis of Masters of Science, İzmir Institute of Technology. Izmir 2012.

[8]Zhe Li and Kenneth A. Ross"PERF Join: An Alternative To Two-way Semijoin And Bloomjoin" http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.2324&rep=rep1&type=pdf

[9]"Chapter 8:Parallel Query Optimization" http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.dc20023_1251/html/optiz-er/X23652.htm

[10]"Parallel Hardware Architeture" http://docs.oracle.com/cd/A58617_01/server.804/a58238/ch3_arch.htm

[11]Patrick Valduriez , " Parallel Database Systems : Open Problems and New Issues" , *Distributed and Parallel Databases I* (1993) 137-165

[12]S.K. Basu, "Design Methods and Analysis of Algorithms", PHI learning Pvt.Ltd. Second Edition.

[13]http://en.wikipedia.org/wiki/Transaction_Processing_Performance_Council

[14]https://northwinddatabase.codeplex.com/

[15]http://www.codeproject.com/Articles/20987/HowTo-Install-the-Northwind-and-Pubs-Sample-Databa

[16]https://msftdbprodsamples.codeplex.com/releases/view/93587

[17]http://www.ccse.kfupm.edu.sa/~fazzedin/COURSES/CSP2005/Reading/NetworkProgramming.pdf

[18]http://nou.edu.ng/NOUN_OCL/pdf/pdf2/DAM%20%20212.pdf

[19]http://www.it.bond.edu.au/inft320/001/lectures/qproc3.pdf

[20] http://www.srpskibre.com/pdf/Fundamentals_of_Database_Systems.pdf

[21]G.M.Lohman , C. Mohan, L.M. Haas, D. Daniels, B.G.Lindsay, P.G. Selinger, and P.F. Wilms. "Query

processing in R*".In query processing in Database Systems. Springer ,New York, 1985.

[22]J.K.Ahn and S.C.Moon. "Optimizing joins between two fragmented relations on a broadcast local network". Information systems, vol. 16, no. 2, pages 185-198, 1991.

[23]J.S.J. Chen and V.O. K. Li. "Optimizing joins in fragmented database systems on a broadcast local network". IEEE Transaction on software Engineering, vol. 15, no. 1, pages 26-38, 1989.

[24]W.K.Osborn."The use of reduction filters in distributed query optimization" Master Thesis, University of Windsor, 1998.

[25]W.T.Bealor."Semijoin strategies for total cost minimization in distributed query processing". Master Thesis, University of Windsor, 1995.

[26]W.T. Bealor and J.M. Morrissey, "Minimizing data transfer in distributed query optimization: A comparative study and evaluation", Computer Journal, Vol 39, No. 8, 1997.

[27]J. M. Morrissey , S. Bandopadhyay, and W.T. Bealor. " A heuristic for minimizing total cost in distributed query processing". In proceedings of the 7Th International Conference on Computing and Information – ICCI'95, 1995.

[28]J.M. Morrissey, S.Bandopathyay, and W. T. Bealor ." A comparison of static and dynamic strategies for query optimization" In proceeding of the 7th IASTED/ISM International Conference on Parallel and Distributed Computing Systems, 1995.

[29]Bernstein, PA, Godman, N.Wong, E.Reeve, C, and Rothnie, J, " Query processing in a system for distributed databases (SDD-1)" , ACM Trans. syst. Vol.6, Dec 1981. Pages 602-625.

[30]Yu, CT and Chang CC, "On the design of a query processing strategy in a distributed database environment", Proc. ACM SIGMOD Intl. Conf. Management of Data, 1983, pages 30-39.

[31]Apers,PMG,Hevner,A, and Yao,SB, " Optimization algorithm for distributed queries", IEEE Trans. Software Engg, Vol. 9, No. 1, Jan. 1983, Pages 57-68.

[32]C. Wang, V.O.K. Li and A.L.P.Chen. " Distributed query optimization by one shot fixed precision semijoin execution". In Proceedings of the 7th International Conference on Data Engineering, pages 756-763, 1991.

[33]C. Wang, V.O.K. Li and A.L.P. Chen."One-shot semi join execution strategies for processing distributed join query". Computer Systems Science and Engineering, Vol. *, No. 4, pages 245-253, 1993.

[34] H.Kang and N. Roussopoulos." Using 2-way semijoins in distributed query processing" In proceedings of the 3rd International conference on Data Engineering, pages 644-651, 1987.

[35]N. Roussopoulos and H.Kang,"A pipielined n-way join algorithm based on the 2-way semijoin program". IEEE Transactions on knowledge and Data Engineering, 3(4) pages 486-495,1991.

[36]B.H.Bloom,"Space/time tradeoff in hashing coding with allowable errors", Communication ACM, Vol.13, July 1970, pages 422-426.

[37]J.M. Morrissey and X.Ma, "Investigating response time minimization in distributed query optimization, In proceedings of the 10Th International Conference on Computing and Information – ICCI'98, 1998.

[38]J.C.R. Tseng and A.L.P. Chen,"Improving distributed query processing by hash semijoins". Journal of Information Science and Engineering. Vol. 8, pages 525-540, 1992.

[39]J.M. Morrissey and W.K. Osborn."Experiments with the use of reduction filters in distributed query optimization." In proceedings of the 9th IASTED International Conference on Parallel and Distributed Computing and Systems, 1997.

[40]http://www.goldparser.org/doc/about/byte-ordering.htm

[41]Zhe Li. and Kenneth A. Ross." PERF Join: An alternative to two ay semi join and Bloom join", Department of Computer Science, Columbia University, New York, NY 10027.

[42]http://library.iyte.edu.tr/tezler/master/bilgisayaryazilimi/T001058.pdf

[43]https://support.microsoft.com/en-us/kb/246071/

[44]http://mu.ac.in/myweb_test/MCA%20study%20material/Advanced%20Database%20Techniques-f.pdf

[45]http://web.cs.wpi.edu/~cs561/s12/Lectures/4-5/ParallelDBs.pdf

[46]http://mazsola.iit.uni-iskolc.hu/tempus/discom/doc/db/tema01a.pdf

[47]http://bnrg.cs.berkeley.edu/~adj/cs262/papers/graefe.pdf

Parallel relational databases for diameter calculation of large graphs

Fabiano da Silva Fernandes
Faculty of Campo Limpo Paulista (FACCAMP)
Campo Limpo Paulista, São Paulo, Brazil
Email: contato@fabianofernandes.adm.br

Eduardo Javier Huerta Yero
Faculty of Campo Limpo Paulista (FACCAMP)
Campo Limpo Paulista, São Paulo, Brazil
Email: huerta@faccamp.br

***Abstract*—Parallel relational databases are seldom considered as a solution for representing and processing large graphs. Current literature shows a strong body of work on graph processing using either the MapReduce model or NoSQL databases specifically designed for graphs. However, parallel relational databases have been shown to outperform MapReduce implementations in a number of cases, and there are no clear reasons to assume that graph processing should be any different. Graph databases, on the other hand, do not commonly support the parallel execution of single queries and are therefore limited to the processing power of single nodes. In this paper, we compare a parallel relational database (Greenplum), a graph database (Neo4J) and a MapReduce implementation (Hadoop) for the problem of calculating the diameter of a graph. Results show that Greenplum produces the best execution times, and that Hadoop barely outperforms Neo4J even when using a much larger set of computers.**

I. Introduction

For decades relational databases stood as the dominant choice for storing, managing and retrieving large datasets, as its widespread adoption indicate. The declarative nature of the relational model allows programmers to specify what data they want to store and retrieve, leaving it to them implementation to choose the data structures and algorithms necessary to do so. Furthermore, features such as integrity constraints, ACID properties and referential integrity, which are present in many implementations of the model, release the programmer from dealing with the intricacies of creating and maintaining a consistent dataset in the face of concurrent access and hardware or software failures.

However, the recent explosion of digitally available data has exposed weaknesses in relational databases. The challenges faced by these traditional systems can be broadly classified in:

- *Volume*: The amount of data stored has grown into orders of magnitude that overwhelm current relational databases. In particular, the size of the tables and, therefore, the time required to perform the *join* operations needed to execute queries (the so-called *join pain*) has made it unfeasible to manage modern datasets using traditional tools.
- *Velocity*: Data velocity refers to the rate at which data changes over time. Modern systems typically deal with high volumes of data operations, often write-heavy, concentrated in short periods of time. Furthermore, the data model usually changes over time as well, due mainly to the fluid nature of modern businesses and the experimental nature of data acquisition and processing. Relational databases have been known to perform poorly during peak loads and are ill prepared to deal with constantly changing data models.
- Variety: Today's data may be dense or sparse, data items may be interconnected or independent and may be structured, semi-structured or unstructured. This landscape is far removed from the type of problems relational databases were designed to solve. Forcing current data into a relational model often produces inefficient and counter-intuitive data models that are hard to maintain.

Relational databases have responded to some of these challenges by increasing the adoption of known techniques such as data sharding, column-oriented tables and the parallel execution of single queries. Databases supporting the latter feature are known as parallel relational databases and have been successfully deployed on highly demanding scenarios in the industry where the more traditional relational databases were not sufficient.

A host of alternatives approaches have also been developed to cope with these challenges. One of the most widely known is MapReduce, a simple and powerful programming model and implementation initially developed by Google [7]. Other approaches have been grouped under the name of NoSQL databases [11], representing their departure from the traditional relational model. NoSQL databases typically alter or completely eliminate some essential feature of the relational model (e.g. data model, ACID properties) in order to improve metrics regarded as more vital, such as performance, availability and/or fault tolerance.

In the particular case of graphs, a number of specific solutions such as GraphLab [18] and NoSQL graph databases such as Neo4J [8] have been developed. For large graphs, the recent body of work has focused on the MapReduce model [7] and to a lesser extent the Bulk Synchronous Parallel (BSP) model [27].

MapReduce has been proved to perform poorly for graph algorithms, both theoretically [20] and through practical experiments [12]. Some of the reasons behind this lack of performance are its reliance on an external data storage layer (e.g. HDFS for Hadoop), its need to materialize intermediate results on this storage layer (e.g the results from the map phase) and the stateless nature of the map and $reduce$ phases

ISBN: 1-60132-444-8, CSREA Press ©

that forces data to be redistributed at the beginning of every cycle.

Graph databases use their multi-node deployments for high availability and fault tolerance, but do not commonly support the parallel execution of single queries. Therefore, when processing complex queries on large graphs, their performance tends to be limited by the computing power of single nodes.

Although proven to be superior to MapReduce in a number of use cases [26], parallel relational databases are not commonly considered as a potential solution for graph problems in current literature. More interestingly, the characteristics that made parallel relational databases outperform MapReduce in [26] (i.e better data distribution and locality, indexing capabilities, execution plan optimization) are all true for graph algorithms as well. When compared to graph databases, parallel relational databases offer a less intuitive data model, but coupled with a highly tuned parallelization mechanism for single queries.

In this paper we intend to explore this issue by calculating the exact diameter of a graph with a graph database, a MapReduce implementation and a parallel relational database and comparing their performances. We selected the diameter calculation because it requires the calculation of the distance between every pair of vertices in the graph (known as the All-Pairs Shortest Path problem), which is computationally intensive. Also, an exact solution to this problem is less likely to be affected by the particular partitioning of the graph among the parallel nodes, since every vertex has to be considered against every other vertex in the graph.

We do not intend to present a full comparison of these models in the realm of graph algorithms, or to provide innovative algorithms for graph calculation (which is typically done using approximate algorithms [13]). Instead, our goal is to present an initial set of experiments to explore the feasibility of using a parallel relational database to process large graphs instead of a MapReduce implementation or graph databases. We believe, however, that the problem of calculating the exact diameter of a graph is complex enough to expose most of the strengths and weaknesses of each solution.

Rest of the paper is organized as follows. Section II describes literature relevant to this paper. Section III briefly presents each model compared. Section IV compares each model and points out their main differences. Section V describes the algorithms used for calculating the diameter of a graph on each model. Section VI explains the design of the experiments and the datasets used. Section VII presents the results of the experiments and comments on them. Finally, Section VIII presents the conclusions of this paper.

II. Related work

A large effort has been devoted to write graph algorithms for the MapReduce model and understanding their performance characteristics. In [6] the author identify challenges and propose MapReduce based solutions for graph mining, matching and querying. In [17] several design patterns are proposed whose objective is to enhance the performance of graph algorithms in MapReduce. In [23] a class of MapReduce algorithms is introduced that is named *Scalable Graph Processing Class*, and its demonstrated that algorithms belonging to it exhibit good scalability when processing large graphs.

The PEGASUS graph mining library [14], built on top of Hadoop MapReduce, implements typical graph-mining tasks such as calculating graph diameter and the radius of a specific vertex, as well as finding the connected components. For the specific problem of calculating the diameter of a graph, [13] offers an approximate solution, while [19] offers the exact results, both based on MapReduce.

MapReduce has also been compared to parallel relational databases in traditional scenarios not involving graphs. In [21] and later in [26] the authors compare the performance and suitability of MapReduce and two parallel databases for problems commonly found in traditional scenarios. The papers concludes that parallel databases are best suited for querying large datasets, while MapReduce is best suited for ETL (Extract, Transform and Load) and complex analytics.

MapReduce has also been extended or enhanced to improve its performance on graph algorithms. In [1] the authors propose an extension to the MapReduce model called PACT, which is based on the idea of parallel contracts. Among the problems used to compare both models are the pairwise shortest path and the edge-triangle enumeration in large graphs.

In [22] a MapReduce implementation is described that uses the well-known Message Passing Interface (MPI) as the communication mechanism. By giving up on some of the most heralded features of standard MapReduce implementations, such as fault tolerance and data redundancy, the authors are able to provide an implementation that outperforms its standard Hadoop counterpart in several graph problems.

In [5] is described a system called Surfer, which extends MapReduce with a *propagation* operation designed to ease the implementation of edge-oriented tasks in graphs. Similarly, [28] and [4] propose an extension to MapReduce that introduces the capability of executing iterative computations, typically needed for graphs algorithms.

Other models have also been compared to MapReduce in the context of graph algorithms. In [15], MapReduce is compared to the PRAM (Parallel Random-Access Machine) model using the Minimum Spanning Tree (MST) of a dense graph in the comparison. In [10] the authors compare the MapReduce, BSP, PACT and GraphLab [18] (which is based on the Gather-Apply-Scatter pattern) programming models for the k-core decomposition problem in graphs. With few exceptions, all other models performed better than MapReduce.

In [25] Microsoft's Trinity [24] and other platforms for graph computing, such as MapReduce, Pregel, Pegasus and several NoSQL based approaches are superficially compared. Finally, in [12] the authors compare MapReduce and BSP for graph algorithms. The problems chosen for the comparison were the Single Source Shortest Path (SSSP) and Relational Influence Propagation (RIP). The results show that BSP is better suited to handle graph problems, although the particular implementation chosen (Apache Giraph) presented some scalability issues for the larger graphs tested that discourage the adoption of this platform.

III. MODELS

In the MapReduce model the input data to be processed is divided into chunks, which are then fed to several *map* tasks. Each map task produces an output as a set of $\{key, value\}$, which are then shuffled by the MapReduce implementation such that all outputs with the same key are fed to the same $reducer$ task. The $reducer$ task then combines these intermediary results to produce the final output.

The power of the MapReduce model relies in letting the programmer focus on the details of the application while the MapReduce implementation deals with all problems associated with its parallel execution, such as code and data distribution, load balancing, fault tolerance and data flow.

Parallel relational databases work by harnessing the power of several database servers to execute single user queries in parallel. Traditional models for parallel and distributed databases include shared-memory, shared-disk and shared-nothing, although shared-nothing architectures have prevailed due to their simplicity, low cost and high performance and scalability.

Relational database servers execute SQL queries that consist of a set of relational operators applied to potentially large sets of data. SQL queries do not specify how they should be executed, thus giving database implementors considerable freedom in the execution plan and offering large opportunities for parallelism [9]. A SQL query can be executed as a dataflow graph, composed of a set of relational operators such that:

- the output of one operator can be the input of another, creating opportunities for pipelined parallelism
- the data to be processed can be partitioned and submitted to several dataflows whose results are merged at the end.

A graph database uses graph theory to represent discrete entities and their relationships. Entities are represented by graph nodes, while relationship among entities are represented through edges. Both nodes and edges can have associated properties to describe their characteristics, much like the Entity-Relationship model. In graph databases relationships are materialized and stored in the database at creation time instead of being calculated through join-like operations at execution time, so graph traversal is more natural and efficient.

IV. IMPLEMENTATIONS

The architectural differences between parallel relational databases, MapReduce and graph databases have been studied in current literature [9] [2] [8]. Comparing these architectures is difficult, given the number of different options available. In this work we chose the standard Hadoop/HDFS implementation of the MapReduce model, a mainstream graph database without support for parallel query execution (Neo4J) and a mainstream parallel relational database (Greenplum). We will now briefly compare these particular systems instead of the general models to which they belong.

- *Parallelism support*: Both Hadoop and Greenplum support parallelism and therefore are capable of partitioning the data to be processes among the available nodes. The main difference is that Hadoop materializes intermediate results on an external data storage and requires the map phase to be finished before the reduce phase begins. Greenplum exploits pipeline paralellism to immediately push intermediate results to the next processing stage. Neo4J do not support parallelism.
- Indexing: Both Neo4J and Greenplum support indexing, while Hadoop does not.
- *Input parsing*: Input data is parsed at creation time by both Greenplum and Neo4J, while Hadoop requires data to be parsed at every execution.
- *Scheduling*: Greenplum creates a distributed query plan so that each node knows exactly what to do before the query execution begins. Such compile-time scheduling contrasts with Hadoop, that dynamically allocates input blocks to map and reduce tasks at execution time. Compile-time scheduling usually produces better performance results, while execution-time scheduling is better at adapting to workload skews and changes in node performance.
- *Fault tolerance*: Both Neo4J and Greenplum provide transaction-level fault tolerance, meaning that if an operation within a transaction fails the whole transaction is rolled back. Hadoop provides block-level fault tolerance, so that if a task assigned to process an input block fails then that block will be reallocated to another task.

V. DIAMETER CALCULATION

The diameter of the graph is calculated using the same strategy for all approaches: a breadth-first search expanding from every vertex of the graph until it reaches every other vertex. This algorithm produces the eccentricity of each vertex, and the diameter of the graph is then determined as the maximum value of all eccentricities.

A. MapReduce algorithm

The MapReduce algorithm used in the comparison is based on the one presented in [19]. The driver program, that repeatedly calls the $map()$ and $reduce()$ functions until all vertices are processed, is shown in Figure 1.

input: Graph G
output: Diameter D
  for all $v \in G$ **do**
    $P \leftarrow P \bigcup \{v, v, 0, PROCESSING\}$
  end for
  while $\exists path \in P, path.state = PROCESSING$ **do**
    map()
    reduce()
  end while
  $D \leftarrow$ **Max**$(P.distance)$

Figure 1. MapReduce main algorithm for the exact calculation of graph diameter

This algorithm relies on a path vector P that contains all currently known paths. Each vector entry contains the path

origin vertex ($path.origin$), the destination vertex ($path.dest$), the currently known distance ($path.distance$) and the path state ($path.state$), which can be either $PROCESSING$ or $PROCESSED$, depending on whether all neighbors of the destination vertex of the path have been processed. The algorithm initializes the path vector (with paths from each vertex to itself with zero distance and PROCESSING state) and repeatedly calls $map()$ and $reduce()$ until all paths are in state $PROCESSED$.

Figure 2 presents the $map()$ function invoked by Figure 1.

```
input: Graph G, path ∈ P
output:
  (path.orig, path.dest) → (path.distance, path.state)

  if path.state = PROCESSED then
      Output((path.orig, path.dest) →
       (path.distance, path.state))
  else
      for all v ∈ Neighbors(path.dest) do
          Output((path.orig, v) →
       (path.distance + 1, PROCESSING))
      end for
      Return((path.orig, path.dest) →
       (path.distance, PROCESSED))
  end if
```

Figure 2. Map function for the exact calculation of graph diameter

The $map()$ function receives the complete graph G and an entry of the path vector P called $path$. If $path$ needs further processing, the map function outputs a new entry for every neighbor of the destination vertex $path.dest$ with its distance increased by 1. The $reduce()$ function invoked by Figure 1 is shown in Figure 3.

```
input:
  (path.orig, path.dest) → values =
  {(path.distance_1, path.state_1), ...}
output:
  (path.orig, path.dest) →
  (path.distance, path.state)

  path.distance ← Min(values.distance)
  /*Consider PROCESSED > PROCESSING */
  path.state ← Max(values.state)

  Output(path.orig, path.dest) →
      (path.distance, path.state)
```

Figure 3. Reduce function for the exact calculation of graph diameter

The $reduce()$ function receives as input all distances and states calculated for a specific pair of vertices. The path distance is calculated as the minimum distance of all entries in the input.

B. Relational DB algorithm

For the relational database algorithm the graph is represented by a table G with two fields containing the origin and destination vertices of an edge. The algorithm then produces a second table D with the distance for every pair of vertices, as shown in Table I.

Table I. TABLES USED TO REPRESENT THE GRAPH IN THE RELATIONAL ALGORITHM: (A) ORIGINAL GRAPH (B) OUTPUT

Name	Type
orig	int
dest	int

(a) Table G

Name	Type
orig	int
dest	int
distance	int

(b) Table D

The algorithm is described in Figure 4. It works by iteratively adding new entries to the result table D, such that at iteration i it contains all pairs of vertices that can be reached with i hops or less from every possible origin vertex. At every iteration a table $Frontier$ is created with the vertices reachable from destination vertices in D, and then updates D with the new entries. The loop finishes when no new entries are found, at which point the eccentricity for each vertex and the diameter of the graph are calculated.

```
input: Table G
output: Table D with distances for every pair of vertices
  D ← G, D.distance=1
  go ← true
  while go do
      Frontier ← Select D.orig, G.dest, D.distance+1
      From D, G Where D.dest = G.orig
      AND G.dest <> D.orig

      Update D with Frontier if D does not contain
      an entry with a lesser distance
      go ← (items updated > 0)
  end while
  Ecc ← Select orig, MAX(distance) From D
  Group By orig
  Diameter ← SELECT MAX(distance) From Ecc
```

Figure 4. Relational database algorithm used to calculate the diameter of a graph.

C. Graph database algorithm

The algorithm for the graph database algorithm is straightforward. It sequentially scans the graph and calculates the eccentricity of every vertex, which is then used to calculate the diameter of the graph, as shown in Figure 5.

```
  diameter ← 0
  for all Nodes in graph as nodeSrc do
      ecc ← 1
      for all Nodes in graph as nodeDest do
          ecc ← max(ecc,distance(nodeSrc,nodeDest)
      end for
      diameter ← max(diameter, ecc)
  end for
```

Figure 5. Sequential algorithm used in graph databases.

Graph databases offer convenient ways to traverse graphs. For instance, Neo4J provides a $Traverser$ class to reach every node in the graph starting at a source node according to settings

specified by the user. In the example shown in Figure 6 a $Traverser$ object is created from $nodeSrc$ that reaches all other nodes in the graph using a breadth-first strategy.

```
Traverser traverser = nodeSrc.traverse(
    Order.BREADTH_FIRST,
    StopEvaluator.END_OF_GRAPH,
    ReturnableEvaluator.ALL_BUT_START_NODE,
    RelTypes.KNOWS,
    Direction.BOTH);
```

Figure 6. Traverser object used in the Neo4J graph database to find all paths from a single node.

VI. Design of experiments

Experiments were conducted using a set of real and synthetic graphs of different sizes. The tests were designed to assess how execution time and speedup were affected by changing the size of the graphs and the number of available computers.

It is worth noting that the size of the graphs used were limited by the available computing time in the test environment. Since the calculation of the exact diameter of a graph is a demanding task, for larger graphs some of the tests would exceed the maximum allotted time and be interrupted.

A. Test environment

Tests were conducted on the Grid'5000 platform [3], a platform composed by approximately 1200 computers with more than 8000 cores combined, distributed among 11 locations in France. The experiments used two of the clusters offered by the Grid'5000 platform:

- *Cluster Paradent*: from the Rennes site, it is formed by 45 computers with two Intel Xeon L5420 processors each with four 2.66 GHz cores each (360 cores total), 32GB RAM and a 350GB SATA hard drive.
- *Cluster Graphene*: from the Nancy site, it is formed by 120 computers with an Intel Xeon X3440 processor containing four 2.53 GHz cores (480 cores total), 32 GB RAM and a 350 GB SATA hard drive.

Computers used in the experiments were connected by a 1 GB Ethernet switch. They were running Linux CentOS 5.9, kernel 2.6.18-371.4.1.el5.x86 64, Java Virtual Machine $1.7.0_51$, Python 2.7.6, Python-NetworkX 1.9.0, Apache Hadoop 1.2.1 and the Pivotal Greenplum Database, version 4.2.2.

B. Datasets

The real graphs used in the tests were obtained from the Stanford dataset [16] and are shown in Table II.

Graph	Vertices	Edges	Diameter
Cahepth	9.877	51.971	17
Cahepph	12.008	237.01	13
Astroph	18.772	396.161	14
Com-amazon	334.863	925.872	44

Table II. Real graphs used in the experiments (Standford Dataset [16]).

Synthetic graphs are divided in two groups. The first group contain graphs with a fixed number of vertices and varying number of edges, and is presented in Table III. The second group has graphs with varying number of vertices and edges and is shown in Table IV.

Table III. Synthetic graphs with 20,000 vertices and varying number of edges.

Edges	Diameter
200.000	7
300.000	6
400.000	5
500.000	5

Table IV. Synthetic graphs used in the experiments with varying number of vertices and edges.

Vertices	Edges	Diameter
10.000	100.000	4
20.000	200.000	6
30.000	300.000	8
40.000	400.000	11
50.000	500.000	13

C. Data distribution

The test graphs had to be loaded on each of the 3 platforms compared. The simplest case was Neo4J: since we only used one computer to process the graphs they were loaded from a file containing vertices and edges.

For Greenplum, the table containing the graph was sharded among all nodes in the cluster using a simple hash function aimed at guaranteeing that data about a single vertex was located on the same node. Hadoop, on the other hand, offers less control about how data is divided. The graph, represented by a file, is loaded into HDFS which then divides it into blocks that are replicated and spread among nodes in the cluster.

VII. Results

A. Real graphs

The best execution times for real graphs are shown in Figure 7. These results were obtained in the Paradent cluster using all 45 computers for both Greenplum and Hadoop and a single computer for Neo4J.

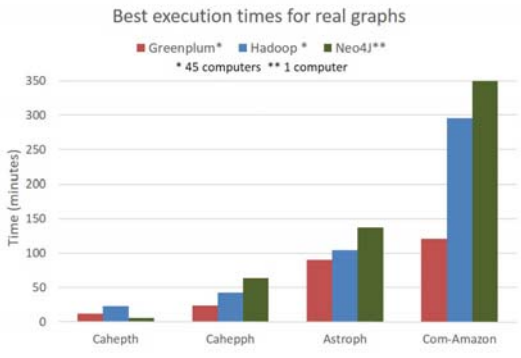


Figure 7. Execution times for real graphs

The performance of Neo4J was better for the smaller graphs and degraded for the larger graphs. Greenplum outperformed Hadoop in all cases, and was almost 2,5 times faster for the Com-Amazon graph.

Figure 8 shows the execution times for Hadoop and Greenplum when increasing the number of computers used. The behavior is similar for all graphs: for smaller numbers of computers Hadoop produces better execution times, while Greenplum produces better results when the number of computers available is larger.

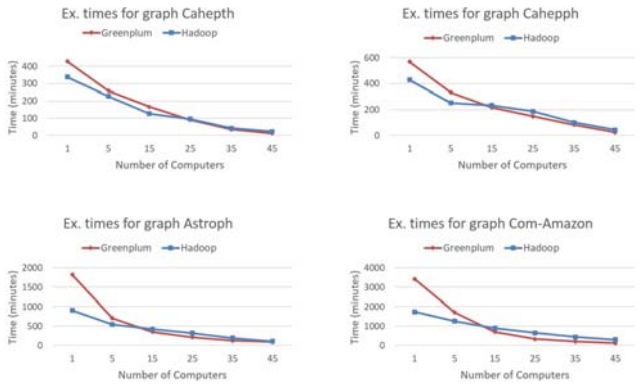


Figure 8. Execution times for real graphs with Greenplum and Hadoop when varying the number of available computers.

Figure 9 shows the relative speedup for the Hadoop and Greenplum execution times for real graphs as the number of computers grow. The relative speedup is calculated dividing the execution time obtained with N computers by the execution time of the same solution when using a single computer.

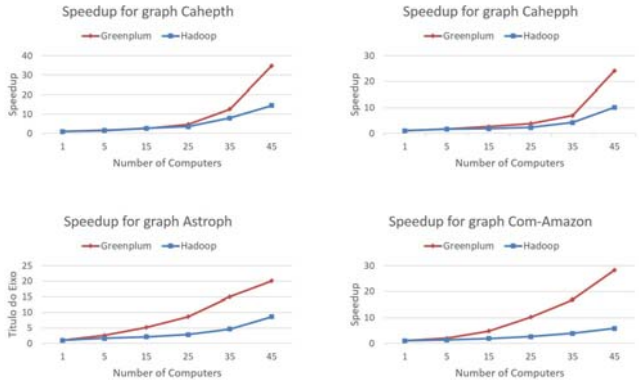


Figure 9. Speedup for real graphs with Greenplum and Hadoop when varying the number of available computers.

Both speedup curves have similar shapes, increasing as the number of available computers grew. However, speedup numbers for Greenplum are clearly superior, reaching nearly 30 with 45 computers for the Com-Amazon graph.

B. Synthetic graphs

All synthetic graphs were processed on cluster graphene using all 120 computers available. Figure 10 shows the execution times for the first set of graphs, all with 20,000 vertices and varying number of edges.

As the number of edges increase there are two opposing forces affecting the execution time. On one side, the larger number of edges means more processing to be done to analyze the whole graph, while at the same time the diameter of the graph decreases and therefore less iterations are needed to reach the end of the algorithms.

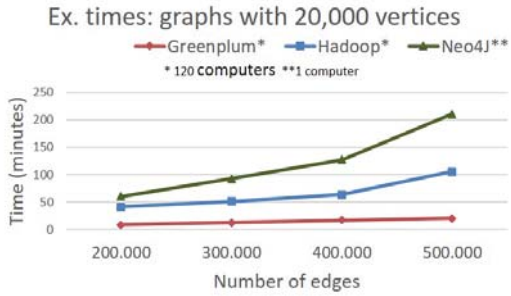


Figure 10. Execution times for synthetic graphs with 20,000 vertices and varying number of edges.

As expected, Neo4J had the worst results, unable to take advantage of all the computing power available. However, it is worth noticing that Neo4J produced results that were about twice the execution times obtained by Hadoop, a remarkable fact considering it only used one computer. Greenplum had the best results and seemed to be less affected by the increasing number of edges, maintaining its execution times nearly constant while those of Neo4J and Hadoop increased.

The speedup values corresponding to the execution times in Figure 10 ares shown in Figure 11.

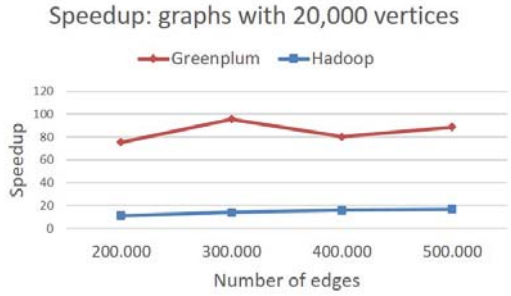


Figure 11. Speedup for synthetic graphs with 20,000 vertices and varying number of edges.

The speedup values for Greenplum were considerably better than Hadoop, staying around 80 and reaching nearly 100 for the graph with 300,000 vertices. Speedup for Hadoop remained nearly constant for all graphs.

The second group of synthetic graphs was built by varying the number of vertices and edges. Unlike the first group, the larger graphs in this group had larger diameters, which meant a larger number of iterations to calculate it. Furthermore, the work to be done at each iteration was larger as well, as a consequence of increasing the number of vertices and edges.

Figure 12 shows the execution times for graphs in the second group. To simplify presentation, graphs are named according to their number of vertices and edges, so that $Graph10$ represents the graph with 10,000 vertices and 100,000 edges and so on.

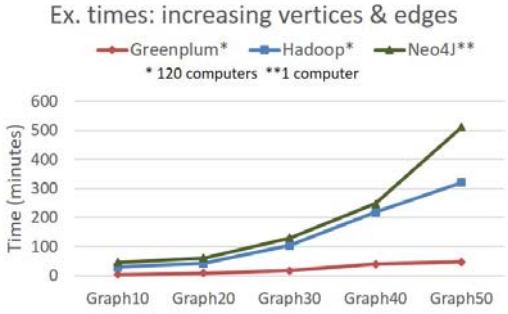


Figure 12. Execution times for synthetic graphs with varying number of vertices and edges.

Again, Neo4J had the worst results but took approximately only twice as long as Hadoop, that used 120 computers for the task. Execution times for Greenplum increased only slightly for the larger graphs, and were remarkably better than Hadoop's.

Speedup values for the results in Figure 12 are shown in Figure 13.

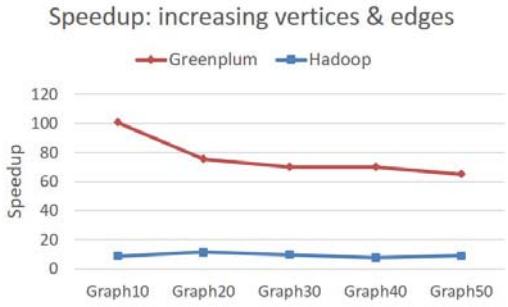


Figure 13. Speedup for synthetic graphs with varying number of vertices and edges.

As expected, speedup values for Greenplum were better than Hadoop. However, Greenplum had better speedup results for the smaller graphs. As the graphs grew larger the speedup values decreased, presumably because a smaller part of the processed data could fit into memory.

C. Analysis

The tests conducted prove that both Hadoop and Greenplum, being able to exploit the computing power available, produce better execution times than Neo4J, which was confined to a single computer. This is hardly a surprise: however, it is worth noticing that the execution time of Neo4J were remarkably close to those of Hadoop, taking about twice as long even when Hadoop used 120 computers and Neo4J was executed on a single computer.

Greenplum produced the best results, with speedups at about 2/3 of the optimum value and execution times more than 10 times faster than Neo4J. Perhaps counterintuitively, Hadoop performed better than Greenplum only in tests with a smaller number of computers, while Greenplum seemed to be better at handling effectively a larger number of nodes.

There are no doubts that a graph database such as Neo4J is better at handling graphs, as they should. This fact was clearly demonstrated by the simple and intuitive algorithm and data models necessary to calculate the graph diameter and the excellent performance results obtained with a single computer.

However, the tests conducted in this paper indicate that, when better performance is required, a parallel relational database such as Greenplum should be considered as a solution before MapReduce does. Greenplum was able to produce shorter execution times while still ensuring fault tolerance and data integrity. MapReduce implementations such as Hadoop seem to be useful when cost is an issue and/or when "quick and dirty" analysis are required for which purchasing, installing and configuring a parallel database would be considered an overkill [26].

VIII. Conclusions

This paper presented a set of experiments aimed at understanding the performance characteristics of a parallel relational database, a MapReduce implementation and a graph database in the context of graph processing. Since there are a considerable number of options and variations of each model, we selected a popular representative of each model and used it in the comparison: Greenplum (parallel relational database), Hadoop (MapReduce) and Neo4J (graph database).

The problem selected was the calculation of the diameter of a graph, since we felt it was demanding enough to shed light on the strengths and weaknesses of each system. Tests conducted with both real and synthetic graphs revealed that Greenplum had the best execution times and speedup values, followed by Hadoop and Neo4J in that order. However, it is worth noticing that although Neo4J does not support the parallel execution of single queries its execution times were only about twice as large as those of Hadoop, even when 120 computers were used by the latter.

We conclude that when performance is an issue, a parallel relational database such as Greenplum is a better choice than a MapReduce solution, unless other concerns such as cost or setup time favors a cheaper and quicker solution with MapReduce. Neo4J combines simple, intuitive and easy to maintain programming and data models with excellent performance results, even with its lack of support for parallelism.

References

[1] A. Alexandrov, S. Ewen, M. Heimel, F. Hueske, O. Kao, V. Markl, E. Nijkamp, and D. Warneke. Mapreduce and PACT - comparing data parallel programming models. In *Datenbanksysteme für Business, Technologie und Web (BTW), 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 2.-4.3.2011 in Kaiserslautern, Germany*, pages 25–44, 2011.

[2] S. Babu and H. Herodotou. Massively parallel databases and MapReduce systems. *Foundations and Trends in Databases*, 5(1):1–104, 2013.

[3] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, Nov. 2006.

[4] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. HaLoop: Efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, Sept. 2010.

[5] R. Chen, X. Weng, B. He, and M. Yang. Large graph processing in the cloud. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 1123–1126, New York, NY, USA, 2010. ACM.

[6] S. Das and S. Chakravarthy. Challenges and approaches for large graph analysis using Map/Reduce paradigm. In V. Bhatnagar and S. Srinivasa, editors, *Big Data Analytics*, volume 8302 of *Lecture Notes in Computer Science*, pages 116–132. Springer International Publishing, 2013.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[8] N. Developers. Neo4J. *Graph NoSQL Database [online]*, 2012.

[9] D. J. DeWitt and J. Gray. Parallel database systems: The future of database processing or a passing fad? *SIGMOD Rec.*, 19(4):104–112, Dec. 1990.

[10] B. Elser and A. Montresor. An evaluation study of BigData frameworks for graph processing. In *Big Data, 2013 IEEE International Conference on*, pages 60–67. IEEE, Oct. 2013.

[11] J. Han, E. Haihong, G. Le, and J. Du. Survey on NoSQL database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pages 363–366. IEEE, Oct. 2011.

[12] T. Kajdanowicz, P. Kazienko, and W. Indyk. Parallel processing of large graphs. *Future Generation Computer Systems*, 32:324–337, Mar. 2014.

[13] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. HADI: Mining radii of large graphs. *ACM Trans. Knowl. Discov. Data*, 5(2), Feb. 2011.

[14] U. Kang, C. E. Tsourakakis, and C. Faloutsos. PEGASUS: A Peta-Scale graph mining system implementation and observations. In *ICDM 2009. Ninth IEEE International Conference on Data Mining*, pages 229–238. IEEE, Dec. 2009.

[15] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[16] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2015.

[17] J. Lin and M. Schatz. Design patterns for efficient graph algorithms in MapReduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, MLG '10, pages 78–85, New York, NY, USA, 2010. ACM.

[18] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. GraphLab: A new framework for parallel machine learning, June 2010.

[19] J. P. Nascimento and C. Murta. Um algoritmo paralelo em hadoop para cálculo de centralidade em grafos grandes. In *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2012.

[20] M. F. Pace. BSP vs MapReduce. *Procedia Computer Science*, 9:246–255, 2012.

[21] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[22] S. J. Plimpton and K. D. Devine. MapReduce in MPI for large-scale graph algorithms. *Parallel Computing*, 37(9):610–632, Sept. 2011.

[23] L. Qin, J. X. Yu, L. Chang, H. Cheng, C. Zhang, and X. Lin. Scalable big graph processing in MapReduce. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 827–838, New York, NY, USA, 2014. ACM.

[24] B. Shao, H. Wang, and Y. Li. Trinity: A distributed graph engine on a memory cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 505–516, New York, NY, USA, 2013. ACM.

[25] B. Shao, H. Wang, and Y. Xiao. Managing and mining large graphs: Systems and implementations. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 589–592, New York, NY, USA, 2012. ACM.

[26] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and parallel DBMSs: Friends or foes? *Commun. ACM*, 53(1):64–71, Jan. 2010.

[27] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, Aug. 1990.

[28] Y. Zhang, Q. Gao, L. Gao, and C. Wang. iMapReduce: A distributed computing framework for iterative computation. *Journal of Grid Computing*, 10(1):47–68, Mar. 2012.

Scalability analysis of Hash Distributed A* on commodity cluster: results on the 15-puzzle problem

Victoria Sanz (1,2), Armando De Giusti (1,2) and Marcelo Naiouf (1)
(1) III-LIDI. School of Computer Science, UNLP, Argentina
(2) CONICET. Ministry of Science, Technology and Productive Innovation, Argentina
Email:{vsanz,degiusti,mnaiouf}@lidi.info.unlp.edu.ar

***Abstract*—The A* algorithm is generally used to solve combinatorial optimization problems, but it requires high computing power and a large amount of memory. In this sense, Hash Distributed A* (HDA*) parallelizes A* in order to benefit from the computing power and the accumulated memory provided by clusters. The parallelization is done by applying a decentralized strategy and using a hash function to distribute nodes among processes. In this paper, we present a detailed implementation of HDA* using MPI, which includes a parameter that determines the number of nodes to be computed by each process per iteration of the algorithm. The experimental work is carried out on a commodity cluster, using the 15-puzzle as study case. Our experimental results reveal that the included parameter favors performance. Finally, we present a performance analysis of HDA*, as the problem size and the number of processors increase, which indicates that the algorithm scales well.**

Keywords:Hash Distributed A*, Commodity Cluster, Scalability, 15-Puzzle.

I. Introduction

In the area of Artificial Intelligence, heuristic search algorithms are used as the basis to solve combinatorial optimization problems that require finding a sequence of actions that minimize a goal function and allow transforming an initial configuration (which represents the problem to be solved) into a final configuration (which represents the solution).

One of the most widely used search algorithms for that purpose is A* [1], [2], a variant of *Best-First Search*, which browses the graph that represents the state space of the problem using a cost function $\hat{f}$ to value the nodes, in order to process the most promising paths first. To that end, function $\hat{f}$ contains known cost information of the path from the initial node s to the current node n ($\hat{g}$), as well as heuristic information to estimate the unknown cost of the path that goes from the current node n to the solution node t ($\hat{h}$), which can never overestimate the actual cost. The algorithm is different from conventional methods because the search tree is implicit and dynamically generated. During the process, it keeps two data structures: one for the unexplored nodes sorted by function $\hat{f}$ (*open list*), and another for the already explored nodes (*closed list*) used to avoid processing the same state multiple times. In each iteration, the most promising node (according to $\hat{f}$) available on the open list is removed, it is added to the closed list, and legal actions are applied to it to generate successor nodes that will be added to the open list under certain conditions (verification known as *duplicate detection*). The search process continues until the node that represents the solution is removed from the open list.

The major drawback of A* is that it requires high computing power and a large amount of memory, as a consequence of the exponential or factorial growth of the state space of the problem. Therefore, over the last decades, the development of parallel A* algorithms has been promoted which, in particular, may benefit from the computing power and the accumulated memory provided by clusters.

In this sense, Hash Distributed A* (HDA*) [3] parallelizes A* by applying a *decentralized strategy* (i.e. each processor has its own open and closed lists and performs a quasi-independent search) and using a hash function to assign each state of the problem to a single processor. In this way, when a processor generates a node, it determines who the owner is and transfers the node to that owner. This mechanism allows balancing the workload and pruning *duplicate nodes* (i.e. nodes representing the same state) in an *absolute* way, as they are always sent to the same processor. The implementation of HDA* proposed by these authors uses MPI and asynchronous communication. Although they carried out an extensive experimental work on an *HPC cluster* with Infiniband interconnection, for applications with different heuristic computation time such as the domain-independent planning and the Sliding Puzzle, the tests carried out on a conventional cluster with Ethernet connection did not take into account the latter application.

In this paper, we present a detailed implementation of HDA*, which includes the *LNPI* (Limit of Nodes per Iteration) parameter that determines the number of nodes to be computed by each process per iteration of the algorithm. In this sense, our version differs from the original algorithm, since in the latter each process computes a single node per iteration. The implementation was carried out using MPI and the 15-Puzzle was selected as study case. The experimental work is focused on analyzing the speedup and efficiency obtained by the parallel algorithm when it is run on a cluster of multi-core machines connected through Ethernet (a *commodity cluster*), as the problem size and the number of processors increase. Our experimental results reveal that the included parameter favors performance. Finally, this analysis shows that the implemented version scales well.

ISBN: 1-60132-444-8, CSREA Press ©

II. Related work

So far, different authors have presented various techniques to parallelize *A**, which vary in the method used to manage the open and closed lists and the strategy used to balance load among processors during the execution. The technique to be chosen will depend on the architecture and the problem to solve [4].

The commonly used parallelization technique is known as *decentralized strategy* [5], and it is based on the following: each processor has its own local open and closed lists, and carries out a quasi-independent search. This strategy is suitable both for shared memory and distributed memory architectures. However, communication among processors is needed due to the following reasons: the search tree is generated at run time, therefore, the workload should be distributed dynamically; *duplicate nodes* can be generated by different processors and should be pruned in order to prevent processors from performing redundant work (it is possible to achieve an *absolute* duplicate detection and pruning by using a hash function that assigns each state to a particular processor); the termination criterion should be modified because if the search is ended when a solution is found, there will be no guarantee that such solution is the best one; the costs of the partial solutions found so far should be communicated in order to use them to prune the paths that lead to suboptimal cost solutions.

The earliest parallel A* algorithms based on the *decentralized strategy* used receiver/sender initiated load balancing algorithms [4], [5], [6], [7]. However, those techniques cause duplicate nodes to arise on the open or closed lists of different processors in the system, because they usually involve carrying out a *partial* duplicate detection and pruning, i.e., that procedure is performed only by the donor processor and the recipient processor. Therefore, redundant work is carried out by different processors, which in turn increases the *Search Overhead*[1] and the amount of RAM consumed by the parallel algorithm, situation that is even worse as more processors are used. Consequently, an *absolute* duplicate detection and pruning procedure should be applied to obtain higher performance.

In this sense, the Hash Distributed A* algorithm (HDA*) [3] parallelizes A* by applying a *decentralized strategy* and using *Zobrist's hash function* [8] to assign each state of the problem to a single processor. Thus, when a processor generates a node that does not belong to it, it determines who the owner is and transfers the node to that owner. This mechanism allows balancing the workload and pruning duplicates in an *absolute* way, as the nodes representing the same state are always sent to the same processor.

The implementation of HDA* proposed by these authors uses MPI and asynchronous communication, so the algorithm is suitable for execution both on distributed memory and shared memory architectures. To avoid congestion in the communication medium caused by messages being sent containing a single node, the algorithm uses the technique proposed in [9], which involves packing in one message a given number of nodes whose recipient is the same before sending the message (in this paper, we refer to this value as *LNPT*, or *Limit of Nodes per Transfer*).

The experimental work carried out by these authors uses the domain-independent *Fast Downward planner* [10], placing HDA* as the search mechanism. On the other hand, it uses the *24-Puzzle* problem, a specific application where processing a state is faster, together with a *disjoint pattern database* heuristic [11], [12], but they exclude the time required for reading these data from the disc. The authors note that the speed of processing a state significantly affects the *efficiency* of the parallel search algorithm: when processing a state is expensive, the impact of parallelization-related overheads, such as communication and synchronization, tends to decrease. For this reason, studying the efficiency achieved by HDA* for various types of applications running on different architectures is of interest.

Consequently, the authors analyze the speedup and efficiency achieved by HDA* on a single, 32GB RAM multicore machine, for planning problems. However, they do not assess performance for the *24-Puzzle* since the instances they used exhaust RAM before finding a solution.

Additionally, the authors study the scalability of HDA* for the applications mentioned above on a multicore cluster with Infiniband interconnection (*HPC cluster*). They show that HDA* achieves good performance and scales relatively well for complex planning problems that require large amounts of RAM. On the other hand, the performance obtained for the *24-Puzzle* application is not as satisfactory as in the previous case, and it rapidly degrades when increasing the number of cores in the architecture.

Similarly, they assess the scalability of HDA* for planning applications on a multicore cluster with Ethernet connection (*commodity cluster*), obtaining an almost linear relative speedup. However, they do not evaluate the performance on this architecture for the *24-Puzzle*.

HDA* is currently interesting due to its simplicity and good scalability. On the other hand, the Sliding Puzzle has recently gained relevance because it is related to real problems such as moving pallets with an automated guided vehicle in high-density storage warehouses [13]. Also, research centers usually have clusters formed by connecting multicore machines through conventional networks such as Ethernet. It is for all these reasons that the study of HDA* behavior on this type of clusters for solving the Sliding Puzzle is an open research line.

III. Hash Distributed A*

Hash Distributed A* (HDA*) [3] parallelizes A* based on a *decentralized strategy*. It was programmed using exclusively the MPI message passing library and asynchronous communication; therefore, it is suitable for execution both on distributed

[1]The Search Overhead represents the percentage of nodes that the parallel algorithm expands in excess as compared to the sequential algorithm. It is calculated using the formula 100x(NP/NS -1), where NP is the number of nodes processed by the parallel algorithm and NS is the number of nodes processed by the sequential algorithm.

memory as well as shared memory architectures. It uses a *hash function* to assign states to processes, thus it implicitly achieves load balancing and absolute duplicate detection (two nodes representing the same state are sent to the same process, which in turn checks for duplicates in its local structures).

Initially, the relevant process adds the initial node to its open list. Then, each process carries out iterations until a global optimal solution is reached. In each iteration:

1) It checks if one or more nodes have been received through messages. If so, for each node received, it carries out the *duplicate detection* to determine if the node must be added to the open list or if it should be discarded.
2) If no messages containing nodes were received, the process selects a node from its open list (the one with the lowest $\hat{f}$-value) and expands it, generating successor nodes. Then, for each successor, it calculates the *hash value* to identify the *owner process* and, if the node belongs to another process, it sends asynchronically the node to its owner through a message.

To reduce the communication overhead, the idea proposed in [9] is used, which involves packing within a single message a given number of nodes whose recipient is the same (in this paper, this number is referred to as *Limit of Nodes per Transfer*, or *LNPT*). The number of nodes to be packed depends on factors such as communication medium speed and number of processors, among others.

To obtain a uniform distribution of nodes, which is necessary to achieve an effective load balancing, *Zobrist's hash function* [8] is used. On the other hand, to detect the end of the computation, i.e., to know the state in which all processes are idle and there are no messages in transit, *Mattern's time algorithm* [14] is used.

IV. Implementation of Hash Distributed A*

We developed our own version of the HDA* algorithm, which is described in this section. For its implementation, the following tools were used: MPI; asynchronous communication; non-blocking query for a message's arrival (MPI_Iprobe) when the process is not idle, blocking query (MPI_Probe) otherwise; the termination detection algorithm proposed by Dijkstra and Safra [15], since we opted for not increasing the size of each *work message* sent with additional information[2]; *Zobrist's hash function* to assign nodes to processes; and the technique proposed in [9] to package a given number of nodes (*LNPT*) before sending the *work message* to its recipient process.

Each process carries out an A* search locally and communicates with its peers for any of the following reasons: sending/receiving *work messages* containing nodes, sending/receiving the *costs of solutions* found, sending/receiving the *termination token*, sending/receiving *termination notification* messages.

Locally, each process has its open and closed lists, which will be empty at first, the cost of the best global solution known so far (*best_solution_cost*), the best solution found by the process (*best_solution*), the data required by the termination detection algorithm, and a variable that changes its value when the computation reaches its end (*end*). For the purpose of packing several nodes in a message before sending them to their owner, processes are equipped with a buffer for each peer process (*send_buffer*); each buffer will contain node records, its physical dimension is known as *LNPT* (Limit of Nodes per Transfer), and its logical dimension must be kept updated to know the number of nodes that have been accumulated.

Although the code is the same for all processes, process 0 is in charge of: loading the initial configuration; generating the initial node, which will be added to the open list of this process (if the node belongs to it) or which will be sent to the corresponding owner process; and carrying out the termination detection, which involves sending the termination notification to the other processes.

Each process performs a series of iterations until receiving the termination notification; at that moment the optimal solution has been reached. In each iteration, the following stages are carried out:

1) *Work message reception stage*: the process checks, in a non-blocking manner (MPI_Iprobe), if *work messages* containing nodes have arrived. If so, it receives each message and, for each node record whose cost is lower than *best_solution_cost*, it carries out the following actions: it allocates space in dynamic memory; it assigns the record received to the allocated space; it performs the duplicate detection adding the node to the open list as appropriate.
2) *Cost message reception stage*: the process checks, in a non-blocking manner (MPI_Iprobe), if *cost messages* containing the cost of a better solution found have arrived. If so, it receives the messages and updates the local variable *best_solution_cost* as appropriate. Now, if the cost of the best open node (according to function $\hat{f}$) is at least *best_solution_cost*, the process empties its open list since the nodes stored in it would lead to suboptimal solutions.
3) *Processing stage*: the process expands at most *LNPI* (Limit of Nodes per Iteration) nodes from its open list. For each extracted node, the process checks if its cost is at least *best_solution_cost*. If so, the process empties the open list. Otherwise, it checks if the node represents the solution and in that case it empties the open list and updates *best_solution* and *best_solution_cost*. When the extracted node is not the solution, it is added to the closed list, it is expanded (i.e., *successor nodes* are generated) and then, for each successor, the *Zobrist function* is calculated to determine its corresponding

[2]This algorithm is similar to Mattern's time algorithm [14], [16]. Both are based on the same idea of counting messages. The main difference between them is that Dijkstra and Safra's algorithm is based on the *double wave* approach, whereas Mattern's time algorithm is based on the *single wave* approach (at the expense of increasing the amount of control information or augmenting every message with a time stamp).

owner process. When the successor belongs to this very process, it carries out the duplicate detection and adds the node to its open list as appropriate. Otherwise, the process adds the node record to the *send_buffer* for the destination process and, if that buffer is full (i.e., it already has *LNPT* nodes), it sends the *work message* asynchronically.

4) *Idle stage*: the process goes into this stage when its open list is empty. If a new solution was found in the *processing stage*, the process sends the solution cost to its peers. It then sends *work messages* to those destination processes whose *send_buffer* contains nodes that were not communicated. Finally, it remains waiting (MPI_Probe) for any type of message: (1) work message, (2) cost message (3) token message, (4) termination notification message. The process ends this stage when it has nodes on its open list, as a result of having received a work message, or when it receives the termination notification message. Messages of types (1) and (2) are processed in a similar way as described above; messages of the type (3) are processed based on the termination detection algorithm (knowing that the process is idle and it must send the token to the next process or assess the termination condition in the case of Process 0); messages of the type (4) are processed by changing the value of variable *end*, and by doing so, algorithm ends.

Each *work message* that is sent or received is processed as indicated by the termination detection algorithm. Three additional fields are added to the token message to make solution retrieval possible: cost of the best global solution found so far, ID of the process that found that solution, and memory address for the solution node. Before sending the token, the process must update these fields with its own information if it has the best solution so far.

When computation ends, the solution is retrieved in a distributed manner, obtaining the sequence of actions that allows transforming the initial state into the final state.

V. Experimental results

Experimental tests were carried out on a cluster composed by 7 machines connected through 1Gb Ethernet. Each machine has two Intel Xeon E5620 processors and 32GB RAM. Each processor has four 2.4Ghz physical cores. Each processor has a memory controller and uses a 5.86 GT/s *QPI* connection.

A* and HDA* were configured to use the *Jemalloc* memory allocator (with 256 arenas) and a heuristic that is a variation of the *Sum of Manhattan Distances (SMD)* of the tiles with the addition of linear conflicts detection, the detection of the last moves applied and an analysis of corner tiles [17]. We showed in [18] that this configuration improves the performance of A* versus using the default memory allocator (*Ptmalloc*) and the *SMD* heuristic.

A* was run on a single machine of the previous cluster.

HDA* was run on various cluster configurations, i.e., varying the number of machines used between 2 and 7. For all tests, 4 processes were assigned to each machine used[3] (two processes per processor in the machine). The parameters and values used were: number of processes/cores (8, 12, 16, 20, 24, 28), LNPI (1, 5, 50, 500) and LNPT (26, 210, 1680). The LNPT values correspond to work messages whose size is 1KB, 8KB and 64KB, respectively.

The tests considered the 15-Puzzle instances used in [19] whose sequential execution time is of at least 5 seconds (numbered 3, 15, 17, 21, 26, 32, 33, 49, 53, 56, 59, 60, 66, 82, 88, 100) and six of the 10 configurations that are part of the test suite proposed in [20] (numbered 101-106 in this paper). Thus, the 22 configurations used present different levels of complexity, which is measured in terms of the number of nodes processed by A*, and varies between 1 and 103 million processed nodes.

We have selected the instances mentioned above because they are solvable on a single 32GB RAM machine. Serial runtimes can not be measured for hard problems, for example *Korf's 24-puzzle instances*, using our architecture because A* exhausts RAM before finding a solution. The same problem arises when running HDA* on a single 32GB RAM multicore machine for those instances [3]. While it may be possible to solve some *Korf's 24-puzzle instances* on our cluster and measure *relative* speedup and efficiency as in [3], as future work we intend to analyze the scalability of our version of HDA*, implemented using MPI, on a multicore machine for the Sliding Puzzle problem, and to compare the results achieved with those presented here and those obtained by our optimized version of HDA* for multicore machines, implemented using Pthreads [18], [21].

HDA* is non-deterministic, i.e., when multiple runs are carried out using the same input data (initial and final states), the results generated by the algorithm may be different. For this reason, 10 tests were carried out for each cluster configuration, initial configuration and set of parameters. The data obtained with these 10 runs were then averaged, which will be referred to as *mean sample*.

In the following sections, we analyze the impact of the LNPI and LNPT parameters on the performance of HDA*, and then we assess its scalability when the parameters values that experimentally improved performance are used.

A. Limit of Nodes per Iteration (LNPI)

The LNPI parameter determines how many nodes a process must expand per algorithm iteration, i.e., it establishes the interval for checking the arrival of *work messages* and *cost messages*.

To analyze the impact of this parameter on execution time, all *mean samples* obtained from runs carried out for LNPT=26 (i.e., 1KB *work messages*) were taken; and those with the same initial configuration and number of cores were grouped. That

[3]It was observed that when 8 processes per machine are used, the performance obtained is poor. This is because each machine has a single network controller and therefore bottlenecks are caused by network I/O; this is even worse because services communicate through the same network.

TABLE I
SD AND CV RANGES SORTED BY LNPT

LNPT	SD range (seconds)	CV range
26	0.063-18.94	0.054-1.28
210	0.04-9.43	0.054-0.24
1680	0.12-9.45	0.07-0.17

is, each group contained the *mean samples* whose only difference among them was the LNPI parameter value. For each group, *Standard Deviation* (SD) and *Coefficient of Variation* (CV) of execution time were calculated, since these values measure how much the execution time of a *mean sample* in the group tends to deviate from the *group mean time*.

From the data obtained for LNPT=26, it was observed that only 60% of the groups have SD values below 1, and 39% below 0.5. Similarly, only 11% of the groups have CV values below 0.1.

The procedure described above was also applied to *mean samples* obtained from runs carried out for LNPT=210 and LNPT=1680 (i.e., 8KB and 64KB *work messages*, respectively).

Table I shows the ranges of SD and CV values for the groups, sorted by the LNPT value. As it can be seen, when *work messages* contain few nodes (small LNPT), the LNPI value has a greater impact on execution time, since there is a greater difference in the execution times of the *mean samples* in each group (greater CV). The results obtained indicate that there is a significant variation in execution time when changing the value of the LNPI parameter among those defined (1, 5, 50 or 500), mainly for LNPT=26 and LNPT=210.

The LNPI value that improves performance depends on the initial configuration, the number of processes and the LNPT value.

For LNPT=26, the LNPI value that improves overall performance is 50. The configurations that presented improvements with LNPI=500 are some of the most complex ones (60, 88, 105, 106 and 104); however, as the number of cores increases, the number of configurations that favored such value decreased. The configurations that presented improvements with LNPI=5 are some of the less complex ones (3, 21 and 49) with a large number of cores.

For LNPT=210 and LNPT=1680, performance improves with LNPI=50 or LNPI=500. In contrast with the previous case, there is no clear preference for either of these two values.

From the data presented above, the following can be concluded:

1) When LNPT is small, processes will send numerous *work messages* containing few nodes. If LNPI is set to a very high value, processes will carry out too much speculative work on their local nodes in each *processing stage*, without adding newly arrived nodes, increasing the *Search Overhead*, which is even worse as the number of processes increases. Therefore, the frequency of checking for message's arrival has to be increased (low LNPI) to allow the addition of nodes that are among the global best ones.
2) When LNPT is large, processes will send few *work messages* containing many nodes. If LNPI is set to a high value, the number of checks for message's arrival, that are likely to fail, decreases. Otherwise, if LNPI is set to a low value, performance is not affected that much because non-blocking checks are used. No significant variations in *Search Overhead* are observed with LNPI changes.

It should be noted that setting LNPI to 1 never resulted in improved performance. This indicates that the parameter that was added to our own version of HDA* is indeed relevant and an original contribution in the area.

B. Limit of Nodes per Transfer (LNPT)

The LNPT parameter indicates the maximum number of nodes that will be included in each *work message*.

For the purpose of analyzing the effect of the LNPT parameter on execution time, all *mean samples* obtained from runs limiting LNPT to 26 nodes (1KB messages), 210 nodes (8KB messages) and 1680 nodes (64KB messages) were taken. Then, all *mean samples* with identical initial configuration, number of cores and LNPI values were grouped; i.e., each group contained *mean samples* whose only difference was the value for their LNPT parameter. For each group, *Standard Deviation* (SD) and *Coefficient of Variation* (CV) of execution time were calculated, since these values measure how much the execution time of a *mean sample* in the group tends to deviate from the *group mean time*.

In general, the results obtained show that the SD for the groups is between 0.045 and 24.37. This indicates that when varying the LNPT parameter between the values defined, *mean samples* execution times tend to deviate at most in 24.37 seconds from their *group mean*. It should be noted that 46.21% of the groups has a SD value below 1, and 23.10% has a SD value below 0.5.

On the other hand, the general CV values obtained for the groups range from 0.019 to 1.15; i.e., the execution time of a *mean sample* in the group tends to deviate between 1.9% and 115% from the *group mean* when the LNPT parameter changes between the values defined. Only 24.4% of the groups have CV values below 0.1.

Based on the high CV values, it is concluded that the LNPT parameter affects execution time. This is because this parameter impacts network traffic, process activity and volume of speculative work carried out. The following can be inferred:

1) A low value of LNPT will cause processes to generate *small work messages*, which increases the number of work messages that are sent from one process to another over the network using MPI, which in turn generates an overhead for handling buffers associated to communications and network congestion.
2) Very high values of LNPT can also degrade performance because, if a process packs too many nodes for a recipient process, it could cause the latter's idleness when it

TABLE II
SPEEDUP AND EFFICIENCY FOR EACH LNPT VALUE AND NUMBER OF PROCESSES

LNPT	Processes/Cores	Speedup	Efficiency
26	8	4.72 - 6.58	0.59 - 0.82
210	8	5.05 - 7.53	0.63 - 0.94
1680	8	3.20 - 8.26	0.40 - 1.03
26	12	6.74 - 10.31	0.56 - 0.86
210	12	6.85 - 11.79	0.57 - 0.98
1680	12	2.90 - 11.35	0.24 - 0.95
26	16	8.52 - 13.73	0.53 - 0.86
210	16	7.16 - 15.62	0.45 - 0.98
1680	16	2.15 - 14.70	0.13 - 0.92
26	20	11.07 - 17.76	0.55 - 0.89
210	20	8.57 - 20.22	0.43 - 1.01
1680	20	2.11 - 19.40	0.11 - 0.97
26	24	12.73 - 21.23	0.53 - 0.88
210	24	8.92 - 24.27	0.37 - 1.01
1680	24	1.72 - 22.71	0.07 - 0.95
26	28	13.49 - 25.16	0.48 - 0.90
210	28	8.60 - 28.86	0.31 - 1.03
1680	28	1.61 - 27.05	0.06 - 0.97

has only few nodes to process or when it is currently idle due to lack of work. The process could also delay the transmission of higher quality nodes (according to $\hat{f}$) than those that are currently being processed by the recipient, which would increase the *Search Overhead*.

Table II summarizes the ranges for Speedup and Efficiency, sorted by LNPT value and number of processes/cores. For each LNPT, initial configuration and number of cores, the *mean sample* with the best performance was selected (i.e., that whose LNPI value reduces execution time).

In most of the cases, the value of LNPT that improves execution times for each configuration and number of cores is 210, i.e., 8KB *work messages*. There are some exceptions with some low-complexity configurations as the number of processes scales, for which performance improved using LNPT=26, since higher values of LNPT increased *Search Overhead*. Other exceptions were observed with some of the more complex configurations when only a few processes were used; in these cases, the best performance was obtained with LNPT=1680 because it produces a lower *Search Overhead* or a lower *load unbalancing*.

The following configurations had their best performance with LNPT=26: 21 with 12 processes; 100, 21 and 101 with 16 processes; 100, 21, 101 and 82 with 20 processes; 100, 33, 21, 101 and 82 with 24 processes; 100, 33, 21, 101, 82, 59 and 53 with 28 processes. The following configurations had their best performance with LNPT=1680: 88, 102, 106 and 104 with 8 processes; 104 with 12 processes.

From the above, it can be concluded that, for less complex configurations, as the number of processors increases it is better to use *smaller work messages* (smaller LNPT). In the case of more complex configurations and only a few processes, the best performance is occasionally obtained with *larger work messages* (larger LNPT). However, empirically for this architecture, as the workload and the number of processes/cores scale, better performance is obtained when using 8KB *work messages* (i.e., LNPT= 210).

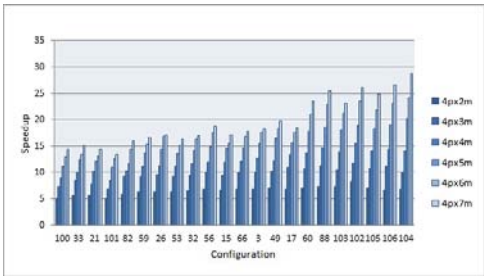


Fig. 1. Speedup obtained by HDA* on cluster

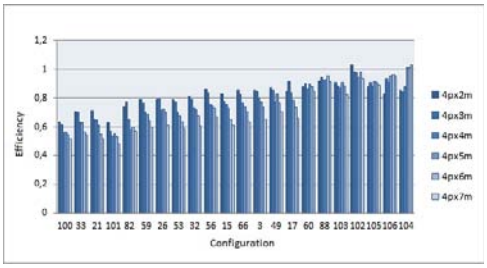


Fig. 2. Efficiency obtained by HDA* on cluster

C. Performance analysis

To analyze the performance of HDA*, for each initial configuration and number of processes, the *mean sample* that minimized resolution time was selected; that is, the sample whose LNPI and LNPT values resulted in the best performance.

To assess algorithm scalability, the various selected *mean samples* were sorted by configuration complexity (that is, based on the number of nodes processed by A*, which is related to sequential execution time). In this sense, scaling the problem means increasing the number of processed /generated nodes. On the other hand, the architecture is scaled by increasing the number of cores/processes used to solve the problem.

Figure 1 shows the speedup obtained by the *mean sample* selected for each initial configuration using 8, 12, 16, 20, 24 and 28 processes/cores (4 processes per machine), while Figure 2 shows the efficiency obtained.

After analyzing the data presented, it can be concluded that, if workload is constant (initial configuration) and the number of cores/processes is increased, speedup improves, meaning that less time is required to solve the problem. However, this improvement does usually not keep efficiency at a constant value. The decrease in efficiency is due to factors such as: sequential portions of the algorithm, particularly at the beginning and at the end of the computation, synchronization, communication, idle time, load unbalancing, increased search overhead, and so forth.

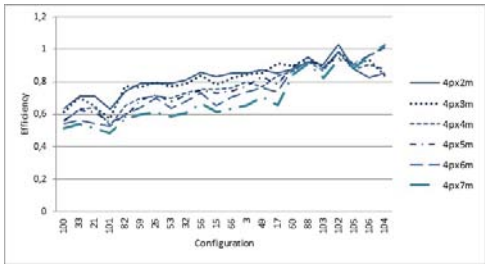


Fig. 3. Scalability for HDA* on cluster

The superlinear speedup obtained for the configuration 102 and 8 processes was due to the fact that the parallel algorithm processed a lower number of nodes than the sequential algorithm, which is possible in this type of search algorithms[4], [5]. The remaining cases are due to the decrease in the number of elements in the open-list and closed-list structures, which causes an acceleration of the operations carried out on them.

Figure 3 shows the efficiency as the workload (problem complexity) and the number of processes/cores increase. As it can be observed, when problem complexity is scaled and the number of cores used is constant, the efficiency generally improves or remains constant because the overhead is less significant for total processing time.

Based on the results presented above, it can be concluded that the behavior presented by the algorithm, when it is run on cluster configurations with 4 processes per machine, is typical of a scalable parallel system, where the efficiency can be kept at a constant value as both problem size and number of processors are increased.

VI. Conclusions and Future Work

We developed our own version of HDA*, which differs from the original version in that it includes the LNPI (*Limit of Nodes per Iteration*) parameter, which indicates the maximum number of nodes to be processed in each iteration.

The effect of the LNPI and LNPT parameters (the latter determines the size of *work messages*) on performance was analyzed. It was concluded that the LNPI value that improves performance depends on the initial configuration, the number of processes and the size of *work messages* (LNPT). It was noted that performance never improved by processing one node per algorithm iteration (LNPI=1), as done in the original version, which indicates that the parameter added to our version favors performance. On the other hand, it was established that, empirically for this architecture, as the problem size and number of processes/cores increase, better performance is obtained when using *8KB work messages* (LNPT=210).

Finally, algorithm scalability was assessed. The results obtained indicate that the parallel algorithm, if run on cluster configurations with 4 processes per machine, presents the typical behavior of scalable parallel systems.

As for future work, we intend to compare performance achieved and memory consumed by HDA* for shared-memory architectures and HDA* for distributed-memory architectures, when they are run on a multicore machine. The former algorithm was presented in [18], [21] and implemented with Pthreads, and the latter was introduced in this paper and implemented with MPI. The results will allow assessing if there are any potential benefits of converting HDA* into a hybrid application that uses programming tools for shared and distributed memory when the underlying architecture is a multicore cluster.

References

[1] Hart et al. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 1968; 4(2):100-107.

[2] Russel et al. *Artificial Intelligence: A Modern Approach* 2nd ed. Prentice Hall: New Jersey, 2003.

[3] Kishimoto et al. Evaluation of a simple, scalable, parallel best-first search strategy. *Artificial Intelligence* 2013; 195: 222-248.

[4] Kumar et al. Parallel Best-First Search of State-Space Graphs: A Summary of Results. *Proceedings of the 7th Nat. Conf. AI*. AAAI:1988; 122-127.

[5] Grama et al. *Introduction to Parallel Computing* 2nd ed. Pearson: Harlow, 2003.

[6] Dutt et al. Parallel A* Algorithms and Their Performance on Hypercube Multiprocessors. *Proceedings of Seventh International Parallel Processing Symposium*. IEEE Computer Society:1993; 797-803.

[7] Sanz et al. Parallel Optimal and Suboptimal Heuristic Search on multicore clusters. *Proceedings of The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications*. CSREA Press:2011; 673-679.

[8] Zobrist. *A New Hashing Method with Application for Game Playing*. Computer Sciences Department, University of Wisconsin: Madison, 1968. Technical Report 88.

[9] Romein et al. A performance analysis of transposition-table-driven work scheduling in distributed search. *IEEE Transactions on Parallel and Distributed Systems* 2002; 13(5): 447-459.

[10] Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 2006; 26: 191-246.

[11] Culberson et al. Pattern databases. *Computational Intelligence* 1998; 14(3): 318-334.

[12] Korf. Recent Progress in the Design and Analysis of Admissible Heuristic Functions. *Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation*. Springer:2000; 45-55.

[13] Gue et al. GridStore: A Puzzle-Based Storage System With Decentralized Control. *IEEE Transactions on Automation Science and Engineering* 2014; 11(2): 429-438.

[14] Mattern. Algorithms for distributed termination detection. *Distributed Computing* 1987; 2(3):161-175.

[15] Dijkstra. *Shmuel Safra's version of termination detection*. Department of Computer Sciences, University of Texas: Austin, 1987. EWD-Note 998.

[16] Mittal et al. A family of optimal termination detection algorithms. *Distributed Computing* 2007; 20(2):141162.

[17] Korf et al. Finding Optimal Solutions to the Twenty-Four Puzzle. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* AAAI:1996; 1202-1207.

[18] Sanz et al. On the Optimization of HDA* for Multicore Machines. Performance Analysis. *Proceedings of the 2014 International conference on Parallel and Distributed Processing Techniques and Applications*. CSREA Press:2014; 625-631.

[19] Korf. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 1985; 27(1): 97-109.

[20] Brüngger. *Solving Hard Combinatorial Optimization Problems in Parallel: Two Cases Studies*. Swiss Federal Institute of Technology Zurich: Zurich, 1998. Diss. ETH No. 12358.

[21] Sanz et al. Performance tuning of the HDA* algorithm for multicore machines. *Computer Science and Technology Series.XX Argentine Congress of Computer Science. Selected Papers*. EDULP: La Plata,2015.

ISBN: 1-60132-444-8, CSREA Press ©

SESSION

SOFTWARE ENVIRONMENTS, LANGUAGES, SYSTEMS, AND SUPERCOMPUTING

Chair(s)

TBA

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

Supercomputer Reliability and Mitigation

Ron T. Ogan
Dept. of Electrical & Computer Engineering
Jackson State University
Jackson, MS. 39217 USA
ron.t.ogan@jsums.edu

Marshall D. Boyette
Dept. of Electrical & Computer Engineering
Jackson State University
Jackson, MS. 39217 USA
marshall.d.boyette@jsums.edu

Paul E. Watson
Dept. of Electrical & Computer Engineering
Jackson State University
Jackson, MS. 39217 USA
paul.e.watson@jsums.edu

Khalid H. Abed
Dept. of Electrical & Computer Engineering
Jackson State University
Jackson, MS. 39217 USA
khalid.h.abed@jsums.edu

Abstract — Reliability, Availability and Serviceability (RAS) are key to high performance computing. Because of the relatively high costs of supercomputers and the required support needed to operate these systems, a holistic approach must be taken to assure system reliability as defined: Reliability is the probability that a material, component, or system will perform its intended function under defined operating conditions for a specific period of time. Extension of the computer system lifetime will be achieved through the implementation of built-in-test-equipment (BITE) monitoring that will provide the required feedback for optimum environmental controls. Reliability improvements of MTBF of 25% have been predicted based upon redesign of the microprocessor cooling system to reduce average case temperatures from 90 °C to 75 °C

Keywords—Supercomputer, Reliability, Availability, Serviceability (RAS), thermal analysis

I. INTRODUCTION

Supercomputers, the heart of High Performance Computing Centers, are designed with the latest processor technology to provide performance and scalable systems that integrate advanced interconnections to achieve the high-bandwidth performance to achieve incredible mathematical calculation rates. Advanced design methodology is utilized to network the rack assemblies with distributed memory to minimize system latency yet provide scalability and maintainability. Incredible performance has been achieved across computing network systems with distributed memory systems to provide programmers with global access to large memory arrays for parallel processing applications to support the most demanding global communication patterns. This paper addresses technological aspects of supercomputer technology that are predominantly being operated at U.S. Federal Laboratories to solve very large complex problems efficiently. The supercomputers are housed in a large environmentally controlled workspace with monitored support for cooling and power equipment to guarantee reliability, availability and serviceability (RAS) of >99%.

High Performance Computing System components include CMOS general purpose processors (GPPs) and heterogeneous computational hardware, in particular, a field programmable gate array (FPGA) [2], memory integrated circuits, wiring, resistors, capacitors and inductors. Each of these components has a design life with resistors being the most stable and least affected by aging and temperature.

MIL-STD-217E, parts count or macro models may be used to predict computer failures based upon the operating environment. Built-in-test-equipment (BITE) circuits are often incorporated into the computer design to monitor voltages or bit errors such as parity mismatch in series data transmissions or miss-compares when co-processors do not show exact results. Failure rates as a function of time include initial failures due to manufacturing defects, constant hazard region where failures are random, and finally, the wear-out region where insulation, capacitors, and mechanical components such as fans and cooling pumps are aged beyond the useful life. Table 1 shows compute module faults that affect performance with possible design mitigation to minimize these effects on performance and module lifetime.

TABLE 1. COMPUTE MODULE FAULTS AND MITIGATION

Defect	Design Mitigation
Memory soft failure	Provide electromagnetic shielding to extent possible
Memory hard failure	Provide fault tolerant systems
Capacitor drift	Design Center to accept drift
Power supply spikes	Power conditioning
Board impedance shifts	Design Center to accept shift
Microprocessor failure	Provide optimum cooling and monitoring
Microprocessor degradation	Provide optimum cooling and monitoring

A. Abbreviations and Acronyms

CPU- Computer Processing Unit

FCBGA- flip chip ball grid array
HPC- High Performance Computers
MTBF- Mean Time Between Failures
MTTR- Mean Time To Repair
RAS- Reliability Availability and Serviceability

II. RELIABILITY MODELING

Reliability is the probability that a material, component, or system will perform its' intended function under defined operating conditions for a specific period of time. Supercomputers are constructed of many single board computer systems that are efficiently connected to form a computer array capable of achieving high rates of mathematical calculations per second. Single board computers, called Compute Modules, with multiple microprocessors have heat sinks or heat pipes to dissipate large amount of heat energy that is required to achieve the high data rates. Most systems have liquid cooling including recirculated cooling water or Freon for more efficient heat transfer. Supercomputer reliability can be determined by parts count with known failure rates, calculated using reliability models or measured experimentally. Circuit interconnections are very difficult to model since the failures may be open, short or intermittent. Failure Modes Effects Analysis (FMEA) techniques are often employed to determine components with the highest failure probability.

CPU health testing, monitors cooling water flow rates and temperature sensors that detect potential microprocessor failures to mitigate and avoid downtime from repairs, correcting temperature and humidity controls. System performance monitoring is used to collect metrics to detect system changes like duty cycles, offline checks, redundancy, overheating, solder connection failures, capacitor drift, intermittent operation, memory loss, addressing errors, board impedance changes of less than or greater than 50 ohm characteristic impedance, semiconductor junction migration, ground or via faults, power supply spurious or harmonics from switching supplies, moisture intrusion, wire bond failures, heat-sink corrosion.

Reliability, Availability, Serviceability (RAS)

Reliability is a function of time that expresses the probability at time t+1 that a system is still working, given that it was working at time t. Availability is the measure of how often the system is available for use (such as a system's up-time percentage). Availability and reliability may sound like the same thing, but it is worth noting that a system can have great Availability but no Reliability. An internet router is a good example of this; it stores no state data. It is one of the few systems wherein data loss is acceptable, as long as high availability is maintained. Availability is typically described in nines notation. For example 3-nines means 99.9%. Obtaining 5 nines or 99.999% availability is an ambitious goal for many vendors when producing hardware and software modules as shown in Table 2 [8].

Table 2-DOWNTIME OF AVAILABILITY

Availability	9s	Downtime
90%	One	36.5 days/year
99%	Two	3.65 days per year
99.9%	Three	8.76 hours/ year
99.99%	Four	52 minutes/ year
99.999%	Five	5 minutes/ year
99.9999%	Six	31 seconds/ year

Serviceability is a broad definition describing how easily the system is serviced or repaired. For example, a system with modular, hot- swappable components would have a good level of serviceability. Technology may limit unscheduled downtime by having systems with redundant dynamic reconfiguration, constant system monitoring and resource management. [5]

Mean Time between Failures (MTBF) is the average (expected) time between two successive failures of a component. It is a basic measure of a system's reliability and availability and is usually represented as units of hours. - Mean Time to Repair (MTTR) (or Recover) is the average (expected) time taken to repair a failed module. This time includes the time it takes to detect the defect, the time it takes to bring a repair man onsite, and the time it takes to physically repair the failed module. Just like MTBF, MTTR is usually stated in units of hours. The following equations illustrate the relationship of MTBF and MTTR with reliability and availability [4] and [5].

Reliability = $e^{-\text{Time}/\text{MTBF}}$

Availability = MTBF / (MTBF + MTTR)

The following conclusions can be reached based on these formulas [4]:

*The higher the MTBF value is, the higher the reliability and availability of the system.

*MTTR affects availability. This means if it takes a long time to recover a system from a failure, the system is going to have a low availability.

*High availability can be achieved if MTBF is very large compared to MTTR.

Sandia Laboratories has suggested new Energy-Reliability (EneRel) metrics that are noted as reliability-aware by the use of a subscript r, for example ErD and ErD2 At a very high level, EneRel can be thought of as: [9]

$$E_{neRel} = E_{rt} + (E_{fail\ recov} * (p(fail) + p(fail_{add\ rt})) \quad (1)$$

The Efail for an energy saving technique under test differs from the Efail of the baseline implementation since the baseline technique's p(failadd rt) is zero. Figures 1a, 1b and 1c show the impact that failures can have on energy consumption given different failure rates and increases in runtime for evenly distributed failures over the runtime. For large node/socket counts the amount of energy that can be lost due to failures in the increased runtime period is non-negligible as HPC component counts keep rising and will continue to rise for Exascale computing and beyond.

Reliability must improve along with rising component counts to make future systems viable. Despite improvements in component reliability, reductions in whole system reliability are still likely to occur. The forecasts in Figure 1 a,b,c show that reliability mechanisms will no longer play a minor role in energy consumption. EneRel proves useful for illustrating that sometimes spending slightly more energy to finish a job faster, thereby reducing the probability that a failure occurs during the job's runtime, may in some cases actually result in lower energy consumption. The exploration of the effect of reliability on energy consumption is made all the more important by the fact that reliability can be reduced when using energy saving techniques due to thermal cycling [10] and lowered operating voltages result in an increase in soft faults [11].

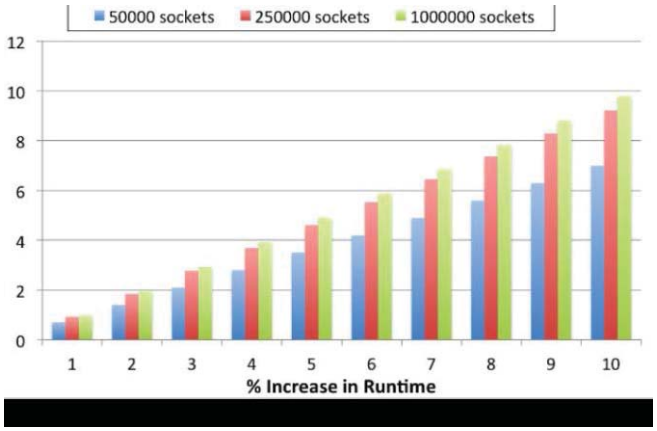


Figure 1 (a) Energy overhead due to reliability for varying percentage increases in runtime for 3 year, MTBF

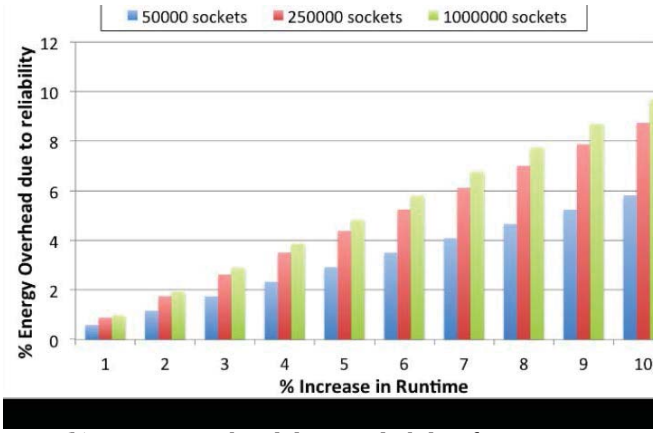


Figure 1 (b) Energy overhead due to reliability for varying percentage increases in runtime for 5 year, MTBF

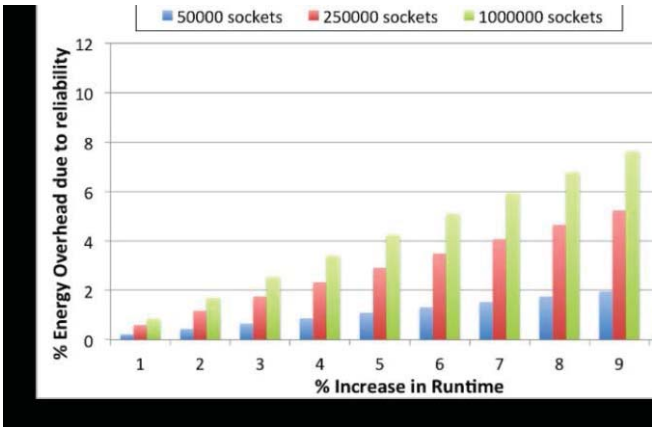


Figure 1 c) Energy overhead due to reliability for varying percentage increases in runtime for 25 year, MTBF

III. The US Army Engineer Research & Development Center (ERDC) Information Technology Laboratory (ITL) facility

The US Army Engineer Research & Development Center (ERDC) Information Technology Laboratory (ITL) facility has 10,000 square feet of air conditioned space with four feet deep false flooring for utilities. The computer systems are Cray Sonexion (12 processors) and experimental LEO. There is a section of UNCLASSIFED servers/processors and a much larger set of CLASSIFIED servers/processors. There is a section of Silicon Graphics Inc. (SGI) computers with speeds of 1.2 or 4.6 Penta Flops. One petaflops is equal to 1,000 teraflops, or 1,000,000,000,000,000 FLOPS.
Computer systems cost $25M and are operated for 4 years before replacement. ERDC HPCC has HPC computers that are ranked 16th fastest in the world. User computers have access to 12 Penta-Bytes of memory. HPCC's do not charge for use of the computer 1.9 Billion core hours per year, but applications must be for US government or non-profit (University) entities. Online access is available at www.uit.hpc.mil Backup generators supply 2.4 Mega Watts of power and there are extensive cooling systems to support operations. Emerson power systems and Data Direct 3.5" disk drives were observed. C ray uses a proprietary interconnect circuitry but Silicon Graphics Inc. (SGI) uses open architecture. Both Cray and SGI compute module have large copper cooling heat transfer plates. Cooling water flow rates provided in channels to maintain the required exchange rate and return the heated water to cooling towers before the return cycle to the HPC system.

IV. Performance Degration

The typical reliability curve is shown below in Figure 2. The initial failure region depends upon the screening and quality of the components and workmanship. The initial failures may be high due to latent defects that were introduced during the manufacturing or assembly process and not caught during the testing phase. The center useful life region is where random, contact hazard occurs. The end

of life region is where mechanical devices such as air cooling fans fail and cause over heating of the electronic and electro migration of device junctions which degrade performance or result in catastrophic failures.

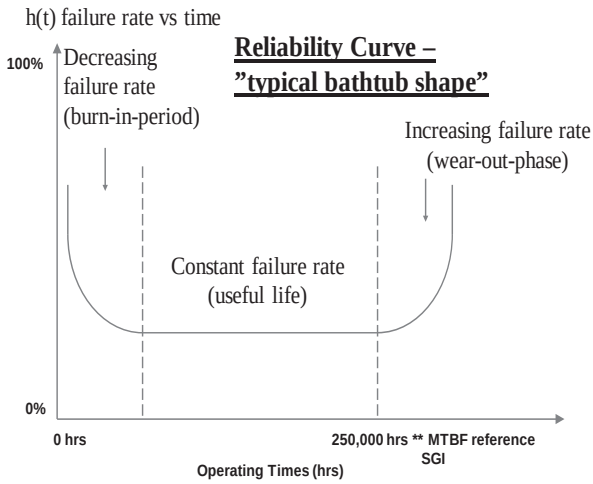


Figure 2 – Failure rate as a function of time at a single environmental condition (temperature, humidity, salt atmosphere, etc.) curve [1]

V. Thermal Control and Heat Dissipation is Critical for Microprocesor Performance and Lifetime

One dimensional conduction requires that the junction and case nodes be isothermal, however, neither the die nor the lid are isothermal; therefore, this thermal model requires corrections for heat spreading from the chip to the heat sink. Theta jc is the temperature rise from the device junction to the case.

$$\theta_{JC} = (T_j - T_c) / \text{Power} \quad (2)$$

Intel ® Xeon ® microprocessor have a specified Theta jc of 0.3 ° C per Watt for maximum applied power of 90 Watts. Rearranging the equation and substituting the specification values results in Tj = Tc + 27 ° C; therefore, the junction temperature is a direct function of the cooling water temperature with these results predicted: Tj = 62 ° for 35 °C Tc, Tj = 57 ° for 30 °C Tc, and Tj = 52 ° for 25 °C Tc, neglecting losses across the Thermal Interface material (TIM).

A simplified one dimensional heat flow model was proposed by Bar Cohen as shown in Figure 5 [6] where Theta junction-to-case jc, Theta case-to-sink cs and Theta sink-to-ambient sa or to a reservoir when liquid cooling is used.

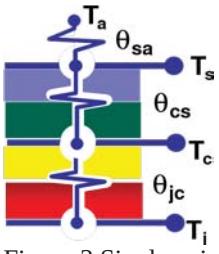


Figure 3 Single axis heat flow model [6]

Figure 4 shows the actual heat flow from the jucntion to case then lateral through the case lid and spreading into the external heat sink for dissipation through the combination of heat conduction into the cooling liquid or convection to the surrounding air flow created by cooling fans. [7]

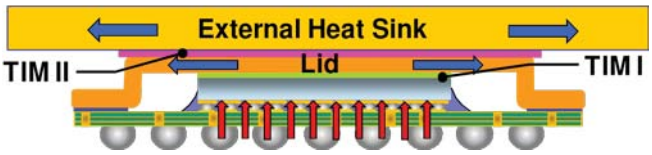


Figure 4 heat flow from flip chip ball grid array microprocessor to the heat sink

It is clear that control of the microprocesor junction temperature is critical to performance and minimizing degradation due to device operation and elevated temperatures near the maximum limits.

VI. Technological and Procedural Methods to Extend Supercomptuer Useful Life

The microprocessor is the heart of the high performance computing system that generates thermal energy during operation. There would be performance degradation unless adequate heat transfer cooling is provided which is normally a case temperature of 0° to 70°C. Thermal case temperature regulation is provided by thermal conduction through the heat sink and transfer to external cooling water or other exchangers to an external support facility. Cooling air is provided by keeping the room temperature in the 65°-70°F range and providing fan air flow rates to change the typical 72 cubic foot equipment cabinet air out every 5 minutes or 14.4 CFM flow rate.

A standard model used by the industry is the Sum-of-failure-rates (SOFR) model, which makes two assumptions to address this problem: (1) the processor is a series failure system – in other words, the first instance of any structure failing due to any failure mechanism causes the entire processor to fail; and (2) each individual failure mechanism has a constant failure rate (equivalently, every failure mechanism has an exponential lifetime distribution). Microprocessor failures are accelerated by temperature, duty cycles and electro and stress migration effects resulting from the AL and Cu metal interconnections and junctions.

The FIT value targeted by reliability qualification, FIT_{target}, is a standard. Currently, processors are expected to have an MTTF of around 30 years – this implies FIT_{target} is around 4000 [12].

From MIL-STD-217Fn2, paragraph 5.1 the failure rate λ_p for microprocessors is estimated by

$\lambda_p = (C_1 \pi_T + C_2 \pi_E) \pi_Q \pi_L$ Failures/ 10^6 Hours per million hours of operation.

The most critical parameter is the π_T the effects of elevated temperature operation. From the MIL-STD-217Fn2, Section 5.2 and 5.8 tables for microprocessor CMOS technology shows values of π_T at 90 °C = 1.1 and π_T at 75 °C 0.71 which will directly result in an approximately +25% increase in MTTF.

The lower temperature operation may be achieved by implementing these changes in the High Performance Computing cooling systems

- Install temperature monitoring and coolant flow rate sensors at the input of rack assembly
- Redesign the parallel and series routing of coolant to achieve a maximum exchanger temperature at the microprocessor heat sink of 70 °C
- Monitor and control processor work flow to avoid peak duty cycles without providing additional cooling.

Performance modeling and testing has shown that Intel® microprocessors (at the very least Sandy Bridge, Ivy Bridge, and Haswell) can run at their maximum Turbo Boost frequency all the way up to 100 °C. While there may be a tiny performance difference between a microprocessors running at 30 °C and one running at 95 °C, our testing has found that the difference is miniscule. In fact, even after running benchmarks dozens of times the difference is so small that it is essentially nonexistent.

Puget Systems recently performed tests to measure the effects on operating frequency when a microprocessor core temperature approaches 100 °C. The Intel Core i7 4790 microprocessor was cooled it with a Gelid Silent Spirit Rev. 2 cooler that was connected to a manual PWM fan speed controller. By running Linpack (a benchmark widely used in the scientific community) and slowly dialing the fan speed down in careful increments, to allow the microprocessor to overheat by incremental amounts. At each cooling increment a log of the Linpack benchmark results were monitored as well as using CoreTemp to record the microprocessor core temperature and frequency, as shown in Figure 5 [1]. Since the Intel microprocessor thermal limit is 100 °C, the amount of overheating occurs when the core temperature was running at > 99 °C.

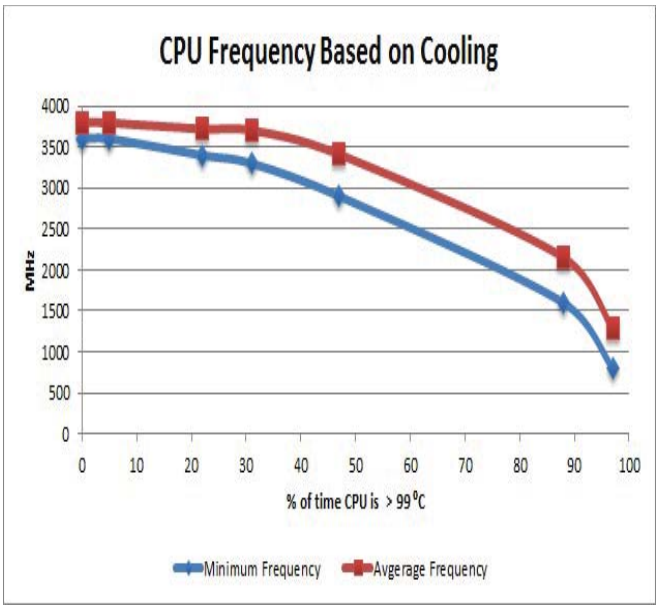


Figure 5 – Intel ® Intel Core i7 4790 cooling effects on microprocessor frequency [1]

The testing showed that while the minimum core load frequency started to drop as soon as the core hit 100 °C, the average microprocessors frequency didn't drop by more than 0.1GHz until the core was overheating more than 30% of the time. In fact, Intel microprocessors are surprisingly good at being able to handle large thermal changes with only a small reduction in the average frequency. However, for high performance computing centers, the frequency changes and subsequent performance degradation could extend run times significantly and also reduce the component lifetimes similar to degradation results from accelerated life testing at elevated temperatures that shortens the useful lifetimes.

Intel ® server-board-s2600jf with Xenon ® processors, are used in high performance computing applications and as shown in Figure 6, may have two or more microprocessors with heat sinks to dissipate the thermal energy. The Mean Time Between Failures (MTBF) is 25,000 hours, 100, 000 hours or 250,000 hours depending upon the operating environment and physical configuration.

Figure 6 – Intel ® server-board-s2600jf with Xenon ® processors that are used in High Performance Compute Modules

VII. CONCLUSIONS AND FUTURE PLANS

Reliability, Availability, Serviceability (RAS) are key factors to consider in improving the performance and extending the lifetime of high performance computer center systems. The Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR) are critical to system availability. Modular hot swappable compute modules, cooling fans and even redundant systems may be required to meet program operating objectives.

Dynamic system monitoring similar to an electrical power generation facility may be required to detect and correct support equipment out-of-specification operation to minimize performance degradation of the computing system.

Further work with the temperature control facility and the compute module supplier RAS may be needed to measure and characterize the relationship of performance degradation as a function of cooling systems under several operating loads.

Extension of the computer system lifetime will be achieved through the implementation of built-in-test-equipment (BITE) monitoring that will provide the required feedback for optimum environmental controls.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. Department of Defense High Performance Computing Modernization Program under the U.S. Army Engineer Research and Development Center (ERDC) contract number W912HZ-15-2-0001 entitled: "*Strategic Cyber Science, Warfare, Security, Application Development and High Performance Computing Research and Development,*" and the research project is entitled: "*Investigating High Performance Heterogeneous Computing with Advanced Architectures,*" and in part by Army Research Office HBCU/MSI contract number W911NF-13-1-0133 entitled: "*Exploring High Performance Heterogeneous Computing via Hardware/Software Co-Design.*"

REFERENCES

[1] Impact of Temperature on Intel CPU Performance– Puget Systems https://www.pugetsystems.com/labs/articles/Impact-of-Temperature-on-Intel-CPU-Performance-606/. Oct 14,2014

[2] Anas Alfarra, Jamory Hawkins, Gerald Morris, and Khalid H. Abed, "Mapping the Conjugate Gradient algorithm onto High Performance Heterogeneous Computers," The 2015 International Conference on Embedded Systems and Applications (ESA'15), Las Vegas, Nevada, July 27-30, 2015. pp 43-46.

[3] Reliability Analysis Center, Reliability Toolkit: Commercial Practices Editions, Rome Laboratory.

[4] Torell, W. and V. Avelar, Mean time between failure: Explanation and standards. White Paper, 2004. 78.

[5] Calculating Total System Availabiltiy Hoda Rohani, Azad Kamali Roosta Information Services Organization KLM-Air France Amsterdam.

[6] Bar Cohen A., R. Eliasi and T. Elperin, "θjc characterization of chip packages – Justifications, limitations and future", presented at the IEEE 5th annual Semi-Therm Conference, 1989.

[7] Jesse E. Galloway, Siddharth Bhopte, Cameron Nelson Amkor Technology 1900 S. Price Rd. Chandler, Arizona 85286 www,amkor.com. Characterizing Junction-To-Case Thermal Resistance And Its Impact On End-Use Applications.

[8] Weygant, P.S., Clusters for High Availability: A Primer of HP Grant, ryan .Solutions. 2001: Prentice Hall Professional.

[9] Ryan E. Grant, Stephen L. Oliver, James H. Laros III and Ron Brightwell, Sandia National Laboratories, "Metrics for Evaluation Engergy Saving Techniquest for Resilent HPC Systems, 2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops.

[10] J. S. S. T. Association, "Failure mechanisms and models for semiconductor devices," Tech. Rep., 2006.

[11] V. Chandra and R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS," in Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08. IEEE International Symposium on, 2008, pp. 114–122.

[12] Jayanth Srinivasan, Sarita V .Adve, Pradip Bose, JudeA.Rivers "The Case for Lifetime Reliability-Aware Microprocessors" , The 31st International Symposium on Computer Architecture (ISCA-04), June 2004.

A Software-Defined Network Configuration Providing Differentiated QoS to an eHealth Environment

Marcus Assuiti[1], Felipe Volpato[1], Madalena Pereira da Silva[2], Mario Antônio Ribeiro Dantas[1]
[1]Department of Informatics and Statistics and [2]Department of Engineering and Knowledge Management
Federal University of Santa Catarina (UFSC), Florianópolis, Santa Catarina, Brazil
assuiti@gmail.com, fvolpato@gmail.com, madalena.silva@posgrad.ufsc.br, mario.dantas@ufsc.br

Abstract - *Sepsis is an inflammatory reaction that starts with an infectious condition whose mortality rate is bigger than that of breast cancer, prostate cancer and HIV together. Early diagnosis and agility in the implementation of a protocol is crucial to ensure the effectiveness of the treatment. This paper presents an approach to Quality of Service (QoS) management aimed at providing services aware of the urgency in the sepsis diagnosis. The model proposes a non-concurrent environment for services delivery where the sepsis application will guarantee QoS using Software Defined Network (SDN) resources. The implementation of the proposed model shows to be feasible and functional and will avoid concurrent data in the delivering of services with the sepsis application.*

Keywords: Software Defined Network; QoS; Sepsis

1 Introduction

Medical Informatics is a multidisciplinary field where information technology is applied to health care. It encompasses efficiency, quality, data analysis, security, better governance, news resources, clinical pathway, and cost reduction. This technology has become indispensable due to various factors for instance high volume of information, secrecy, need for information availability, complex diagnosis and treatment, knowledge management, disclosure of relevant information, among others.

According to study [1], sepsis, severe sepsis and septic shock have mortality rates of 33.9%, 46.9% and 52.2%, respectively. There is evidence that the clinical skills to diagnose Systemic Inflammatory Response Syndrome (SIRS), infection and septic shock are satisfactory, but those to diagnose sepsis and severe sepsis are still unsatisfactory.

Some facts can be considered about sepsis:

- One person dies of sepsis every four seconds [3].
- More than six million babies and children die of sepsis every year [3].
- If treated within the first four hours the chance of survival is 50%, after 12 hours the chance drops to 15% [4].
- Treatment of sepsis occupies 25% of the ICU beds[3].
- Treatment for non-survivor patients is more expensive than for survivor patients [3].

Sepsis clinical pathway is a priority and it is urgent within the hospital routine. Therefore it requires SDN resources to ensure QoS for the sepsis application. In other words, SDN resources will provide QoS to the application that will control the sepsis protocol, ensuring the necessary bandwidth and background traffic control, thus facilitating rapid and effective implementation of the sepsis protocol.

2 Characteristics

Sepsis is a body response that is triggered due to an infection and can cause tissue damage, organ failure, and death. It is crucial to diagnose sepsis in the early stages for a prompt treatment because it evolves fast and should be faced as an emergency. Sepsis is also known as a septic or generalized infection syndrome. As a topic of investigation, it is important to first define some terms related to sepsis

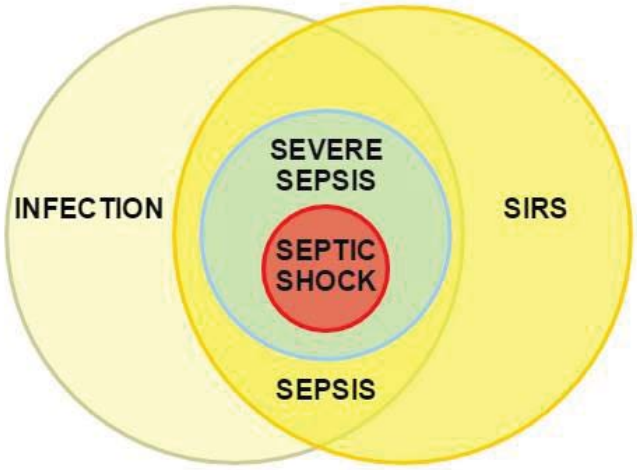


Figure 1 - Sepsis taxonomy

Figure 1 show terms discussed and defined in [6] and how they relate in a Venn diagram:

- Infection occurs due to the permanence of a living micro-organism that is able to multiply and cause pathological changes.
- SIRS is a systemic response of the body to an infectious or noninfectious insult such as trauma, pancreatitis, burns, with at least two of the characteristics below:

1) Fever, body temperature > 38 Celsius or hypothermia, body temperature < 36 Celsius;
2) Tachycardia - heart rate>90 bpm;
3) Tachypnea - respiratory rate > 20 breaths per minute or PaCO2 < 32 mmHg;
4) Leukocytosis or leukopenia - leukocytes > 12, 000 cells / mm3 or < 4, 000 cells / mm3, or presence of > 10% of young forms (rods)

• Sepsis is defined by SIRS criteria and a suspected or confirmed infection.
• Severe sepsis presents organ dysfunction and tissue hypoperfusion identified by lactic acidosis, oliguria or altered level of consciousness, or hypotension with systolic less than 90 mmHg. This is considered the most common cause of death in noncoronary critical care units (CCU).
• Septic shock is an acute collapse, and hypotension does not respond to the administration of intravenous fluids. Hypotension is defined as systolic blood pressure < 90 mmHg reduction of > 40 mmHg from baseline, and mean arterial pressure < 60 mmHg despite adequate fluid resuscitation, requiring vasopressors, in the absence of other causes of hypotension.

2.1 Sepsis complexity and urgency

Ensuring a clinical pathway for the sepsis problem is a tough challenge. According to the conclusion in [1], the diagnosis of sepsis is satisfactory in case of infection, SIRS and septic shock, but it is still insufficient in case of sepsis and severe sepsis. Furthermore, intensive care doctors should have a better performance in this regard. Study [7] categorizes some factors that influence clinical decision in three kinds:

• Personal: motivation; bodily fatigue; financial situation; stress.
• Patient: poor instructions; psychic diseases; lack of clarity; newly born; unreliable information.
• Environment: infrastructure and multidisciplinary team; costs.

The roadmap implementation suggested by [3] demonstrates that the sepsis protocol involves a large part of an institution. Some steps include for example a desired interval of 30 minutes. Treatment urgency is evidenced in Figure 2 showing the inversely proportional relationship between survival rate and early antibiotic action.

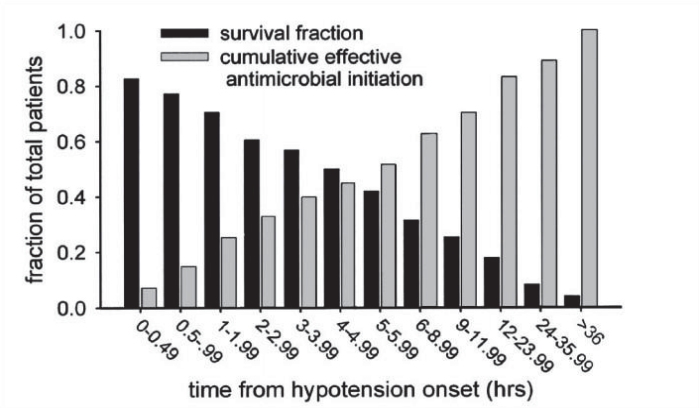


Figure 2 - Sepsis urgency[8]

2.2 Sepsis bundles

This subsection will discuss treatment protocols after sepsis suspicion. Sepsis bundles were developed based on [10] and [12].

There are two severe sepsis bundles:

a) *Sepsis resuscitation bundle*

This bundle is defined by [11] as the combination of evidence-based objectives that must be completed within six hours for patients with severe sepsis, septic shock or lactate > 4*mmol/L*.

- Measure serum lactate. Thirty minutes for completion of this step;
- Obtain blood cultures prior to antibiotic administration;
- Broad-spectrum antibiotic within three hours of ED admission and within one hour of non-ED admission. Thirty minutes for completion of this step.
- Treat hypotension and elevated lactate with fluids;
- Administer vasopressors for hypotension not responding to initial fluid resuscitation to maintain mean arterial pressure (MAP) > 65 mmHg;
- Achieve a central venous pressure of > 8 mmHg;
- Achieve a central venous oxygen saturation > 70% or mixed venous oxygen saturation (SvO2)> 65%.

b) *Sepsis management bundle*

It consists of evidence-based goals for patients with severe sepsis and septic shock. It must be completed within twenty four hours.

- Administer low-dose steroids for septic shock in accordance with a standardized ICU policy;
- Administer recombinant human activated protein C in accordance with a standardized ICU policy;
- Maintain glucose control lower than limit of normal, but < 180mg/dL(10mmol/L)
- Maintain a median inspiratory plateau pressure for mechanically ventilated patients.

3 Related Work

A literature review on QoS and SDN involving sepsis showed no occurrence of proposals going further than the one in our study. Table 1 displays a review with the type of application and a brief description of the proposal in each paper.

Paper	Application	Description
[2]	eHealth. Video streaming and gaming services	Different QoS use approach for QoE modeling and architecture in order to adapt network resources using Software-Defined Networking (SDN), centered in the user experience.
[14]	Several. Network management	QoS policy management framework that monitors the network and automatically applies pre-determined policies using SDN.
[17]	Several. Network management	System that supports high-level configuration using SDN in order to manage QoS in home networks.
[15]	Video streaming	SDN controller that provides dynamic QoS routing for multimedia applications.
[16]	Several	OpenFlow 1.0 extension that improves QoS by employing Linux multiple packet schedulers.
[18]	Several. Network Management	QoS approach for an open-source SDN controller focused on DiffServ and Traffic Shaping.
[20]	eHealth. Video streaming	QoS applied for video stream and clinical data.

4 The Proposed Model

This section presents the proposed environment and proposed tests and the means used to ensure QoS in the case of sepsis, considering the infrastructure and network traffic and providing means to facilitate the hospital administration in terms of good practices.

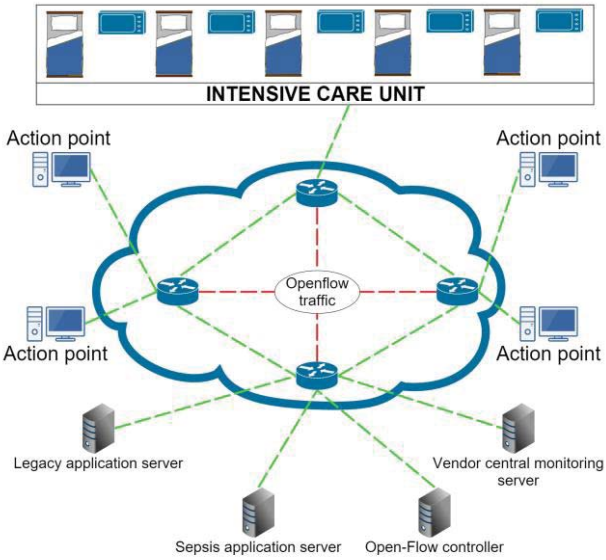


Figure 3 - Model Proposal

In a quick overview, Figure 3 shows a hospital environment with an **intensive care unit (ICU)** monitored with bedside monitors that communicate with the **vendor central monitoring server**. The central integrates the interface with the **legacy application server,** which sends data related to the sepsis protocol to the **sepsis application server**.

Possessing the relevant information obtained from the electronic medical record (EMR) through the sepsis application support can help in the diagnosis of sepsis. Then, the protocol starts to alert the sectors where implementation tasks are needed, i.e., the **action points**.

We intend to provide best QoS as regards the network behavior. Our goal is to ensure clinical pathway using SDN in the sepsis protocol, as suggested by institutions involved in campaigns against sepsis.

4.1 Component integration model

An idealized environment consists of an integration of three systems: the supplier of bedside monitors; the legacy application of the institution; the sepsis application. Figure 4 illustrates this integration.

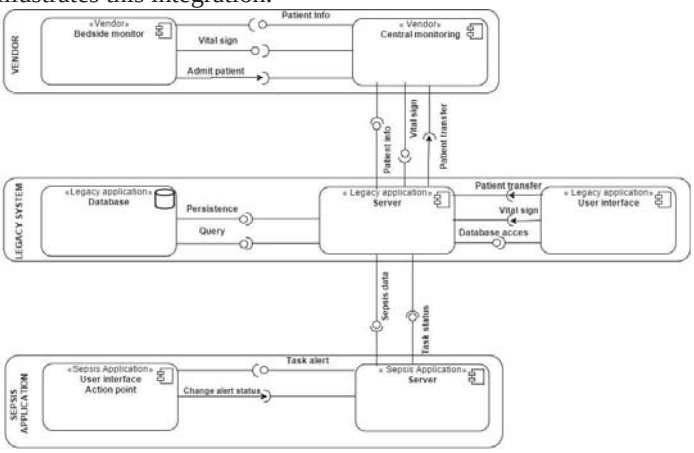


Figure 4 - Component integration model

The sepsis application will work in this integration, thereby causing little impact to the existing infrastructure of the institution. The main idea is to use existing tools and information to provide a clinical decision support system with an integrated EMR for diagnosis, flow control and alerts related to sepsis, and with the urgency required by the sepsis protocol.

4.2 QoS with the legacy system

Keeping a medical record of the patient's health is a legal and organizational need, as can be seen in [9]. The large amount of information and the hospital routine require very consistent medical systems, so the solution presented here is based on the legacy system and mainly in the EMR.

Our experiment will formulate an external application to the legacy application and will make the resource available with a conventional integration. In other words, the legacy

application will receive information related to sepsis which will be passed on to the sepsis application server. However, this information will be initially used only by the sepsis application in the action points notifying activities with high priority to the sectors where they are needed.

QoS will be applied in two ways, as defined below:

a) Input flow: the input flow refers to relevant data about sepsis registered in the EMR and sent to the sepsis application server, especially in ICU environments since they are monitored sectors with high incidence of sepsis. For instance, the national report [3] pointed out a number of 81.6 cases of sepsis recorded in ICUs.

As it is a critical environment, the patient's vital signs are monitored by bedside monitors that can be from different suppliers and in general communicate with the vendor central monitoring station which, in turn, forwards the information to the legacy system.

The QoS between bedside monitors and the central monitoring station will be performed by resource reservation for communication in this local network. This is simple among bedside monitors because they are hardware for the sole purpose of communicating with the central monitoring station.

b) Output flow: the output flow begins when the sepsis protocol starts, then tasks are sent to the sectors where some action is needed. This is the biggest challenge of this proposal because at this point we need to differentiate the regular traffic from the traffic of sepsis bundles in the network as an integration of imaging examinations, antivirus, the legacy system, emails and any other processes of the institution. Messages sent to the action points will not have their QoS guaranteed through resource reservation, but through queue management. This will be done through a specific port for all network traffic related to the sepsis application and the flow controller will prioritize all the traffic going on through this port. It is crucial that no other application will use this port defined in the network.

5 Case Study

Case study [13] describes JVF's case – a nineteen-year old pregnant woman admitted to a hospital in the municipality of Limeira, state of São Paulo, Brazil. Her case quickly evolved as follows:

- 06/08/2011, 11:31 am: Was admitted to the emergency department and later was found to have pyelonephritis;
- 06/10/2011, 06:35 am: After worsening, was transferred to a semi-intensive care unit where was diagnosed with sepsis of urinary origin;
- 06/11/2011: Was requested remain in the ICU due to new worsening;
- 06/20/2011: After appropriate treatment, was discharged from the ICU due to improvement;
- 06/24/2011: Was discharged from hospital and referred to the high-risk pregnancy service of the municipality
- 10/21/2011: Underwent a cesarean section giving birth to newborn with 2510g;
- 10/24/2011: The newborn was discharged from hospital.

The following aspects of the abovementioned study can be discussed:

- The study was conducted at ISCML, a teaching hospital that provides tertiary care in the municipality of Limeira.
- The institution has medical intensivists working 24 hours a day;
- The institution suggests that the Surviving Sepsis Campaign protocol can be one of the strategies to achieve the United Nations' goal which is to reduce maternal mortality by 75%; whereas serious infections are one of the three leading causes of death in the pregnancy-puerperal cycle.
- It discusses the difficulty of the gynecologists team in diagnosing sepsis, and points out that this is a common disease in any specialty. It highlights the need to follow the sepsis protocol and extend it to all areas.
- The discussion finally reports that even though the pregnant woman and the baby survived, this case is classified by the World Health Organization (WHO) as a near miss. This term refers to a stage that precedes death, which was interrupted by timely and adequate management or luck.

Questions

- The diagnosis of sepsis was given in the ICU almost two days after admission and then the correct antibiotics were administered. Could not it have started before, giving the patient greater survival rate and less cost?
- Is it reasonable to expect, as pointed out in this case study and confirmed in [1], that intensive care physicians are more qualified to diagnose sepsis?
- In the diagnosis of sepsis, can it be expected that all sepsis triggers are properly analyzed based on calculations with standard deviation and the time recommended in the guidelines for sepsis bundles?
- Considering the low popularity of sepsis and the difficulty in diagnosing it, as pointed out in [1], is it reasonable to expect that the institution should have a professional able to make a correct diagnosis? Can we expect that a professional be available? How about young professionals?

6 Experimental Results

Firstly, this section provides information about the environment used in the experiment. Subsequently, we describe the tests performed. Finally, we present the results obtained.

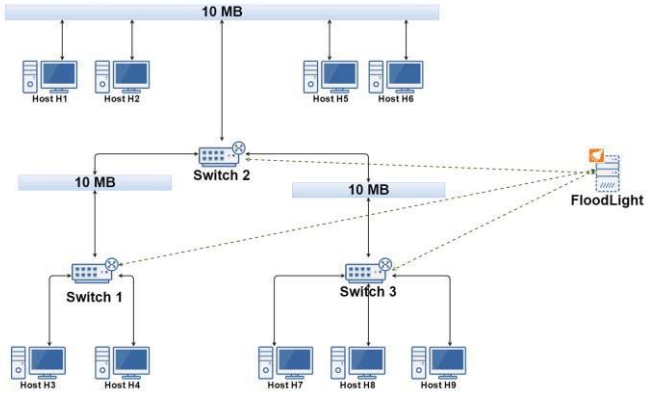


Figure 5 - Network Topology

Figure 5 displays the network topology on a virtual machine with the Mininet[22] emulator, the Floodlight[21] controller and the Open vSwitch software[24].

Mininet is a complete network emulator platform that allows the creation of multiple nodes (hosts/switches/controllers). Besides, it is useful for experimenting different topologies and is widely adopted in SDN research. Floodlight is an open-source SDN controller project written in Java and has a large community and good documentation. Open vSwitch is an open-source software switch project that is compatible with protocols such as Open-Flow and QoS queues configuration. All the switches in our experiment are Open vSwitch implementations.

Moreover, the Floodlight controller, nine hosts (h1 to h9) and three switches (s1 to s3) were created for the tests.

A. Test 1 – Bandwidth between hosts

For the first experiment, the following six connections (C) were established:

- C1: between h1 and h9 on port 9000, lasting 30 seconds;
- C2: between h2 and h9 on port 9000, lasting 30 seconds;
- C3: between h3 and h8 on port 80, lasting 50 seconds;
- C4: between h5 and h8 on port 80, lasting 50 seconds;
- C5: between h4 and h7 on port 9001, lasting 30 seconds;
- C6: between h6 and h7 on port 9001, lasting 30 seconds.

For all the connections we used the Iperf [23] software.

Connections 1 and 2 simulated traffic between the bedside monitors and the central monitoring station. Connections 3 and 4 simulated background traffic, for instance: HTTP web surfing, file download, video streaming, and even imaging exams. Finally, connections 5 and 6 simulated the sepsis application data.

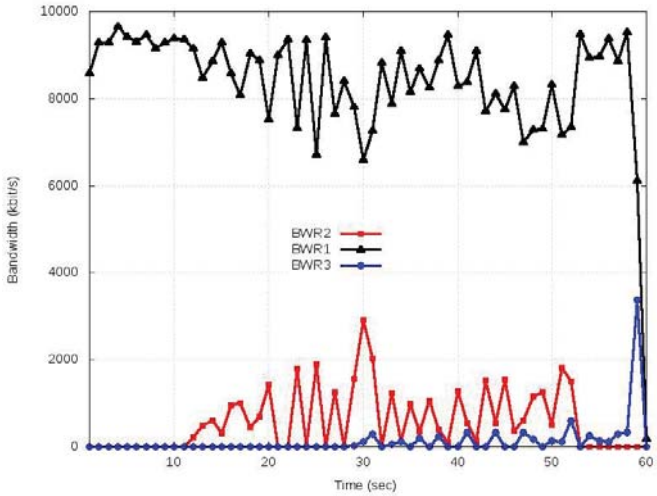


Figure 6 - Results of measurements without QoS mechanism

The test in Figure 6 began with connections 3 and 4 performing background traffic simulation without speed restrictions. This is represented in the graph by the bandwidth received in h8 (BWR1). Ten seconds later, connections 1 and 2 started to simulate the application traffic (ICU legacy server) on port 9000 (BWR2). At 20 seconds from the start of the test, communication began on the AP-sepsis application server on port 9001 (BWR3). For this test, there was no QoS setup among the nodes, only the bandwidth set at 10mb among all links.

Figure 6 shows that the background traffic ends up consuming virtually all the available bandwidth across the links and impairing the applications connections among the bedside monitors, the central monitoring station, and the sepsis application. In this scenario, there was no bandwidth guarantees among the hosts.

In order to ensure bandwidth for our applications (Figure 7), we defined queues in the Floodlight Controller as follows:

- Queue 1 with maximum rate of 8 megabits;
- Queue 2 with minimum rate of 4 megabits and maximum rate of 4 megabits;
- Queue 3 with minimum rate of 2 megabits and up to 2 megabits.

After setting up the queues, we have the following configuration:

- Traffic between h1 and h9 - queue 3.
- Traffic between h2 and h9 - queue 3.
- Traffic between h5 and h8 - queue 1.
- Traffic between h3 and h8 - queue 1.
- Traffic between h4 and h7 - queue 2.

- Traffic between h6 and h7 - queue 2.

By comparing the graphs in Figure 6 and Figure 7, it can be seen that, from the moment they came into being, an SDN configuration ensuring QoS for the applications using specific ports delivered much more consistent data in line with the urgency required by the application.

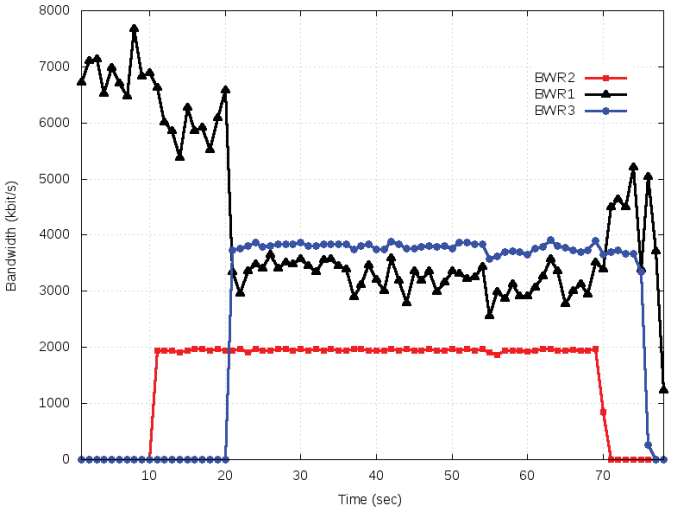


Figure 7 - Results of measurements with QoS mechanism

As described in section two, the sepsis protocol requires urgent implementation of its stages, where the hospital management should take steps with estimated time of 30 minutes. Thus QoS provided by SDN can help meeting the protocol's urgency requirements.

TABLE
FILE TRANSFER TIME (IN SECONDS)

File size(MB)	FT1	FT1 with QoS	FT2	FT2 with QoS
4	60.41	17.03	89.26	8.79
8	72.66	34.03	57.70	17.43
1	113.99	74.46	87.58	34.52
3	168.11	135.16	181.49	70.41

B. Test 2 – File sent through hosts

For the second test (Figure 8, Table I), we made file transfers with different file sizes between two pair of hosts: h1-h9 on port 9000 (FT1), simulating traffic between the bedside monitors and the central monitoring station, and h4-h7 on port 9001 (FT2), simulating the sepsis application data. We kept connections C3 and C4 from the first test for background traffic purposes.

We used the same QoS parameters from the first test, so FT1 traffic used queue 3 with minimum and maximum rate of 2 megabits. The traffic for the FT2 configuration used queue 2 with minimum and maximum rate of 4 megabits.

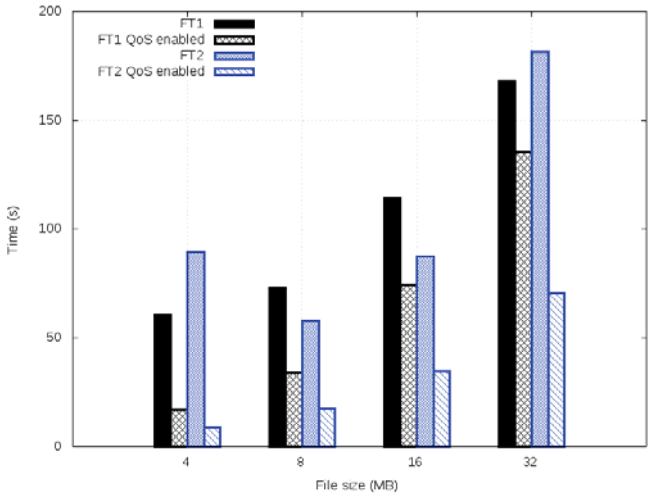


Figure 8 - File transfer test

The results in Figure 8 showed that the implementation of QoS can significantly reduce the time to send data. In this study, in particular, this proved to be essential in order to maintain the applications' requirements. We can also conclude that the greater the minimum bandwidth rate, the greater the difference between using or not QoS parameters.

C. Test 3 – Latency

This test took exactly the same parameters from the first test. Our goal was to check the latency between connections C1, C2, C5 and C6. Figure 9 shows that when there is no QoS guarantee in the SDN, latency tends to have very high values according to the network competition. In turn, with the QoS mechanisms, latency always tends to range between 0 and 1.

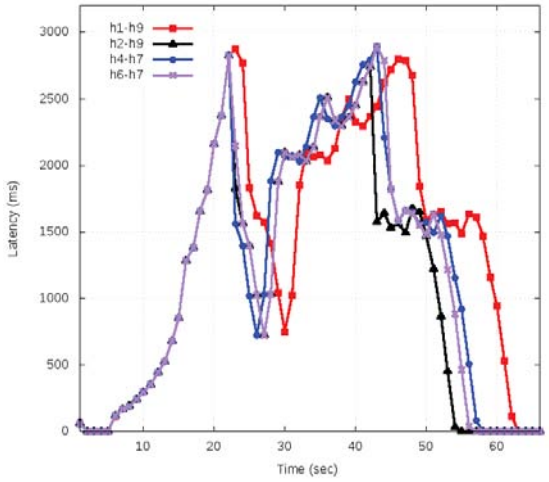


Figure 9 - Host latency

7 Final Remarks And Future Work

This paper proposed the use of Software-Defined Networking technology to meet the urgency and ensure the quality of service required for a critical application, maintaining an entire topology and architecture with low

impact to the environment. In a hospital setting, it provides service guarantees for the application that implements the sepsis protocol.

The test showed to be promising as regards the use of QoS through queues with the sepsis application. Other concurrent hosts on the network without QoS contributed to communication deterioration. The test also demonstrates that it is possible and relatively simple to apply SDN features in an application so as to guarantee its required QoS, as the application has an exclusive port in the network where it is applied.

As future work, we intend to analyze and build a clinical decision support system with capabilities of sepsis diagnosis, alert and management. Initially, we intend to deploy the sepsis protocol proposed by [3] and then enable the institution to customize the flow in relation to the sepsis triggers. We also intend to study a way to use the context of sepsis variables to ensure QoS, as in [19]. For example, body temperature between 36 and 38 degrees Celsius is not interesting for the sepsis protocol. It is however a big challenge because some sepsis triggers are calculated according to standard deviations of vital signs. Besides, we intend to consider a way to differentiate packets that could be sent to an action point located in a monitored sector. In this sense, as soon as the sepsis protocol is started, that information should be given a higher priority for patients who are being monitored and may not develop sepsis.

Once applied in a real environment, it would be very interesting to develop management indicators, providing the institution with a vision of a sort of Business Intelligence (BI), thus allowing better management of the sepsis steps. It would also be possible to display the time that each flow task is taking to run as well as the data during the progress and outcome of the sepsis case.

In the previous sections, it is clear that the problem of sepsis is critical because it is a major cause of death in ICUs and because the medical staff is not able to diagnose it in a timely manner for treatment. It is urgent, as can be seen in Figure 4, and an analysis of the sepsis packages suggested by [3] for an early diagnosis is important for an effective treatment. It is also complex, as it is a current problem that involves the entire medical staff and, according to The Latin American Sepsis Institute (ILAS), a medical diagnosis in early stages guarantees a more efficient treatment.

8 Acknowledgments

We would like to thank Philips Health Care for the support in this work.

9 References

[1] Assuno Murillo, et al. *Survey on physicians knowledge of sepsis: Do they recognize it promptly?* Journal of critical care 25.4 (2010): 545-552.

[2] da Silva Madalena P., et al. *A Managing QoE Approach for Provisioning User Experience Aware Services Using SDN* Proceedings of the 11th ACM Symposium on QoS and Security for Wireless and Mobile Networks. ACM, 2015.

[3] ILAS, Latin American Institute, in *http:\\ilas.org.br, feb-2016.*

[4] WORLDSEPSEDAY, World Sepse day, *in http:\\world-sepsis-day.org*

[5] Shorr Andrew F. et al. *Economic implications of an evidence-based sepsis protocol: Can we improve outcomes and lower costs?* Critical care medicine 35.5 (2007): 1257-1262.

[6] Levy Mitchell M., et al. *2001 sccm/esicm/accp/ats/sis international sepsis definitions conference.* Intensive care medicine 29.4 (2003): 530-538.

[7] Guilherme Almeida Rosa da Silva. *O processo de tomada de deciso na prtica clnica: a medicina como estado da arte* Rev Bras Clin Med. São Paulo 11.1 (2013): 75-9.

[8] Kumar Anand et al. *Duration of hypotension before initiation of effective antimicrobial therapy is the critical determinant of survival in human septic shock.* Critical care medicine 34.6 (2006): 1589-1596.

[9] Federal Council of Medicine (Brazil). *Sets medical records and mandates the creation of the Medical Records Review Committee in health institutions* Official Journal the Federative Republic of Brazil resolution CFM 1.638/2002(2002): 184-185.

[10] Rhodes Andrew et al. *The Surviving Sepsis Campaign bundles and outcome: results from the International Multicentre Prevalence Study on Sepsis. The IMPreSS study.* Intensive care medicine 41.9 (2015): 1620-1628.

[11] Khan P Divatia JV. *Severe sepsis bundles. Indian Journal of Critical Care Medicine* Peer-reviewed, Official Publication of Indian Society of Critical Care Medicine. 2010;14(1):8-13.

[12] Dellinger, R. Phillip, et al. *Surviving Sepsis Campaign guidelines for management of severe sepsis and septic shock.* Intensive care medicine 30.4 (2004): 536-555.

[13] Fatima Aparecida Henrique Lotufo. *Sepse Grave de Origem Urinria na Gestao* (2012).

[14] Bari, M. F., Chowdhury, S. R., Ahmed, R., and Boutaba, R. (2013). Policycop: an autonomic QoS policy enforcement framework for software defined networks. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE.

[15] Egilmez, H. E., Dane, S. T., Bagci, K. T., and Tekalp, A. M. (2012). Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–8. IEEE.

[16] Ishimori, A., Farias, F., Cerqueira, E., and Abelém, A. (2013). Control of multiple packet schedulers for improving QoS on openflow/sdn networking. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 81–86. IEEE.

[17] Seddiki, M. S., Shahbaz, M., Donovan, S., Grover, S., Park, M., Feamster, N., and Song, Y.-Q. (2014). Flowqos: Qos for the rest of us. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 207–208, New York, NY, USA. ACM.

[18] Wallner, R. and Cannistra, R. (2013). An sdn approach: quality of service using big switchs floodlight open-source controller. *Proceedings of the Asia-Pacific Advanced Network*, 35:14–19.

[19] Cabral Nazario, Debora, et al. (2014) An Approach to Evaluating Quality of Context Parameters in an Ambient Assisted Living Environment. *Computer-Based Medical Systems (CBMS), IEEE 27th International Symposium on. IEEE.*

[20] Ongaro, Francesco. Enhancing Quality of Service in Software-Defined Networks. *Diss. alma Master Studiorum-University of Bologna,* 2014.

[21]Floodlight-QoS-Beta, *Floodlight-QoS-Beta*, in http:\\floodlight.atlassian.net\floodlight.atlassian.net\wiki\display\floodlightcontroller\How+to+implement+

[22] Mininet, *Mininet*, in http:\\mininet.org

[23] Iperf, *iperf*, in http:\\dast.nlanr.net\Projects\Iperf\

[24] Open vSwitch, *openvswitch*, in http:\\http://openvswitch.org/\

Information System for Smart Grid: Systemic and UML combined approach

Ali SNOUSSI [1], **Samir BEN AHMED** [2]
[1]LR LISI, INSAT, University of 7th November at Carthage, Tunis
E-mail: ali.snoussi.smartgrids@gmail.com
[2]LR LISI, INSAT, University of 7th November at Carthage, Tunis
E-mail: samir.benahmed@fst.rnu.tn

Abstract - *The effective management of large and complex systems depends on the presence of all relevant information describing system within the reach of the actors and decision makers. To this extent, we propose in this paper a combined methodology for information system design based on a participatory approach that brings together several parts to ensure better result juxtaposed with an object-oriented process based on Unified Modeling Language diagrams to design the database of the information system following a spiral development cycle. This methodology is implemented through the systemic method: PPPO, the Simplified Unified Process and the spiral development cycle. It allows us to model the smart grid and develop an information system that describes all information necessary for its good management.*

Keywords: Smart Grid, Systemic methods, Information System, Participatory Planning of Project by Objectives, Simplified Unified Process.

1.INTRODUCTION

The use of information systems in the company has become a daily reality. Therefore, the management and design of information systems represent today a major problem of organizations. To meet this need, several studies and attempts are developed to facilitate and even automate the information system development process.

This paper describes a systemic methodology combined with an object-oriented modeling and design tool for analysis, modeling and planning projects and systems and designing their information systems. This methodology is based on systemic approach and follows a participatory method juxtaposed with an object-oriented development process. The system analysis and modeling are made by the Participatory Planning of Project by Objectives method (PPPO) and the Simplified Unified Process (SUP).

This paper is divided into three parts. First, we start with the presentation of the methodologies and tools used for analysis and modeling the smart grid and the development of its information system. Second, we present the output (result) validating the proposed methodology for the design of smart grid information system through the implementation of the PPPO method that aims to capture the problems and objectives of the system and deduct the appropriate strategies, the adoption of the spiral cycle, and the simplified unified process as a development and process cycle respectively for the design of the database. Finally, we discuss some points about the choice of these methodologies, the work done and the future one.

2.METHODOLOGIES & TOOLS

Based on the key characteristics of the smart grid namely its complexity, extended and heterogeneity [6], we chose to analyze the system and develop its information system a systemic approach for the first step (modeling of the problem). For the second one, we opted to use a development process based on UML diagrams through an iterative cycle.

2.1. Systemic approach (Methodology)

The world phenomenon known as "systemic movement" and the large number of ideas and practices that can be classified under "systemic approach" are keys to the implementation of a solid methodology that considers the following systemic aspects [7]:

- The consideration of "systems" as abstract objects for building a better understanding and a shared vision of a complex system.
- The identification of the essential dimensions and high priority issues.
- The predominant importance given to the study of relationships between elements of the system and the system with its environment.

- The interest in collective and participatory reflection and group work.
- The need for autonomy and integration into decision making.
- The possibility to use images and graphics in the development of specific actions and the information system.

The systemic approach in the analysis and design of information systems allows also being conscious of the future world in which the treaty system should be located. It allows thus to spend a major weakness of the organizations that is the resistance to changes in their environments therefore secure their sustainable development [8].

2.2. Systemic used method

We used to analyze the control problem of smart grid and develop a coherent and faithful to reality information system the method: Participatory Planning of Project by Objectives (PPPO).

PPPO's goal is to collect the knowledge and expertise of experts and stakeholders in order to reach a broad consensus and a collective solution to the questions related to the Smart Grid [3]. It serves to mobilize the efforts and know-how of different stakeholders around a structured and systematic approach by linking them in workshops and structured meeting of "brainstorming" to generate the maximum of ideas that will later be the subject of analysis, classification and modeling [3], [8].

PPPO has the capacity during its exercise to motivate stakeholders who may have different repositories to be active, influential and participate in the analysis and design process to be part in decision-making process and have a strategic reflection on future actions [8]. The method tries to find a compromise land between the different actors through an iterative negotiation process for the exchange of views and even debates around some critical situations [8].

It is widely flexibilized by its instigator (German Cooperation Agency "GTZ") to avoid its mechanistic, rigid and falsely participative use. So, it has become just a general framework counselor rather than a series of tools and prescribed mandatory steps. This flexibility puts the actors and their perceptions of the problems, needs, interventions, etc. at the center of a non-deterministic planning process, an iterative negotiation and not final process [8].

PPPO is based on a very linear approach to reality (cause produces effect). It starts with a negative reading (reverse) of this reality, i.e. a succession of problems [8].

Like many modeling tools, PPPO provides possible simplifications [8]. In short, we limit ourselves at this stage to some problems and causal links, and we choose some prospects to prove our solution.

The method is based on five methodological steps; identification of stakeholders, developing a problems tree, developing an objectives tree, defining strategies and developing the planning matrix (logical framework). This approach is retroactive as a spiral process, i.e., each step used for supplying the previous steps (addition of new actors, problems, goals, etc.).

2.2.1. Stakeholders Identification

The first step is to identify the individuals, groups and institutions that are relevant to the system. Determine their interests and views on the issues. The relations and the positioning of these players are also subjected to analysis in order to properly master the debates between them and understand the "contradiction" of their viewpoints sometimes. For example debates between economists on the one hand and naturalists and environmentalists on the other hand [8].

Stakeholders meet in workshops for a common reflection and shared discussion of different points of view.

2.2.2. Development of problems tree

The first phase is to identify problems in bulk (brainstorming). This collective exercise is done using small cards by annotating one problem per card. Thereafter, prioritize and organize the cards problems according to causality relations (cause gives effect) to build a problems tree [3], [8].

The tree can have one or more roots (one / many major problems). Each node of the tree represents a problem cause while his descendants (branches) are the negative consequences.

2.2.3. Development of objectives tree

The issues identified in the previous phase are translated into objectives insofar the resolution of these issues results the desired future state. i.e., negative states are translated into positive desirable states. An objectives tree is automatically built by translating causality links into end-means links. Only solutions of possible problems are retained [3], [8].

2.2.4. Definition of strategies

The objectives tree helps bring up several possible alternatives and strategies that can help in solving the posed problems and issues. One or more potential alternatives are selected to form adequate strategies and policies according to well-defined and carefully selected criteria defined by domain experts such as skills, priorities, available resources (human, financial, etc.), economic and political climate, etc.

2.2.5. Development of the planning matrix (logical framework)

Once the objectives and strategies are well defined, indicators and tools to measure, verify and evaluate the result of the planned activities will have to be specified. These activities are determined according to the strategies defined to achieve the desired objectives.

All these information are transposed and summarized in a matrix that attempts to answer the following questions [3], [8]:

- The why of the project?
- What are the results (objectives)?
- What are the external factors of importance to project success (hypothesis / risks)?
- How to evaluate the success / failure of the project (objectively verifiable indicators)?
- Where to find the data needed to evaluate the project?
- How much the project will cost (means)?

2.3. Development Process

To design the solution of the Smart Grid control problem and the development of its SI, we have chosen as a development process a reduced and simplified derivative of the unified process.

A unified process is a software development process built on UML. It is iterative and incremental, focusing on architecture, driven by the use cases and piloted by risk. The management of such a process is organized in four phases presented as follows: pre-study (inception), elaboration, construction and transition. Its development activities are defined by six fundamental disciplines that describe business modeling, requirements capture, analysis and design, implementation, testing and finally deployment [30].

The adopted simplified unified process comprises the following nine steps [30]:

2.3.1. Development of use cases

This step is very important, in fact, these diagrams explain clearly the expected application services and the needs it must satisfy.

2.3.2. Development of System Sequence Diagram

This diagram shows the interactions between actors and the system.

2.3.3. Implementation of Human Machine Interface (HMI) models

The models give the possibility to work on the design appearance before delivery of the final version of the software and discuss his critics from the start of work.

2.3.4. Development of the domain model

This is a first class diagram to identify the entities and their attributes and associations.

2.3.5. Development of participating class diagram

It is a diagram that includes the three types of classes; dialogs, controls and entities, it is based on the domain model.

2.3.6. Development of the navigation diagram

Two diagrams can fulfill this task, the activity diagram and the statechart. They complement the sequence diagram.

2.3.7. Development of intercation diagram

It is a detailed diagram of sequences, it reflects the communication between the different implemented objects.

2.3.8. Development of the design class diagram

It is obtained by the enrichment of participating class diagram by adding functions from detailed sequence diagrams.

2.3.9. Implementation

This step consists in generating the database (classes). It ends by filling the code completely, test and validate it.

2.4. Development Cycle

In order to increase productivity and to estimate development time, it is imperative advised to follow a development cycle. It turns out that several approaches and methodologies exist and can meet this need.

After a comparative study of these methodologies and cycles, we felt that the spiral cycle is the most suitable for our case.

It is a software design procedure that want more pragmatic than traditional methods. This model enables highly responsive to its requests. It aims to real satisfy the needs. The spiral model is distinguished compared with conventional methods by its principles that present its strengths [1]:

- Giving more importance to individualities and foster communication than following fixed processes to ensure communication and serious discussion between the project participants.
- Perform iterative tests to ensure having fully tested software at the end.
- Involve all participants in development in order to avoid possible misunderstandings.
- Instead of following an exact plan, develop and evolve the project while responding to any changes identified during work.

This software development cycle model includes the different steps of cycle V. By implementing successive versions, the cycle starts by offering a product more complete and robust. However, it puts more emphasis on risk management in the cycle V. There are four phases in the course of the spiral cycle [1], [31]:

i. Determining objectives, alternatives and constraints.
ii. Analysis of risks, evaluation of alternatives.
iii. Development and verification of the solution.
iv. Review of results and verification of the next cycle.

3. CASE STUDY OF A SMART GRID (RESULT)

3.1 Application of PPPO

The PPPO method is used in order to model the problem of management and control of the Smart Grid to identify the needs of the system that are used later in the design of the solution. In short, these needs present the starting point for use cases diagrams.

The result of the application of this method is presented in the following in this section.

3.1.1 Identification of Stakeholders

The Smart Grid is a complex system [5], [6] that affects many parties. To ensure the identification of all individuals and groups interested in energy system and that can contribute to the completion and success of the Smart Grid project, we chose to identify stakeholders layer by layer of the Smart Grid Architecture Model "SGAM" (Component Layer, Communication Layer, Information Layer, Function Layer, Business Layer) [9].

Some stakeholders are included in several layers, namely the Manager of the grid which is involved in the physical chain (production, transmission and distribution of electricity) and in the logic chain and grid control.

3.1.2 Problems tree

After brainstorming workshops, and the classification of identified Smart grid problems, a main problem is unveiled: the implementation and control of smart grid based on the existing aging infrastructure. We partitioned the problems that arise from this main problem into four groups (Management problems, Technical problems, economic problems and Environmental problems).

3.1.2.1 Management problems

The first part concerns the real-time management of energy production, transportation from fields, turbines and power plants to customers through a transport and distribution network. The power management also extends to cover energy consumption. Indeed, one of the benefits of smart grids is the inclusion of the user in the management process.

Logical management issues are classified as a sub-part dealing with data flow management, the taking of adequate and timely decisions and the management of resources (human, financial, etc.).

3.1.2.2 Technical problems

As the first part, the second one is divided into two categories, physical and other logical problems. The first correspond to the power system namely issues related to the equipment and installations especially cohabitation between new technologies and equipments and existing infrastructure.

One of the most relevant issues in this part is the distributed generation through the introduction of renewable and intermittent new energy resources [10].

The explosion of the demand and the new arrival (electric vehicle (EV)) [11], [12], the automatic incident detection and resolution of faults [13], [14], [15] are also predominant issues in the implementation of Smart Grid.

The second category in this part concerns the logical problems: security of the energy system against cyber-attacks and confidential data hacking threats of customer and all parties included in the energy chain [16].

Among posed technical problems, we identify the issues of standardization and legislation whose role is so important that they must be answered before any efforts in the implementation of Smart Grid [17], [18], [19], [20].

Besides the energy network traffic, a data traffic system is essential for rapid collection of information and reliable communication between consumers and regional centers of conduct and parties concerned by this information [19], [21], [22], [23].

Technical problems also include backup and archiving problems of electric grid characteristic values and history of the entire energy system for a possible electric network analysis.

3.1.2.3 Economic problems

The economic problems are linked mainly to the financing and investment, the stress of the energy market, the speculation problem and economic balance income / expenses.

The dynamic billing [18], [24] and prices of energy use affect the economic side especially with the increase of distributed generation and consumer participation in energy production (e.g. providing excess energy produced locally by its solar panel).

The loss of jobs presents also a very critical social and economic problem. In fact, the use of new intelligent control and management technologies and the automation of many parts of the energy system may eliminate a lot of jobs [25].

3.1.2.4 Environmental problems

The obvious first problem is related to the pollution issues (emission of greenhouse gases, waste of nuclear activities, etc.). The second problem is the resources depletion mainly those based on fuels due to its overexploitation. The third one, which is very serious and dangerous, is the health problem and environment damage from radiofrequency (RF) and electromagnetic frequency (EMF) radiation from new information and communications technologies (ICT) and Smart Meter installation [25].

3.1.3 Objectives tree

The objectives present the resolution of the previously mentioned problems and the response to the relevant questions. In short, the same structure characterizes the two trees (problems tree and objectives tree).

3.1.3.1 Management objectives

The Smart grid allows to maintain balance between supply and demand by developing an effective real-time work plan, optimizing decisions in terms of time and efficiency and using intelligent management of stocks of raw materials (fuels, nuclear resources, etc.) to satisfy customers and supply new needs (electric vehicles) and reduce energy consumption especially during peak hours and avoid falls of the electricity grid. This will help to reduce damage and bad impacts to incidents and organize the relationships between parts of the smart grid.

3.1.3.2 Technical objectives

In the technical level, the smart grid aims to upgrade existing infrastructure by adding new technologies, increase the estimation and analysis capacities of climate changes and their impact on the energy system balance. It improves techniques and energy storage means and secures the communication network. It is necessary also to ensure the interoperability of different parts of smart grid by standards and different parties by adequate laws and appropriate legislations.

3.1.3.3 Economic objectives

Economic objectives affect both clients whatever their kind (industrial, commercial, individual) by rationalizing consumption and consequently billing and Increase local energy production and society as a whole (the state) by reducing energy costs and make economic gains, founding a robust economy and limit the loss of jobs.

3.1.3.4 Environmental objectives

The environmental objectives take into consideration the conservation of energy resources against overexploitation and the reduction of pollution and health problems.

3.1.4 Definition of strategies

The search for solutions to achieve the desired objectives permits to classify them into groups. Each group allows the appearance of many alternatives to satisfy those goals. These alternatives form the Energy Policy and the strategies leading to the success of the smart grid.

The strategies adopted are:

- Diversify and distribute energy resources and improve the use of renewable resources to reduce dependence on fuel-based resources that have unstable prices [10].
- Decentralizing the management and control [24].
- Inform the consumer of his consumption and introduce him in the control process.
- Securing the exchange and sharing of data between control centers [16].
- Facilitate investment in the energy market with adequate decisions and legislations.
- Evolving the energy storage technologies.
- Adjust the production program in real time according to supply and demand.

3.1.5 Developpement of logical framework

In this step, we limited ourselves to the definition of control units and their activities which allow us to simulate the Smart Grid and implement our solution in the rest of the work. Four units are defined (inspired from [17], [2], [28]) to manage the Smart Grid in order to rationalize consumption and local energy production. In short, the Smart Grid is divided into three compartments based on the seven domains of National Institute of Standards and Technology (NIST) model. The first compartment is the Production (Generation), the second encompasses the energy delivery grid and the parts affecting the system such as markets managers, operations and service providers. Third, the consumption compartment. Each compartment is managed by a type of logic units. The fourth one is responsible for the management of information flows circulating in the system.

This step has enabled us to develop a set of specifications containing functional and technical requirements of the solution. The functional requirements are used to set the activities and uses of each identified unit (agent).

3.2 Application of simplified unified process

The simplified unified process starts up with the needs identified by the application of PPPO.

The second modeling phase of the Smart Grid enables the design of an IT solution to the problems mentioned above and the development of a coherent IS for the Smart Grid.

As mentioned, the proposed solution is based on four types of units (agents):

- Consumer Agent: responsible for the management of energy consumption (consumption compartment).
- Producer Agent: responsible for the management of energy production (production compartment).
- Control Agent: manages the entire energy system including the management and coordination of other agents (control compartment).
- Database Agent: responsible for data management and present the access point to the database for other agents.

Each agent is considered as an autonomous, independent and application that communicates with others parts.

The output of the simplified unified process is presented as a design class diagram enabling the development of the source code of the solution. The IS presents the direct projection of this class diagram.

The diagrams elaborated specify the characteristics (attributes) of each agent, its tasks (methods), interactions (associations) with its environment and with its neighbors (agents).

4. DISCUSSION

4.1 Choice of systemic methodology

Several techniques are developed for the analysis and modeling of IS, most of which are part of two major classes "Cartesian methods" and "systemic methods" [4].

We chose a systemic class method given to the nature of Smart Grid system. Indeed, the electrical system is among the widest and complex human systems [5], [6], [24]. It encompasses several subsystems (power subsystem, management subsystem, and human subsystem [24]) and brings together many and various stakeholders. Such a system requires great coordination among its heterogeneous parts for its better management. That is why we felt that the Cartesian methods are unable to meet our needs for analysis and modeling of the Smart Grid as asserted C. FLOYD "Cartesian methods are applicable to medium-sized systems with little human-machine interaction and when the features of the system are relatively clear in advance" [4].

The table TABLE I shows a comparison between the two classes of methods:

TABLE I. COMPARISON OF GENERATION AND CASES OF APPLICATION BETWEEN SYSTEMIC AND CARTESIAN METHODS

Systemic methods	Cartesian methods
2nd generation [4]	1st generation (appeared in the 60s) [4]
• large and complex systems [27] • systems with heterogeneous parts [27]	• small and medium systems [4] • existing systems [4] • known system functions [4]

We excluded the analytical approach also because it aims to reduce the system to its smallest elements [29] in contrast to the systemic approach that addresses the system in the whole of its components and its internal interactions [26]. The analytical approach applies the rule "divide and rule" so that each element of the system becomes an independent problem, smaller and easier to solve. Thereafter, evaluate all solutions and use compensatory algorithms to mutually compensate the resulting values of different dimensions. Therefore, the use of analytical methods entails many, lengthy and intensive calculations [29].

For these reasons, the analytical approach is reserved for easy problems and its use as a problem solving process does not always result solutions when the system studied is complex [29].

The TABLE II illustrates an opposition of Joel de Rosnay of the two approaches (systemic and analytical) [29]:

TABLE II. COMPARISON BETWEEN SYSTEMIC AND ANALYTICAL METHODS

Systemic approach	Analytical approach
Connects : focuses on the interactions between system elements	Islands : focuses on the elements
considers the effects of interactions	considers the nature of interactions
is based on the global perception	is based on the precision of details
insufficiently rigorous models as a basis of knowledge, but, usable in the making decision and action	precise and detailed models, but difficult to use in action
effective approach when interactions are important and nonlinear	effective approach when interactions are low and linear

Systemic methods have some gaps too. For example, the method adopted (PPPO), although it is used by many funders, and that it benefit of a systemic and participatory approach, it admits shortcomings. One of the risks of the method lies in the disproportionate importance that can be given to planning to the detriment of reflection and constant questioning of the changing context. It can become demotivating and disempowering [8]. But PPPO allows benefiting from other tools and juxtaposing other techniques to overcome its limitations. In fact, it represents just a counselor [8].

4.2 Utility of PPPO

The SUP starts from needs and based on use cases. In our case, we do not have tender specifications that contain these requirements and functional and technical needs and we are invited to identify them. Hence the need for a tool that precedes the SUP to make the problem analysis.

We believe that the systemic approach is the most appropriate class as explained previously.

We used the systemic approach PPPO given the clarity of its principle and the simplicity of its application.

4.3 Implementation of PPPO

4.3.1 Meeting stakeholders and brainstorming workshops organization

A Smart Grid is a wide and complex system [5], [6] that affects several stakeholders from different fields. Thus, the meeting of all these parties is a non-obvious mission. Indeed, it requires the availability of these parties and their agreement to share their knowledge and expertise. This phase also requires significant budgetary and human resources not all currently available. However, we have exploited existing ones and we have benefited from the meeting and workshops organized to analyze Smart Grid and design a minimal IS that can be enriched afterwards and assist in decision making.

4.3.2 Logical framework development

This step requires even more coordination between participants in the project. Indeed, the establishment of a complete logical framework and plan for the implementation of a national Smart Grid requires more financial and human resources. That's why we are limited in this step to the definition of solution actor's and the definition of their tasks to identify the use cases of the proposed solution.

4.4 Choice of Unified Process Simplified

PPPO is insufficient to realize the SI in a support usable for decision making. So it is necessary to complete it with another tool to design and implement the IS.

We chose the SUP because it is an object-oriented method that represents the market trend toward. It plays a dual role. It models the IS as a set of diagrams finalized by a design class diagram to realize the IS and develop a database in the first hand. On the other hand, it allows the passage to the development of decision-making system in the future based on the same elaborated design.

The SUP benefits from the power and completeness of Unified Process (UP) and speed of the extreme programming, of course while reducing the cumbersome of standard UP [30].

4.5 Choice of the spiral cycle

We opted to use the spiral model because it is the most suitable for new applications, and it allows benefiting from several advantages of which we mention [1]:

- Allows catching up if we miss one or more use cases.
- Practical and safe solution validation.
- Validation as early as possible.
- Upward design.
- Identify possible changes early.
- Make variants iterations (sprints) to give time to code. Each sprint has a specific goal or "backlog".

5. CONCLUSION

This paper presents a methodology for the study and modeling of complex systems to develop an information system. The systemic method PPPO and the simplified unified process based on UML are used to analyze the smart grid and develop its information system in a

participatory process that brings together all stakeholders in order to identify problems and challenges of implementation of the Smart Grid, classify and structure them in the form of a tree that will be converted into an objectives tree that reflects the strategies and the planning of Smart grid implementation activities. It is proved that these steps are very important and sensitive. They greatly affect the following steps. Indeed, the design of solution of all these issues in the form of UML diagrams developed as part of the unified process and refined in each iteration (sprint) following a spiral development cycle are deducted from this analysis and modeling performed.

The proposed approach enabled to design a minimal IS that can be enriched permanently and will serve the decision system.

In the future, it is important to treat the hosting and access to data to complete a logical framework.

6.REFERENCES

[1] J. J. NGANG BILOUNGA. Méthode de Conception des Systèmes d'Information.[Online]. Available : « https://www.google.tn/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiw-ajv8JDLAhVHaRQKHXYhC7IQFggZMAA&url=http%3A%2F%2Fwww.foad-mooc.auf.org%2FIMG%2Fpdf%2Fcours_mcsi.pdf&usg=AFQjCNGurzasr7tSEzqEplipRdFroAc6CA&bvm=bv.115277099,d.bGQ » (Accessed 2015-02-19).

[2] M. Pipattanasomporn, H. Feroze, and S. Rahman. Multi-Agent Systems in a Distributed Smart Grid: Design and Implementation. In Power Systems Conference and Exposition (PSCE). Seattle. IEEE 2009. pp. 1-8.

[3] M.N. LAKHOUA Analysis and Modelling of Industrials Systems in order to develop an Information System. In Research Challenges in Information Science. 2009. Fez. IEEE Third International Conference on pp. 403-408.

[4] C. ROLLAND, A. FLORY. Nouvelles perspectives des systèmes d'information. In congrès 90 de l'Association informatique des organisations et systèmes d'information et de décision. 1990, Paris, Editiond Eyrolles. P 3-40.

[5] G. GUERARD, S. BEN AMOR, A. BUI. Approche système complexe pour la modélisation des Smart Grids.

[6] C. PETERMANN, S. BEN AMOR, A. BUI et al.Optimisation de Smart Grid: d'un modèle intégratif vers une simulation multi-agents autonomique. In Modélisation Agents pour les Systèmes Complexes. 2013. Lille.

[7] E. COUDERT, J.P. GIRAUD, J. MONTGOLFIER (eds.). Guide d'utilisation de 'Imagine' - Analyse Systémique et Prospective de Durabilité. Blue Plan n°3. Mars 2006. ISBN: 2-912081 - 17-3.

[8] C. ACHEROY, H. HADJAJ-CASTRO. Méthode de planification par objectif (PIPO, PPO, PPPO, ZOPP). Creative Commons belgique Attribution. Octobre 2006.

[9] CEN-CENELEC-ETSI Smart Grid Coordination Group. Smart Grid Reference Architecture. November 2012.

[10] C. BELET CESSAC. Analyse du cadre réglementaire de l'accès au réseau des producteurs d'électricité à partir d'énergies renouvelables en Tunisie - Etude de préfaisabilité sur les axes de développement. June 2014. Tunis.

[11] Y. K. PENYA, J. C. NIEVES, A. ESPINOZA et al. Distributed Semantic Architecture for Smart Grids. In Energies 2012. pp 4824-4843.

[12] J. MIRANDA, J.BORGES, M. J. G. C. MENDES et al. Development of a multi-agents management system for an intelligent charging network of electric vehicles. 18th IFAC World Congress. August 28 - September 2, 2011. Milano.

[13] S. BOU GHOSN, P. RANGANATHAN, S. SALEM et al. Agent-orineted Designs for a Self Healing Smart Grid. In Smart Grid Communications (SmartGridComm). 2010. First IEEE International Conference on pp. 461-466.

[14] S. MOHAN, K. BHALERAO, S. A. KHAPARDE. A Review of Self healing applications in Smart Grids. In Fifth International Conference on Power and Energy Systems. October, 2013. Kathmandu, Nepal.

[15] D. SUTANTO, D. YE, M. ZHANG. Design of an Intelligent Self-Healing Smart Grid using a Hybrid Multi-Agent Framework. In JOURNAL OF ELECTRONIC SCIENCE AND TECHNOLOGY, VOL. 9, NO. 1, MARCH 2011. pp 17-22.

[16] D. VON OHEIMB. IT Security architecture approaches for Smart Metering and Smart Grid. Siemens Corporate Technology, Munich, Germany.

[17] National Institute of Standards and Technology (NIST). NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 3.0. February 2014.

[18] J. DEDRICK, Y. ZHENG. Information Systems and Smart Grid: New Directions for the IS Community. In iConference. 2013. Fort Worth, TX, USA.

[19] V. C. GUNGOR, D. SAHIN, T. KOCAK et al. Smart Grid Technologies: Communication Technologies and Standards. IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 7, NO. 4, NOVEMBER 2011.

[20] V. Vyatkin, G. ZHABELOVA, N. HIGGINS et al. Standards-enabled Smart Grid for the Future EnergyWeb. In Innovative Smart Grid Technologies (ISGT). 2010. Gaithersburg. pp. 1-9.

[21] B. BENAOUDA. La Communication Sans Fil dans un Réseau Electrique Intelligent (Smart Grid) - Méthodologie de Développement. Informatique. Université du Québec à Montréal. 2013.

[22] V.K. SOOD, D. FISCHER, J.M. EKLUND et al. Developing a Communication Infrastructure for the Smart Grid. In Electrical Power & Energy Conference (EPEC). Montreal. 2009. pp. 1-7.

[23] W. WANG, Y. XU, M. KHANNA. A survey on the cmmunication architectures in smart grid. In Computer Networks 55. 2011. pp 3604-3629.

[24] S. CHEBBI. Production - Transport et Distribution d'Energie - Notions de base sur les réseaux électriques. Université vertuelle de Tunis.

[25] N. BEETY. Analysis: Smart Meter and Smart Grid Problems. Second Edition. December 2012.

[26] G. MINATI. Introduction à la systémique.

[27] Définition systémique et approche systémique | L'approche systémique. [Online]. Available : « http://www.approche-systemique.com/definition-systemique/ ». (Accessed 2014-10-20).

[28] Saifur Rahman, Manisa Pipattanasomporn, and Yonael Teklu. Intelligent Distributed Autonomous Power Systems (IDAPS). In Power Engineering Society General Meeting. Tampa. IEEE 2007. pp. 1-8.

[29] E. GAULIN. Approche Analytique et Approche Systémique | L'approche systémique. [Online]. Available : « http://www.approche-systemique.com/approche-systemique/approche-a... ». (Accessed 2014-10-20).

[30] UML 2. [Online]. Available : « http://laurent-audibert.developpez.com/Cours-UML/?page=mise-en-oeuvre-uml ». (Accessed 2015-12-17).

[31] Modèle en spirale. [Online]. Available : « https://fr.wikipedia.org/wiki/Modèle_en_spirale ». (Accessed 2015-12-17)

A JSON-Based Markup Language for Deploying Virtual Clusters via Docker

Scott Morton, Salvador Barbosa, Ralph Butler and Chrisila Pettey
Department of Computer Science, Box 48
Middle Tennessee State University
Murfreesboro, Tennessee, USA

Abstract - *Our team develops large cluster-based projects (often using MPI). At times it is necessary to simulate a cluster and perhaps even two or more networked clusters. This occurs particularly in a testing mode where one or more of the clusters may be involved in production work. It is possible to run multi-rank MPI jobs on a laptop or desktop, but it is more problematic to simulate a cluster or a multi-cluster environment. The technique presented in this paper uses Docker containers, a JSON system configuration description, and Python scripts to quickly and easily build and run user defined networked systems on any of the three major host operating systems - Linux, Mac OS X, and Microsoft Windows.*

Keywords: Virtual Clusters, Linux Containers, Docker, JSON, Python

1 Introduction

While configuring a cluster, or a network, requires an understanding of networking (e.g. IP addresses) and operating systems concepts (e.g. host-based authentication), most users of high performance clusters are not the system administrators and never have to concern themselves with the potentially tricky issues involved in configuring them. However, at times, it would be convenient to simply use a virtual cluster that is a replica of the actual hardware. Such situations might include, the system not being available due to scheduling, or the need to run a test of a simple change to a piece of software, or the system is inaccessible due to being in a remote location. Whatever the situation, we need to run some experiments, and we cannot run them on the actual hardware. In this instance, it would be nice to have a virtual replica of the system on our laptop. However, configuring a virtual system on our laptop suddenly puts us in the position of being the system administrator.

In order to alleviate the hurdles faced in such a situation, we proposed a markup language for describing virtual clusters in 2005 [1]. That system based on XML, QEMU, VDE, TUN/TAP, and Python was called VCML, was fairly simple to use, and was almost indispensable in debugging software running on various, sometimes confusing, hardware configurations. However, over time, technologies changed. Some software is no longer supported, while newer, more powerful software becomes available. With the popularity of JSON [5], and the advent of Linux Containers [6] and Docker [3], we have developed a new virtual system configuration software system, VCML2. This paper describes VCML2 in section 2 and illustrates its use with some example clusters in section 3.

2 Virtual Cluster Markup Language

To understand the proposed system for describing and deploying virtual clusters, it is necessary to be somewhat familiar with the underlying software that is combined to create VCML2. This underlying software, including Linux Containers, Docker, and JSON will be discussed in section 2.1. Then section 2.2 will present the basics of VCML2

2.1 Support Software

The project presented in this paper is about building virtual clusters on a host computer running any of the three most common operating systems: Linux, Mac OS X, and Microsoft Windows. The prior project, VCML, used virtual machine software, but this project, VCML2 uses container software. When using container software, it is important to understand two notions: image and container. An image is a passive or static entity like an executable program. It consists of an executable operating system configured with services and other programs. A container, on the other hand, is an active entity, like a process. In fact, it actually runs as a process on the host operating system. The container process runs the operating system and provides the services and programs made available from the associated image.

Linux Containers [6], LXC, are the Linux community's attempt to create their own version of a technology that has been around for several years. Two older examples are Solaris Zones [9] and FreeBSD Jails [4]. Linux containers being relatively new are still in development and have not fully addressed some issues such a security. However, they formed the basis for early Docker implementations, and so they are included here. Docker has since moved on to their own model.

Docker containers [3] almost immediately proved extremely useful, and were quickly deployed in many production shops. One good reason for the popularity of

Docker is that it is much less resource intensive to spin up hundreds of containers on a single system, than to do something similar with virtual machines. Additionally, it is possible to develop software in a container on a test machine and then deploy the container with the developed software onto the production machine - thereby ensuring that the production environment is identical to the development environment. Since there is such a large Docker community, their containers are rapidly becoming production level software. For these reasons, we chose Docker containers as the basis for VCML2.

JSON [5] originally became a de-facto standard among web developers because of its simplicity and its obvious relationship to JavaScript. Mostly due to the simplicity, its popularity spread to other communities, e.g. Python, where it has largely displaced markup languages such as XML. Indicative of this popularity is that Python has a JSON module that is part of the standard distribution. Thus it was an obvious choice for our project for the system configuration language.

Python [7] is a scripting language. Given the ubiquitous nature of Python, it seems unnecessary to explain our use of it in this project. That is particularly true given Python support for JSON. Our current project consists of a handful of scripts that can be used to build, deploy, and manage virtual clusters configured with VCML2 and running on Docker containers. These scripts, on average, consist of about 120 lines of executable code each. We make those scripts freely available, not only to our colleagues and students, but also to anyone who would like a copy.

2.2 VCML2

As was stated previously, the goal of this project was to be able to build a wide range of containers - from a couple of nodes on a single network to multiple clusters with routing between them. Additionally, we wanted to be able to do this on a single computer running any of Linux, Mac OS X, or Microsoft Windows. Specifically, we wanted to provide an easy to use toolset that would quickly create a virtual system for the user on the host platform of their choice. We envisioned that the user would sketch the configuration they wanted to deploy, express that configuration in a simple markup language, and then run a script that would build and deploy that configuration. We also wanted to provide management functionality in terms of routing and users and some file system support.

To accomplish this goal, we first developed the language that allows the user to express the configuration in a JSON file. To keep the language simple, there are only two types of entities - *networks* and *nodes*. Each type of entity has a set of attributes.

Networks have a *name*, a *driver*, and a *subnet* attribute. The *name* attribute is used by the *nodes* to specify which *networks nodes* are on. It makes configuring the Docker containers more convenient. With regards to the *driver* attribute, we currently have only tested with *bridge*, but it can use any that the user has support for on their system. The *subnet* is represented in CIDR notation.

Nodes ultimately become containers that represent computers and/or routers attached to a network. In the JSON configuration file they have the following attributes: *name*, *hostname*, list of *networks* to which they are attached, *image*, *volume*, and *router*. As with *networks*, the *name* attribute is a Docker convenience, and we usually have it match the hostname, although that is not a necessity. The *image* is the virtual disk from which the OS and all its facilities will be booted. It should be noted that executables, such as MPI, that are resident on the host platform can be mounted into the virtual cluster nodes, instead of having to do an install into the node. The *volume* is a colon-separated pair giving the volume on the physical host that is mounted on the container when it boots up. The *volume* attribute is optional. *router* is a boolean attribute that indicates if the node performs routing functions possibly in addition to being a compute node.

Once the desired JSON system configuration file is created, the Python *dbuild* script can be used to build and start the system running. *dbuild* must build the networks, build the nodes (containers) from their associated images, and make sure the nodes are attached to the appropriate networks. After building the system, *dbuild* will configure routing. Any node that is a router has to have routing turned on and have their routing tables set up. For nodes that are not routers, there has to be minimal setup of their routing tables (which network/router to use). In setting up the routing tables, our assumption is that the user wants each node to be able to communicate with all nodes it could possibly communicate with in the given configuration. So, if there is any communication path between two nodes (even going through multiple routers), we set up routing tables. If the user does not want routing to occur between two nodes, then they must set the routing tables by hand or else reconfigure the system. For example, in Figure 1, m2 is configured as a router. This means that m1 can communicate with m4 or m3 through m2. If the user does not want m1 to be able to communicate with m3, then they either need to reconfigure the system as in Figure 2 where there is no m2, or if they need m2 as a compute node, then they could configure the system as in Figure 3. If the user does not want to change the configuration and does not want the default routing tables provided by our scripts, then they will need to configure their own routing tables.

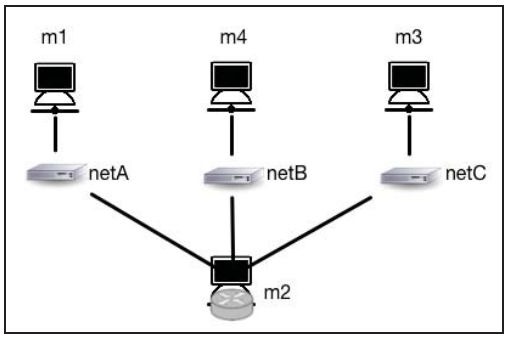


Figure 1. All nodes can communicate with each other.

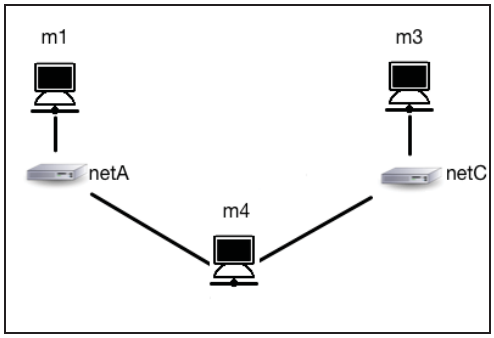


Figure 2. Node m1 and node m3 cannot communicate with each other.

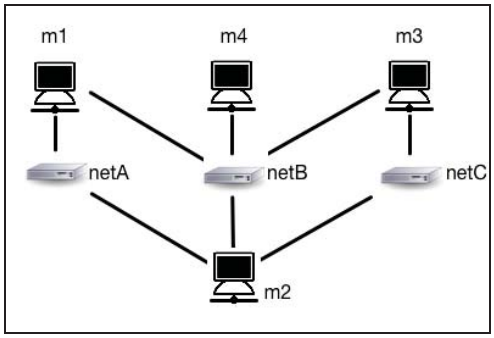


Figure 3. All nodes can communicate with nodes m2 and m4, but m3 and m1 cannot communicate with each other.

In addition to routing you might want root to be able to ssh among nodes. Therefore *dbuild* sets that up automatically. Also, you might want to be able to run something such as MPI, and you don't want the user to have to set up ssh keys, etc., so *dbuild* also sets up host-based authentication.

The final item that must be considered in building a system is the image that will be run on a container. There are three possibilities for obtaining an image. The first, and simplest, is to just use the image we provide which will automatically be pulled from an online repository when you do the first build of a system. If, however, you wanted to work offline, and wanted to make sure you had our image prior to building the system, then you could do an appropriate *docker pull* command. And finally, if you want to build your own image, then we provide a small **Dockerfile** file that you can customize and use in an appropriate *docker build* command.

As was mentioned previously, there are five small Python scripts for building, deploying, and managing the virtual systems. *dbuild*, as described above, fully builds out and starts the configuration described in a config.json file. *dstop* will leave the configuration built, but stops the execution of the nodes including the routers. *dstart* can be used to start the configuration up again if it is already built. *dremove* does a stop and removes all the components - nodes and networks. *duseradd* can be used to add a user to all currently active nodes providing the same valid user id on the running container as you have on the host machine.

. Admittedly, you could do the same thing we provide with VCML2 by installing and using the extra tools provided by Docker and Weaveworks [10] (e.g., Weave, Compose, Swarm, etc.). However, along with the need to install and configure additional software, we found there to be a fairly steep learning curve to using these tools. Additionally, Docker Compose - the tool that allows you to specify the system configuration - is not supported for Microsoft Windows. Unlike the Docker tools, to use VCML2 you install Docker, download our five Python scripts, create your JSON configuration file, run *dbuild*, and the system is ready to be used. For a simple network, this whole process can be done in under ten minutes. And it will work on Microsoft Windows, Linux, and Mac OS X. The following section demonstrates VCML2's simplicity.

3 Example Virtual Systems

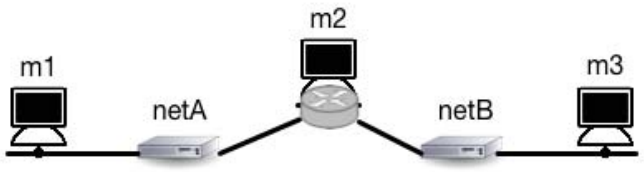
Shown below (Figures 4 - 6) are three architecture models. The simplest is the single cluster with two nodes on a shared network. The second example is of two separate systems on disparate networks that are capable of communicating with each other through a shared router. The final example is of four separate systems with two shared routers. The left column of the table contains the diagram of the proposed system, while the right column contains the JSON necessary to describe it.

In the JSON of Figure 4 there is only one network, netA, in the NETWORKS section. In the NODES section, both m1 and m2 are described along with the networks accessible to them and whether or not they are nodes capable of acting as routers. In figure 5, node m2 is somewhat more interesting in that it is configured as a router that is attached to both netA and netB. In figure 6, m5 is an example of a node where *dbuild* must be careful in setting up the routing table, because m5 can access two routers.


```
{
   "NETWORKS": {
      "netA": {
         "driver": "bridge",
         "subnet": "172.16.0.0/26"
      },
   },

   "NODES": {
      "m1": {
         "hostname": "m1",
         "image": "vcml/vcml2",
         "router": false,
         "networks": ["netA"],
         "volume": "/nfshome/home/scott:/home/scott"
      },
      "m2": {
         "hostname": "m2",
         "image": "vcml/vcml2",
         "router": false,
         "networks": ["netA"],
         "volume": "/nfshome/home/scott:/home/scott"
      },
   }
}
```

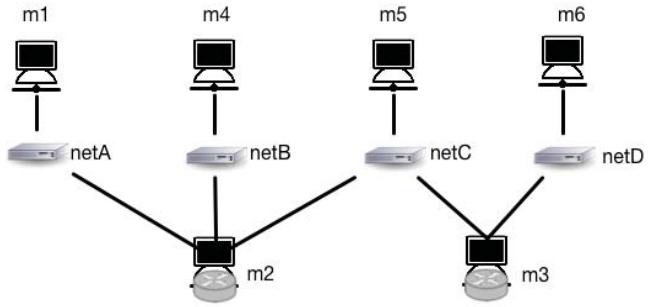
Figure 4. Single Cluster architecture with two nodes and a shared network and the JSON file that describes it.




```
{
  "NETWORKS": {
    "netA": {
      "driver": "bridge",
      "subnet": "172.16.0.0/26"
    },
    "netB": {
      "driver": "bridge",
      "subnet": "172.16.0.64/26"
    },
  },

  "NODES": {
    "m1": {
      "hostname": "m1",
      "image": "vcml/vcml2",
      "router": false,
      "networks": ["netA"],
      "volume": "/nfshome/CNL/home/scott:/home/scott"
    },
    "m2": {
      "hostname": "m2",
      "image": "vcml/vcml2",
      "router": true,
      "networks": ["netA","netB"],
      "volume": "/nfshome/CNL/home/scott:/home/scott"
    },
    "m3": {
      "hostname": "m3",
      "image": "vcml/vcml2",
      "router": false,
      "networks": ["netB"],
      "volume": "/nfshome/CNL/home/scott:/home/scott"
    },
  }
}
```

Figure 5. Two separate systems capable of communicating via a shared router and the JSON that describes them.




```
{
  "NETWORKS": {
    "netA": {
      "driver": "bridge",
      "subnet": "172.16.0.0/26"
    },
    "netB": {
      "driver": "bridge",
      "subnet": "172.16.0.64/26"
    },
    "netC": {
      "driver": "bridge",
      "subnet": "172.16.0.128/26"
    },
    "netD": {
      "driver": "bridge",
      "subnet": "172.20.0.0/24"
    },
  },

  "NODES": {
    "m1": {
      "hostname": "m1",
      "image": "vcml/vcml2",
      "router": false,
      "networks": ["netA"],
      "volume": "/nfshome/home/scott:/home/scott"
    },
    "m2": {
      "hostname": "m2",
      "image": "vcml/vcml2",
      "router": true,
      "networks": ["netA","netB","netC"],
      "volume": "/nfshome/home/scott:/home/scott"
    },
    "m3": {
      "hostname": "m3",
      "image": "vcml/vcml2",
      "router": true,
      "networks": ["netC","netD"],
      "volume": "/nfshome/home/scott:/home/scott"
    },
    "m4": {
      "hostname": "m4",
      "image": "vcml/vcml2",
      "router": false,
      "networks": ["netB"],
      "volume": "/nfshome/home/scott:/home/scott"
    },
    "m5": {
      "hostname": "m5",
      "image": "vcml/vcml2",
      "router": false,
      "networks": ["netC"],
      "volume": "/nfshome/home/scott:/home/scott"
    },
    "m6": {
      "hostname": "m6",
      "image": "vcml/vcml2",
      "router": false,
      "networks": ["netD"],
      "volume": "/nfshome/home/scott:/home/scott"
    },
  }
}
```

Figure 6. A complex system including four disparate systems with two shared routers and the associated JSON description.

4 Conclusions

For all the times when it would be convenient to simulate a system of networked routers and compute nodes on your desktop or laptop, we have provided a relatively simple, easy technique for specifying, building, deploying, and managing possibly complex virtual systems using Docker containers. The technique, called VCML2 uses JSON with only two entities for describing the system configuration. Building, deploying, and managing the specified system is accomplished with five small Python scripts that we make freely available to the community. VCML2 works on Linux, Mac OS X, and Microsoft Windows - a necessity to us as all three operating systems are in use by different members of our team, consisting of both faculty and students. While VCML2 was created to assist our team with research, one side benefit was that it is also useful for teaching. A Docker image alone can be used to replace the practice of having a virtual machine either on a jump drive [2] or via download [8] that makes it possible for students to do assigned programs in an environment comparable to whatever the professor needs for the class. However, VCML2 gives the added ability of allowing students to configure networked systems that can be used for a variety of purposes - for example to run MPI jobs, to run client-server jobs, or to learn about setting up routing to name a few.

While we had anticipated needing to change VCML2 so that it had the ability to run distributed virtual networked systems - i.e. nodes located on multiple physical hosts - so far it has met all of our needs. In the future, if we find that we need to add that capability, then we will investigate the technologies that we used in the prior VCML project and contrast them with newer technologies that may exist.

5 References

[1] Butler, Ralph, Lowry, Zach, and Pettey, Chrisila, "Virtual Clusters," **Proceedings of the Eighteenth International Conference on Systems Engineering,** August 2005, pp. 70 - 75.

[2] Butler, Ralph, Pettey, Chrisila C., and Lowry, Zach, "CPVM: Customizable Portable Virtual Machines," **Proceedings of the 44th ACM Southeast Conference**, March 2006, pp. 616 - 619.

[3] Docker. https://www.docker.com/

[4] FreeBSD Jails https://www.freebsd.org/doc/handbook/jails.html

[5] JSON http://www.json.org/

[6] Linux Containers. https://linuxcontainers.org/

[7] Python https://www.python.org/

[8] Sayler, A., Grunwald, D., Black, J., White, E., Monaco, M., "Supporting CS Education via Virtualization and Packages: Tools for Successfully Accommodating "Bring-Your-Own-Device" at Scale," **Proceedings of the 45th ACM Technical Symposium on Computer Science Education**, 2014, pp. 313-318.

[9] Solaris Zones http://docs.oracle.com/cd/E36784_01/html/E36848/zones.intro-1.html#scrolltoc

[10] Weaveworks https://www.weave.works/company/

Distributed Objects based Programming Constructs for PGAS based High Performance C++

Salwa D. Aljehan and Arvind K. Bansal
Department of Computer Science, Kent State University, Kent, OH 44242, USA

Abstract -- *For high performance application software development on massive parallel processing, existing languages have to be augmented with new constructs and paradigms that exploit massive parallel computing and distributed memory models while retaining the user-friendliness. Available object-oriented languages for massive parallel computing such as Chapel, X10 and UPC++ exploit data and task parallelism at the process level in the PGAS memory model. However, they do not support automated class-template distribution, object migration and user-transparent dynamic growth of regions for load balancing. This paper describes new constructs that extends C++ with distributed class template distribution, dynamic regions, object cloning, object migration; and integrates data parallelism, task parallelism, automated class-template distribution, and object migration. The integration supports MIDD (Multiple Invocation Distributed Data) programming paradigm for user-transparent invocations of multiple copies of methods concurrently working on different data elements of the same distributed data.*

Keywords: C++, distributed programming, high productivity, object-mobility, programming language, PGAS.

1 Introduction

Currently available supercomputers have peta-scale (10^{15} instruction/second) capability. It is anticipated that by the end of the next decade, we will have exa-scale (10^{18} instructions/second) computing power. This processing power needs to be fully exploited to solve big data problems in health science, weather science, agricultural science, space science, managing the Internet of things and modeling the population-related problems at the global and regional scale that will generate huge amount of data. Productivity is the most important issue that faces high performance computing [14] against the backdrop of existing software library and the existing familiarity with the programming paradigms.

The processing of big data will require the development of user-friendly high-productivity programming tools that exploit massive number of processors. The development of such programming tools should be paradigm-friendly, and should be downward compatible to use the existing libraries. The development of such tools requires the integration of user-friendly paradigms such as event-based programming, object-oriented programming, web-based programming in addition to task parallelism and data parallelism currently being exploited on high performance computers.

Task parallelism splits a task into multiple subtasks that run on different processing elements concurrently; data parallelism broadcasts the same instructions to multiple processing elements to perform same operations on multiple data elements concurrently.

Currently available languages supporting large scale concurrent processing exploit data parallelism and task parallelism including spawning of multiple threads, and their integration. However, massive parallel processor configurations and available memory models pose issues about how to map problems and tasks among the processors to preserve efficiency due to synchronization introduced to handle race conditions and message passing overheads.

PGAS (Partitioned Global Address Space) is a popular model [3, 12, 17] for high-performance computing. In PGAS, the distributed address space is divided into multiple local spaces connected through a global address space. The local spaces support multiple concurrent threads each with their own data area and a common shared space called heap. The communication between local address spaces is done by using global address space and message passing. Compared to MPI (Message Passing Interface) that suffers from excessive overhead of message passing, the use of local memory and distributed partitioned address space improves the productivity and execution-efficiency in PGAS model [15].

Currently available high-performance and high-productivity computing languages on PGAS such as Chapel [8, 9], X10 [10, 11] and UPC++ [18] support global partitioning of distributed data like distributed arrays, SPMD (Single Program Multiple Data) paradigm, task parallelism, asynchronous computation and invocation of remote threads. UPC++ also supports runtime distributed memory allocation. While these languages support object-oriented programming, they do not incorporate object distribution and mobility; and remote method invocation described in Emerald [4, 14], Java and other agent based languages [3]. These languages also do not support dynamic distribution of objects and class templates for dynamic load management. In addition, they do not support the constructs that can be derived by the integration of object-oriented programming, task parallelism and data parallelism.

This paper describes extension of C++ by developing new programming constructs and paradigms that integrate object-

oriented programming, compile-time user-transparent distribution of class-templates to a set of processing nodes, object migration, object cloning, task parallelism and data parallelism. The integration of these programming paradigms supports MIDD (Multiple Invocation Distributed Data) in which a distributed class-template is automatically distributed to a region (possibly dynamic) that is set of logical processing nodes called places in X10 [10, 11], and multiple copies of methods in the same distributed-class are invoked concurrently to process different elements of distributed data. The major contributions in this research are as follows:

1. Incorporation of sets of static as well as dynamic logical regions that can dynamically grow and shrink to accommodate load balancing in a user-transparent manner.
2. Incorporation of different types of classes for distributed computing: *distributed class*, *local class* and Emerald like *flat class* [4].
3. New programming constructs integrating object-mobility, cloning and distribution; task parallelism; and data parallelism for C++ and UPC++ languages.
4. Incorporation of higher level primitives such as monitor as in Emerald [4].

The rest of the paper is organized as follows. Section 2 describes the PGAS model, current languages for PGAS model and object-mobility in Emerald. Section 3 presents the abstract concepts. Section 4 describes the extended data abstractions. Section 5 presents the extended control abstractions. Section 6 illustrates the new constructs using examples. Section 7 presents related work. The last section concludes the paper.

2 Background

The "Partitioned Global Address Space" (PGAS) [3, 12] memory model has been proposed to overcome the limitations of the shared and distributed memory models. Overall address space is partitioned into multiple local spaces each having their own heap. These local spaces are connected through a shared global address space. Multiple threads execute locally, and can access remote locations asynchronously. The PGAS model supports SPMD control model. The data structures in this model can be distributed across address spaces. A distributed array is a data-space in shared memory such that different subranges are mapped on different *places* of local activities to exploit data-parallelism.

2.1 PGAS based high-productivity languages

There are many PGAS based high-productivity programming languages such as Chapel [8, 9], X10 [10, 11], UPC++ [18] that support object-oriented programming.

Chapel is a multithreaded high productivity computing language. It adopts a global view model, which means that a program starts with one thread, and based on the construct written by the programmer new threads can be spawned [3]. Data distribution and logical partitions are static and user-defined to map onto different architectural configurations. Chapel supports both data parallelism and task parallelism. Parallel and distributed data structures are supported by shared address space that connects various local partitions. The constructs *forall-loop*, *domains*, *ranges* and *array* are the basic data parallel features in Chapel.

X10 [10, 11] is a statically typed, object-oriented, high-performance and high-productivity computing language. It extends a sequential core language using features called *places*, *activities*, *clocks*, *arrays*, and *struct types* [11]. Like Java and C++, X10 makes use of *classes, structs*, and *interfaces*. X10 supports single inheritance [11]. Methods can be inherited and overridden in the subclasses. The reserved words "private", "public" and "protected" are used to control the visibility of a method.

Central to X10 is the concept of a *place*, a collection of data and resident lightweight threads called "activities" [10]. *Places* map to a local processor, and contain a bounded number of *activities* and a bounded amount of storage. Creation of multiple places allows cluster-level parallelism. X10 introduced the notion of asynchronous activities for creating threads locally and remotely. X10 uses atomic statements to secure data-values limited only to the local scope. Since multiple processes need to be coordinated, it is necessary for X10 to use multiple barriers.

UPC++ [18] provides three main functionalities: 1) an object-oriented model for C++ language; 2) a collection of parallel programming constructs, not included in C++, to support high performance execution; and 3) a transition to PGAS programming through interoperability with other similar systems. The execution model of UPC++ is "Single Program Multiple Data" (SPMD). UPC++ implements asynchronous features through distributed-memory systems similar to C++11 standard asynchronous libraries for shared-memory systems. Synchronization is provided using primitives such as *barriers, fences* and *locks* to facilitate parallel programming. Remote function invocation allows asynchronous remote function with a single thread ID that is a *place* or a group of threads.

2.2 Emerald

Emerald is a flat object-based programming language [4, 14] that supports object-mobility [3] in a networked environment. Location-independent addressing allows object-mobility from node to node [14]. All entities are treated as objects. Concurrency is supported between objects and within an object. Variables shared by operations are synchronized using high level control-abstraction *monitor*. Emerald's runtime system is responsible for the location and transfer of control to the target object.

2.3 Notations

We denote the data and control abstractions in *italics* and reserved words within double quotes "..". The non-terminal symbols in the data-abstractions are enclosed in angular-brackets, are written in italics, and are self-

explanatory. For example, the non-terminal symbol <*distributed-class*> discusses about the declaration of distributed class. The grammar rules are written using extended BNF when required: sets are written using curly brackets {…}, optional use is written using square brackets [..]; alternatives are written using parenthesis and vertical bar (..|..); multiple occurrences are denoted by {…}+.

3 Abstract concepts

The motivation behind the proposed DOPC++ (Distributed Object based Programming for C++) language is to incorporate high level user-friendly constructs that provide integration of object distribution and mobility with task and data parallelism while retaining the user-friendliness in the PGAS model. DOPC++ introduces a high level abstraction, and hides the low level object mapping on physical processors to increase the usability and user-friendliness.

DOPC++ supports: 1) distributed creation and dynamic migration of objects; 2) communication between remote objects; 3) cloning of distributed objects; and 4) extension of the notion of region in Chapel. Unlike static regions in Chapel, the regions in DOPC++ can dynamically grow and shrink. It describes distributed class, distributed methods, dynamic migratory methods that can be remotely invoked for dynamic load balancing. User-transparent dynamic object migration facilitates load balancing and performance-improvement without loss of any functionality.

In DOPC++, a *place* is a logical computational entity like a virtual processor that is mapped to physical processor statically or dynamically. A *region* is a set of places and/or sub-regions that allow mapping of distributed data-abstractions.

Regions could be static or dynamic. A *static-region* is fixed during compile time. A *static region* is mapped to physical processors at compile-time, and does not grow or shrink during execution. Declaring a static-region allows every place in that region to get a copy of a distributed class-template and each place within this region will create a copy of the object in response to object-creation instruction. Unlike other PGAS based languages, DOPC++ also supports *dynamic regions*. A *dynamic-region* grows and shrinks at runtime based upon: 1) the computational need of the executing task; 2) resource-availability of the HPC system; and 3) load-balancing of the physical processors. Operating system performs runtime allocation of the places and the physical processors for dynamic regions.

Dynamic regions are bounded by a *problem space*. A *problem space* is a fixed set of places in which a dynamic region can grow. The rationale for the boundedness of dynamic regions is to limit the spread of very large problems that may affect the execution efficiency of other tasks. A dynamic region grows and shrinks during runtime, but cannot go beyond the *problem space*.

A *distributed data-abstraction* is distributed within a region with compiler and operating system deciding the granularity based upon: 1) available memory; 2) processing speed; 3) processor load; and 4) processor configuration table.

System level utilities inserted by the compiler take care of the mapping at run time. Objects methods and data elements migrate between processors dynamically in a dynamic region to balance the process-load. Within a dynamic region every place gets a copy of the class-template automatically. Similarly, the object-creation checks the number of places at runtime, and invokes objects in every place of the dynamic region concurrently. The use of dynamic-regions allows migration of the objects, methods and data elements potentially to any *place* for load balancing. Multiple objects could be invoked concurrently within a region, and each can work on an array of data elements independently.

A region provides an added scope rule for the visibility of objects and classes. A class declared within a region is visible only in the places included in the region. This also limits migration of objects to places within the region where a class has been declared. However, for dynamic regions, the migration pattern of objects changes dynamically. For the dynamic regions, the operating system keeps a mapping table of logical places and regions to the physical processors.

Places exchange information with other places using PGAS shared address space, and require constructs to access address space in remote places within the same region. The migration of objects can be place-to-place, many places–to-one place, one place-to-many places, region-to-place, or region-to-region. When an object migrates from one place to another place <*place*> in the region <*region*> then the runtime system utilities will creates an alias <*region*>.<*place*>.<*object-name*> to point to the same object. The migration of objects from a region requires a broadcast of the code part of the object to the destination-region from one of the places in the source region. The data part of an object migrates based upon the type of mapping and available physical processor-load.

Communication between objects is done through a *remote invocation*. The parameters that are passed during invocations can be objects themselves. Parameter passing uses *call by object-reference* or *call by object move* as in Emerald [14]. In *call by object-reference*, the identifier that allows accessing the object remotely is passed as parameter. *Call by move* involves migration of an object to the remote place. In addition, the interface of any method to execute remotely is done by accessing the object.

DOPC++ supports different type of classes: *1) distributed-class*; 2) *flat-class* as used in Emerald [14]; 3) *local-class; and 4) regular C++ class.* A *distributed-class* has distributed data elements and/or distributed methods. The scope of a *distributed-class* is within a declared region. A user gives an initial region to start with.

A method embedded in a flat-object is invoked at the time of creation of the object. Active objects can share information using a *blackboard*. A blackboard is a synchronized shared address space that is shared between multiple threads. A *shared blackboard* can be in the global shared address space or it could be distributed among the places in a region. A *local-blackboard* is shared between local threads in a place. A *distributed-blackboard* is shared

between the threads in the places within a static or dynamic region. A *global-blackboard* is shared between all the threads among multiple regions. *Local-blackboards* are used for sharing information between threads within the same place. All other processes sharing the blackboard are suspended to achieve synchronization until the writing process finishes writing on the blackboard. The synchronization-construct *monitor* ensures mutual-execution in the shared blackboard to serialize writing. The lock for a local blackboard is kept locally in the same place where the thread activities are taking place. The lock for a distributed-blackboard is kept in a shared address-space. When a thread writes on a blackboard, the *lock* is first captured to ensure that no other thread can write on the blackboard.

4 Extended data abstractions

A distributed data abstraction is declared after the specification of problem space and region (or sub-regions). Problem-space is declared as "problem space" {<*place-identifiers*>}+. A region is declared as "region" <*region-name*> {<*place-identifiers*>}+. A distributed-data-abstraction is declared as [<*synchronization-type*>] <*compile-type*><*data-abstraction*><*location-type*>.

Compile-type could be *static* or *dynamic*. Distributed-class or distributed data structures could be *distributed*, *local*, or *global*. Location-type specifies the region or place where data-abstraction is located.

4.1 Distributed-class declarations

A *static distributed-class* is distributed in a static-region declared at compile-time. The name of the distributed-class is unique within a region, and an object-name is qualified by the place-identifier to make it unique. For dynamic distributed-class, a class-template is created in every place of the initial dynamic region at compile-time. A class-template migrates at run time as the region grows or shrinks based on the load balance. In the case of nested classes (inheritance), whole class-hierarchy migrates. A *flat-class* allows objects to be easily distributed since there is no hierarchy. A class is considered hierarchical by default. A *local-class* resides in a specific place. An object-instance of a local-class is created in the same place without any possibility of object-migration. One motivation of using the local-class is to reduce the overhead of migration.

A *distributed-method* can be declared in a region or a specific place. A method-declaration in a region results in a copy of the method in each place within the region. The location-name of a method-declaration can be different than the location-name of *the* corresponding class-declaration if the methods are invoked remotely.

A <*distributed-class*> is declared as ("static"| "dynamic") ("distributed" | "flat" | "local") <*class-declaration*> <*region-declaration*>. A <*flat-Class*> is declared as "flat" <*class-declaration*> <*region-declaration*>. A <*local-class*> is declared as "local" <c*lass-declaration*><*place-declaration*>.

A <*distributed-method*> is declared as "distributed-method" <*method-declaration*> "in" ("region" | "place") <*location-name*>. A *distributed-method* is declared as <*data-type*> followed by the <*method-name*> that is followed by the <*typed parameter-list*>.

4.2 Distributed data structures declarations

Elements of a distributed-array (or distributed-vector) are split in different places of the region based upon user-defined granularity. *Granularity* is the number of consecutive elements in an array (or vector) that occur in the same place. The granularity for each dimension can be different.

A <*distributed-array*> (or <*distributed-vector*>) is declared as "distributed" (or "global") followed by "array" (or "vector") followed by <*array-name*> (or <*vector-name*>). Array-name is followed by *dimension-list. Dimension-list* is a non-empty sequence of the form <*dimension*>:<*granularity*>. The distributed-array (or distributed-vector) declaration is followed by location declaration ("in region" | "in place") <*location-name*>. A *global-array* is allocated in the shared partitioned space.

The elements of distributed-array are processed in parallel using multiple invocations of a method in the resident copies of the objects in different places of the region. This paradigm is called MIDD (Multiple Invocation Distributed Data). When a distributed-array is defined as "global" then the elements are shared in global address-space.

4.3 Shared data and synchronization

A <*distributed-blackboard*> is declared by "distributed blackboard" <*blackboard-name*> "at region" <*region-name*>. A<*local-blackboard*> is declared as "blackboard" <*blackboard-name*>. A <*global-blackboard*> is declared as "global" <*blackboard-name*>.

Data-objects are synchronized by tagging with a reserved word "synchronized" preceding the declared data-objects. The granularity of the synchronization is at the data abstraction level. A *monitor* control-abstraction is used to provide mutual exclusion of statements working on synchronized data-objects. All the statements occurring within the block following the reserved word "monitor" form the critical section. The use of monitor secures the lock(s) associated with the corresponding synchronized data-objects before executing the embedded methods. Monitor creates a queue for the processes waiting to access the corresponding synchronized data-object(s). This restriction imposes sequentiality. For SPMD operations on distributed data-objects, this restriction can cause serious efficiency overhead. To avoid this, the default mode for aggregate data-abstractions is asynchronous.

5 Control abstraction extensions

Major extensions are *migrate*, *remote invocation of methods, clone*, *get-object-location, move-object, remotely-delete-object, get-remote-value, blackboard-put, blackboard-get* in addition to SPMD statements working on distributed-

arrays, distributed-vectors, and distributed-blackboards.

The purpose of object migration is to distribute the workload. During object-migration, code-template is broadcast to all the places in the destination region. However, data-area of the object is migrated to balance the data distribution in limited number of places of the destination region. Object-migration is declared as "migrate" <*object-name*> "to" (<*region-name*> | <*place-name*>).

A remote-method is needed to perform some computation to balance the computational load or to use a specific resource present at a specific place. A remote-method is invoked using <*place-name*>**.<**<*object-name*>**.<***method-name*> ( ).

Cloning is used to duplicate an object for migration to another place or region. However, the object should be clone-able to execute this method. An object that is an instance of a local-class is fixed in one place and cannot be copied to another place. Cloning of a distributed-object is declared as "clone" <*object-name*> "*in*" (<*region-name*>|<*place-name*>) "to" (<*region-name*> | <*place-name*>).

The possible name-conflict during migration of objects or cloning of objects across regions needs two global tables for each region: 1) a table of object-names for each region that is checked when an object is migrated to avoid duplication; and 2) a correspondence table between the object-name in the source-region and the object-name in the destination region.

There are two control-abstractions that support SPMD paradigm: 1) standard *forall* statement that works on every element of a distributed-array (or distributed-vector) concurrently; and 2) *foreach statement* that works on elements in a subset concurrently. A *foreach* statement is written as "foreach**"**<*variable-name*> "in" (<*set*> | <*region-name*>). The set could also be a set of places marking a sub-region. *Foreach* construct invokes multiple copies of a method distributed in a set of places in a region. For communication between places, objects can be passed as arguments.

DOPC++ uses three additional built-in methods: *size, length* and *Indexset*. The method *size* computes the number of elements in a distributed-array that can be allocated to each place by knowing the place capacity and load. The method *length* returns the number of allocated elements of dynamic distributed array (or vector) at each place. *Indexset* computes the set of indices of a distributed array element in a place.

6 Programming examples

In this section, we illustrate extended constructs using two examples. Example 1 illustrates static-region, MIDD (Multiple Invocation Distributed Data), object-distribution, object-migration, and the use of distributed-arrays and static distributed-class. Example 2 illustrates dynamic-region, object-cloning, synchronization using monitor, static distributed-class and dynamic distributed-class. We have used extended C++ syntax in the programs.

6.1 Example 1 - object migration and MIDD

The program in Figure 1 illustrates *distributed class* and *object migration* from a place in a region *R* = {*1, 2, 4, 5*} to another region *S* = {*3*} to utilize a printer resource available in the region S. Declaring a "static distributed-class" in region *R* allows every place in region *R* to get a copy of the distributed class-template *dictionary*. Four class templates, one for each place within the region *R* are created transparently by a compiler. The program has a distributed array *word* and four functions: *main*, *lookup*, *store* and *printData*. The program creates distributed objects for static distributed-class *dictionary*, and calls *store* and *lookup* methods in a region R = {*1, 2, 4, 5*}, and the data is printed in a different region S = {*3*}.

```
problem space = {1, 2, 3, 4, 5};
region R = {1, 2, 4, 5}; region S = {3};

static distributed class dictionary at region R
 { Public:
        int n = 100; gs = 25; // gs is grain-size of word in a place
        dynamic string distributedArray word[n] at region R;

        distributed void store ()
          { cout<< "enter new words";
                for (i = 0; i < gs; i++) cin >> word[i]; // read words
          }

        distributed void lookup () in region R
                { string w;
         Boolean found = false;
                Cout << " what is your word to look up:"; cin >> w;
                foreach i in indexset(word) // search word
                        if (word [i] == w) { found = true; break;}
                If (found) return('found'); else return('search fails');
          }

        remote printData ( ) in region S
          { int i, m = 0;
            foreach p in R // for each place in region R do
               {  m = p.word.size(word.length);
                  for (i = 0; i < m; i++) // read the words
                        cout << word[i]; // print the words
               } // end foreach
          }
 } //end distributed class dictionary

int main ()
 {  int value;
    Boolean keepLooping = true;
    new dictionary d; // create object d in every place in R;
    cout << "1 to store new words; 2 to look up; 3 to print:";
    while (keepLooping) // read the commands
         { cin >> value;
           switch (value)
                {  1: store (); break;
                   2: lookup (); break;
                   3: keepLooing = false; d.printData( ); break;

         } // end reading the commands
 } //end main
```

Figure 1. Illustrating object migration and MIDD

The function *lookup* looks at various places concurrently. The number of spawned threads depends upon operation system according to the load balancing. The function *store* reads one word at a time, and stores in the distributed array *word*. The function *store* spawns concurrent

threads in all four places of the region R. Migration creates an object in the place *3* that will print the distributed array *word* iteratively. The object *d* migrates to place 3, which has a printer resource. The remote-method *printData* spawns a thread to iteratively print the dictionary words.

The main program invokes three functions: 1) *store( )* to store the words in the distributed array word; 2) *lookup( )* to lookup a given word in the distributed array word; and 3) print function *printData()* to print the data. The main program terminates after printing the dictionary. The functions are invoked based upon the input value *value.*

The statement "dictionary d" in main program automatically creates multiple instances of objects *1.d, 2.d, 3.d, 4.d*, one in each place, within the region *R*. Each copy of the object works concurrently on the distributed array *word* during the execution of functions *store( )* and *lookup( )*. The local Boolean variable *found* is used to break the foreach loop after the successful search. To insert new words inside the distributed array *word*, multiple invocations for all objects are done concurrently. A built-in method *size* computes the number of words allocated in each place.

6.2 Example 2 – cloning and synchronization

The program in Figure 2 illustrates *dynamic distributed-class*, *object-cloning* and *synchronization*. The program has a dynamic distributed-class *math* in a dynamic region *R* with four initial places {*1, 2, 3, 4*}. Multiple copies of the class-template *math*, one for each place within the dynamic region R, are created automatically after the execution of the statement *math m* in the main program. The method *m.add( )* and the distributed array *num* are created in the places *1, 2,* and *3* constituting the sub-region *R1*.

The distributed array *num* within the region *R1* is added by spawning concurrent threads in different places in the region *R1*. The result is stored in the local copy of the variable *accum* in each place of the region *R1*. The values stored in local copies of *accum* are added to the synchronized global shared variable *sum* to get the cumulative total*;* concurrent threads are spawned in the places within the region *R1*. The synchronization between concurrent active threads is maintained using a monitor.

A static distributed class *test* is created in the region *R2* to derive *even* or *odd* numbers using *isEvenorOdd*(). Cloning of the object *m* in the main program creates distributes copies of the object *testarray* in the places {6, 7, 8}. The method *isEvenorOdd*( ) filters and stores the numbers in two distributed arrays: *even* and *odd.* The built-in method *size* retrieves the number of data-elements of a distributed-array stored in a place based upon the reconfiguration table. The built-in method *length* gives the number of data-elements in the distributed-array.

In the method *isEvenorOdd*, monitor is needed for the shared synchronized index-variables *j* and *k* that are being updated by multiple concurrent threads, each trying to update distributed-arrays *even* and *odd* concurrently.

```
problem space = {1, 2, 3, 4, 6, 7, 8};
dynamic region R = {1, 2, 3, 4};
dynamic region R1 = {1, 2, 3};
synchronized int sum = 0; // stores final sum-value

dynamic distributed class math in dynamic region R
{   Public:
    int n = 20, gs = 7; // gs is grains-size of num in a place
    static int distributedArray num[n:gs] in region R1;

     distributed void add (distributedArray <int> num)
                                          in region R1;
     int i = 0, accum = 0; // initialize variables
     { gs = num.size(num.length);
          for (i = 0; i < gs; i++) accum = accum + num[i];
               monitor { sum = sum + p.accum;} // end monitor
   } // end method add
}//end class

region R2 = {6, 7, 8};
static distributed class test at region R2
{  Public:
        synchronized int j, k = 0;
        int n = 20 , gs = 8;
        static distributedArray even [n:grain] at region R2;
        static distributedArray odd [n:grain] at region R2;
        static distributedArray remainder[n:grain] at region R2;

  distributed void isEvenorOdd(distributedArray <int> A);
        {   gs = A.size(A.length);
            forall (i = 0:gs)
                  {  remainder[i] = A[i] %2;
                     monitor
                       {    if (remainder[i] == 0) even [j++] = A[i];
                            else odd[k++] = A[i];
                       } //end monitor
                  } //end forall
        } //end method isEvenorOdd
} //end distributed-class test

int main ()
{   int value;
    new math m in region R; // create object m in region R
    m.add(m.num); // add the element in the array num
     new mClone = clone m in region R2; // clone m
     new test testarray in region R2;
     testarray.isEvenorOdd(mClone.num); // test
} // end main
```

Figure 2. Illustrating object cloning and synchronization

7 Related works

Three types of high-productivity languages are being developed: 1) MPI based languages such as mpiJava [2] and PObC++ [16]; and PGAS based languages such as Chapel [8, 9], X10 [10, 11], and UPC++ [17]; 3) CSP (Communicating Sequential Processes) based such as OCCAM [13] and Rain[7].

In mpiJava [2] and PObC++ [16], the integration of MPI and object-oriented programming supports: 1) point-to-point communication; 2) process topologies; and 3) dynamic process creation. PObC++ also supports automatic dynamic process spawning when a flat-object is created. However, the overhead of message passing is an issue in MPI. MPI based languages do not provide the same type of productivity as PGAS based languages [15]. In addition, MPI based languages cannot exploit the data abstractions based upon the shared partitioned space.

Both UPC++ [18] and PObC++ [16] extend C++11 constructs to incorporate task and data parallelism while retaining object-oriented programming constructs of C++11. However, they lack: 1) object distribution; 2) object cloning; 3) object migration; and 2) integration of object-distribution with task and data parallelism.

Multiple integrations [5, 7] of C++ with OCCAM [13] have been proposed to integrate object-oriented programming in C++ with CSP-like parallelism [6]. However, these integrations do not support object-migration, object-cloning and automated distribution of distributed-class templates.

This paper extends the work of prior PGAS based languages [8, 9, 10, 11, 18] by introducing: 1) the concept of static and dynamic regions for better user-transparent load balancing; 2) integration of object-distribution and object-migration to exploit MIDD (Multiple Invocation Distributed Data) paradigm; 3) dynamic growth of regions; and 4) separation of logical notion of place and regions from physical processors. Many new constructs have been developed using these extension of concepts.

8 Conclusion and future work

We have extended the PGAS based high performance computing language development effort by integrating class-template distribution, object-mobility and remote method invocation with task and data parallelism. We have also extended the concept of region to include dynamically growing and shrinking regions to take care of load balancing. The integration of object-distribution and remote method invocations supports MIDD (Multiple Invocation Distributed Data) programming paradigm where multiple methods can be automatically distributed to work on distributed data-elements of an object in a distributed class. The separation of logical place and regions from physical processors provides more architecture independent high level programing that we believe will provide better adaptability of object-oriented high performance computing to desktop as high performance computing moves to desktops.

We need to investigate other process invocation paradigms in the models integrating CSP derivatives and C++ [5, 7] to look into alternate communication models. We are also looking into developing a translator that can translate these constructs to languages like UPC++ or X10. We are extending the language further for integrating event-based programming with object distribution.

9 References

[1] S. Aljehan. "DOPC++ - Extending C++ with Distributed Objects and Object Migration for PGAS". MS Thesis, Department of Computer Science, Kent State University, Kent, Ohio, USA, November 2015.

[2] M. Baker, B. Carpenter, G. Fox, S. H. Ko and S. Lim. "MPIJava: An Object-oriented Java Interface to MPI"; in Parallel and Distributed Processing, Vol. No. 1586, Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 748-762, 1999.

[3] A. K. Bansal. "Introduction to Programming Languages". 1st edition, 2nd print, CRC Press: Boca Raton, FL, USA, 2014.

[4] A. P. Black, N. C. Hutchinson, E. Jul and H. M. Levy, "The Development of the Emerald Programming Language"; Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages, pp. 11-1 - 11-47, 2007.

[5] D. Boles. "Parallel Object-Oriented Programming with QPC++"; Structured Programming, Vol. No. 14, 158 - 172, 1993.

[6] S. D. Brookes, C. A. R. Hoare and A. W. Roscoe. "A Theory of Communicating Sequential Processes"; Journal of the ACM, Vol. No. 31, Issue No. 3, 560-599, 1984.

[7] N. Brown. "Rain: A New Concurrent Process Oriented Programming Language"; Communicating Process Architectures, P. Welch, J. Kerridge, and F. Barnes eds, IOS Press, 237-251, 2006.

[8] B. L. Chamberlain, D. Callahan and H. P. Zima. "Parallel Programmability and the Chapel Language"; International Journal of High Performance Computing Applications, Vol. No. 21, Issue No. 3, 291-312, 2007.

[9] B. Chamberlain. "A Brief Overview of Chapel". http://chapel.cray.com/papers/BriefOverviewChapel.pdf , 2013. (last accessed March 20, 2016).

[10] P. Charles, C. Grothoff, V. Saraswat, et. el. "X10: An Object-oriented Approach to Non-uniform Cluster Computing"; ACM Sigplan Notices, vol. No. 40, 519-538, 2005.

[11] S. Crafa, D. Cunningham, V. Saraswat et. el. "Semantics of (Resillient X10)"; European Conference on Object-Oriented Programming (ECOOP 2014), Springer Berlin Heidelberg, 670-696, 2014.

[12] J. Diaz, C. Munoz-Caro and A. Nino. "A Survey of Parallel Programming Models and Tools in the Multi and Many-core Era"; IEEE Transactions On Parallel and Distributed Systems, Vol. No. 23, 1369-1386, 2012.

[13] M. Elizabeth and C. Hull. "Occam-A Programming Language for Multiprocessor Systems"; Computer Languages, Vol. No. 12, Issue No. 1, 27-37, 1987.

[14] E. Jul, H. Levy, N. Hutchinson and A. Black. "Fine-grained Mobility in the Emerald System"; ACM Transactions on Computer Systems (TOCS), Vol. No. 6, 109-133, 1988.

[15] J. Kepner, "HPC Productivity: An Overarching View"; International Journal of High Performance Computing Applications, Vol. No. 18, 393-397, 2004.

[16] E. G. Pinho and F. H. de Carvalho Jr.. "An Object-oriented Parallel Programming Language for Distributed Memory Parallel Computing Platforms"; *Science of Computer Programming*, vol. 80, pp. 65-90, 2014.

[17] S. Spetka, H. Hadzimujic, S. Peek and C. Flynn, "High Productivity Languages for Parallel Programming Compared to MPI, " in *DoD HPCMP Users Group Conference*, 2008. DOD HPCMP UGC, pp. 413-417, 2008.

[18] Y. Zheng, A. Kamil, M. B. Driscoll, et. el., "UPC++ : A PGAS Extension for C++, " in the *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium,* pp. 1105-1114, 2014.

FREDDO: an efficient Framework for Runtime Execution of Data-Driven Objects

George Matheou[1] **and Paraskevas Evripidou**[1]
[1]Department of Computer Science, University of Cyprus, Nicosia, Cyprus

Abstract— *In this work we introduce FREDDO, an efficient object-oriented implementation of Data-Driven Multithreading (DDM) model, and its evaluation on a 32-core machine using a suite of ten benchmarks. FREDDO is a C++ framework that supports efficient data-driven execution on conventional processors. FREDDO incorporates new features like recursion support and simpler programming methodology to the DDM model. Our aim is to create a robust system that can scale to HPC level.*

The performance evaluation shows that FREDDO scales well and tolerates scheduling overheads and memory latencies effectively. We have also compared our system with the DDM-VM, the OpenMP and the OmpSs frameworks on several benchmarks with different characteristics. The comparison results show that the DDM systems achieve better results than OpenMP and OmpSs. Furthermore, FREDDO achieves similar results with DDM-VM despite the fact that the former provides more functionalities. Additionally, our object-oriented approach reduces the size of the DDM applications to half.

Keywords: Data-Driven, Multithreading, Object-Oriented Programming, Recursion Support

1. Introduction

The switch to multi-core and many-core systems has created the need for efficient parallel processing in mainstream computing [1]. Conventional multiprocessor systems based on control-flow architectures suffer from long memory latencies and waits due to synchronization events [2]. As the number of cores increases in the future, traditional ways for achieving large scale computations will need to evolve from legacy models such as OpenMP and MPI [3], [4], [5]. Thus, efficient parallel programming models and architectures must be developed that will be able to efficiently keep all the available resources busy [6]. Such a model is the Data-Driven Multithreading (DDM) model [7] of execution.

DDM is a non-blocking multithreading model that allows Data-Driven scheduling on conventional processors. Data-Driven scheduling enforces only a partial ordering as dictated by the true data-dependencies which is the minimum synchronization. This is very beneficial for parallel processing because it exploits the maximum possible parallelism.

In this work, we present the FREDDO framework, an optimized object-oriented C++ implementation of DDM. The main contributions of this work can be summarized as follows:

- We introduce a new optimized implementation of the DDM model which provides several improvements over the previous implementations. Our aim is to create a robust framework that can scale to HPC level. Thus, the first step is to implement and evaluate our framework on a single-node system. Our next goal is the development of an efficient distributed system.
- We are providing the basic functionalities for supporting recursion in the DDM model. As a proof of concept, we have implemented three famous recursive algorithms: Fibonacci, Knight's Tour and NQueens.
- We improve the programmability of DDM programs through object-oriented programming.

FREDDO is evaluated on different number of cores and problem sizes (Small, Medium and Large) using a suite of ten benchmarks on a 32-core AMD processor. The evaluation showed that FREDDO scales very well across the range of the benchmarks and it achieves very good speedups. For the Large problem size, FREDDO achieves an average speedup of 3.98 out of 4, 7.79 out of 8, 15.33 out of 16 and 29.48 out of 31.

We have also compared our system with the DDM-VM [8], [9], the OpenMP [10] and the OmpSs [11], [12] frameworks. Our framework outperforms the OpenMP and OmpSs especially in the case of high-complexity applications. Finally, we observed that FREDDO achieves similar results with DDM-VM, despite the fact that the former provides more functionalities. Our new programming interface reduces the size of the DDM applications to half, compared to the size of the DDM-VM applications.

2. Data-Driven Multithreading

The Data-Driven Multithreading (DDM) [7] is a non-blocking multithreading model that allows data-driven scheduling on sequential processors. A DDM thread (called DThread) is scheduled for execution after all of its required data have been produced, thus no synchronization or communication latencies are experienced after a DThread begins its execution.

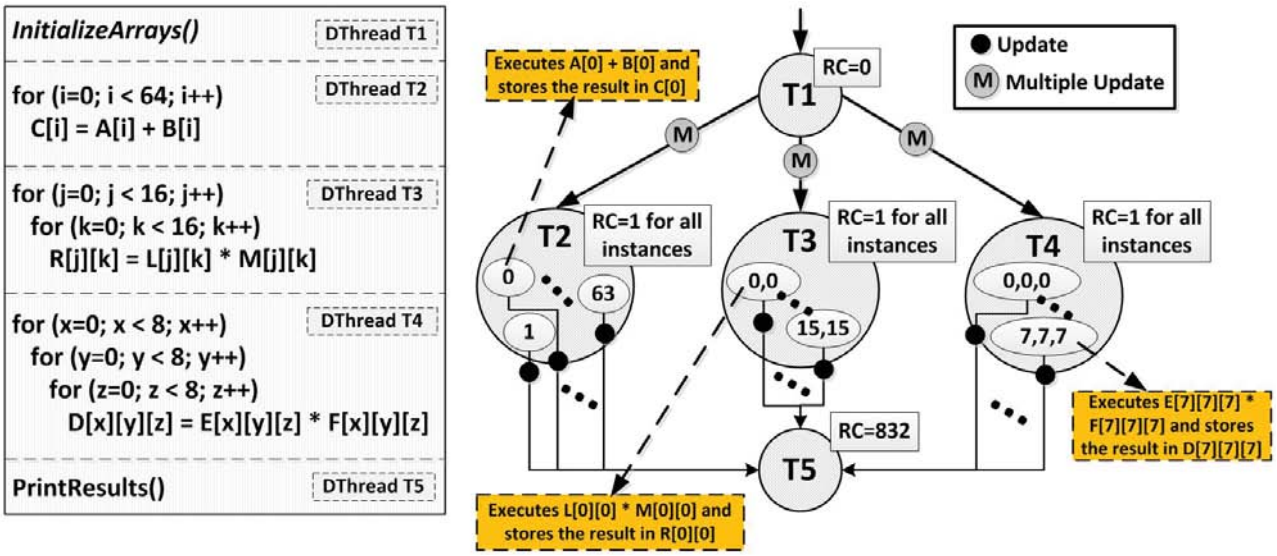


Fig. 1: Example of a DDM program.

In DDM, a program consists of the DThreads' code, the *Thread Templates* and the *Dependency Graph*. A Thread Template holds information about a DThread. The Dependency Graph describes the consumer-producer dependencies amongst the DThreads.

DDM is utilizing the Thread Scheduling Unit (TSU), a special module that is responsible for scheduling the DThreads in a data-driven manner. The TSU uses the Thread Templates and the Dependency Graph to schedule DThreads for execution when all of their producer-threads completed their execution. This ensures that all data needed by a DThread is available, before it is scheduled for execution.

The DDM model was evaluated by several software implementations. The first implementation, the Data-Driven Network of Workstations (D^2Now) [7], was targeting Networks of Workstations. That was followed by two other implementations, the TFlux [13] and the Data-Driven Multithreading Virtual Machine (DDM-VM) [8], [9], [14]. Both TFlux and DDM-VM were targeting data-driven concurrency on sequential multiprocessors. DDM was also evaluated by a hardware implementation [15], [16] where the TSU was implemented as a hardware peripheral for sequential multicore systems.

2.1 The Context and Nesting Attributes

The DDM's tagging system has been based on the U-Interpreter's tagging system [17]. The tagging system of DDM enables multiple instances of the same DThread to co-exist in the system. More specifically, it maps the tag of the U-Interpreter into a unique 32-bit integer, called the *Context*. This enables concurrency in re-entrant constructs, such as loops and function calls.

The DDM model allows the parallelization of nested loops that can be mapped into a single DThread by using the Nesting attribute [8], [16]. This attribute is a small number that indicates the loop nesting level for the DThreads that implement loops. FREDDO supports three nesting levels, i.e. the DThreads are able to implement one-level (Nesting-1), two-level (Nesting-2) or three-level (Nesting-3) nested loops. If a DThread does not implement a loop, its Nesting attribute is set to zero (Nesting-0).

2.2 The DDM Dependency Graph

The DDM Dependency Graph is a directed graph where the nodes represent the DThreads and the arcs represent the data-dependencies amongst them. In this work we categorize the DThreads as *simple* or *multiple*. A simple DThread has only one instance (i.e., its Nesting=0 and Context=0). A multiple DThread has Nesting>0 and several instances where each instance has a unique Context value. Each instance of a DThread is paired with a special value called Ready Count (RC) that represents the number of its producers.

An example of a DDM program is shown in Figure 1. On the left side of the figure, the pseudo-code of a synthetic application and its partitioning into five DThreads are depicted. From this code it is possible to observe a number of dependencies. DThreads T2, T3 and T4 depend on T1 which is responsible for initializing the data. Also, T5 depends on T2, T3 and T4 since T5 prints the output results generated by them. These dependencies form the DDM Dependency Graph of this application which is presented on the right side of Figure 1.

The RC values are depicted as shaded values next to the nodes. The RC value is initiated statically and is dynamically decremented by the TSU each time a producer completes its execution. A DThread is deemed executable when its RC value reaches zero, such as the DThread T1.

In DDM, the operation used for decreasing the RC value is called *Update*. Update operations can be considered as tokens that are moving from the producer to consumer instances through the arcs of the graph. *Multiple Updates* are introduced in order to decrease multiple RC values of a DThread at the same time. This reduces the number of tokens in the graph. For instance, DThread T1 sends a Multiple Update to DThread T2 in order to spawn all its instances, instead of sending 64 single Updates.

The three for-loop blocks are fully parallel and are mapped into three different DThreads. Each instance of a DThread is identified by the Context and it executes the inner command of the block. T2 has 64 instances (with Contexts from 0 to 63), T3 has 256 instances (with Contexts from 0,0 to 15,15) and T4 has 512 instances (with Contexts from 0,0,0 to 7,7,7). DThread T5 depends on all the instances of DThreads T2-T4, thus its RC is equal to 832. Moreover, the instances of DThreads T2-T4 have RC=1 because they have only one producer (the DThread T1). When T1 finishes its execution a Multiple Update will be sent in each consumer-thread. As a result, all the instances of DThreads T2-T4 will be executed concurrently.

3. The FREDDO Framework

FREDDO is a robust implementation of DDM that provides extra functionalities compared to the previous software DDM systems, like, support for recursion, larger Context values and better programming interface. C++11 was selected as the programming language in order to take advantage of object-oriented techniques such as maintainability, re-usability, data abstraction and encapsulation. Also, new features of C++11 are used such as range-based loops, initializer lists, Lambda expressions and atomic operations. FREDDO allows efficient DDM execution on multi-core and many-core systems by utilizing three different components: the TSU, the Kernels and the Runtime Support.

3.1 New features and Improvements

3.1.1 Extending the Context Attribute

The Context attribute of DDM was encoding up to three nesting levels in a 32-bit long word. The coding and decoding of Context values was cumbersome and it was difficult to have nested loops with large indexes. FREDDO supports three different Context sizes: 32-bit, 64-bit and 96-bit. In this work we are using 96-bit Context values for our evaluation.

3.1.2 Introducing dynamically allocated data-structures in TSU

The data-structures of the previous TSU modules were implemented using static memory allocation in order to increase the performance. This forces the programmer to recompile the TSU code when:

- The number of cores that will be utilized by the TSU has to be changed.
- The queues (called Input Queues) that hold the Update commands are full.
- The Synchronization Memory (SM) is full. SM is responsible for holding the RC values of the DThreads.

In order to address the aforementioned limitations we have implemented the majority of the TSU's data-structures by using dynamic memory allocation. We have also introduced an auxiliary data-structure, the Unlimited Input Queue (UIQ), for holding the Updates in case the Input Queues are full.

3.1.3 Reducing the memory allocated by the SM module

In the previous DDM implementations, an RC value is allocated for each instance of each DThread. As a result, SM was holding thousands or even millions of RC values. We observed that it is not necessary to allocate RC values for DThreads that have RC value equal to one. The instances of such DThreads can be scheduled immediately for execution when Update operations are received for them. This approach reduces the memory usage of DDM applications significantly as well as it accelerates the Update operations.

As an example, consider a for-loop with one million iterations that is able to be parallelized with a DThread that its RC value is equal to one. In this case our framework will avoid allocating one million RC values, i.e. 4MB of data (notice that an RC is a 32-bit integer value). Furthermore, this approach is vital for hardware DDM implementations [15], [16] where the size of the SM is limited.

3.1.4 Improving the DDM programmability

a) Object-Oriented Approach: In the previous DDM implementations the programs were developed using C macros [8], [9]. This approach was very restrictive and complicated since special macros had to be used for: (i) marking the boundaries of the DThreads, (ii) creating the DThreads' IFPs and (iii) extracting the index of the loops from Context values, regarding the Nesting value. Notice that IFP (Instruction Frame Pointer) is a pointer to the address of the DThread's first instruction. In this work, these functionalities are provided automatically.

FREDDO provides a C++ API (Application Programming Interface) that enables the programmers to develop DDM applications. The API includes a set of runtime functions and classes which are grouped together in a C++ namespace called *ddm*. The user is able to create and manage DThreads by just creating and accessing objects of special C++ classes. The development process is more efficient which reduces the programming effort. In this paper we present only a small subset of the API's functionalities due to the lack of space. An interested reader can find a full documentation of FREDDO as well as programming examples in [18].

We are providing four basic C++ classes for creating, updating and removing DThreads: *SimpleDThread*, *Multi-*

pleDThread, *MultipleDThread2D* and *MultipleDThread3D*. These classes correspond to DThreads with Nesting-0, Nesting-1, Nesting-2 and Nesting-3 respectively. Furthermore, these classes require the RC value to be specified by the programmer in their constructors.

b) Introducing DFunctions: In the previous software DDM systems the code of all DThreads had to be placed in the same function/place. This is because the runtime support of these systems uses *label* and *goto* statements for executing the code of the DThreads. Thus, the programmers were restricted from having parallel code in different files as well as in different functions.

In FREDDO, the code of the DThreads can be embodied in any callable target (called *DFunction*) like: (i) standard C++ functions, (ii) Lambda expressions and (iii) functors. This methodology allows us to have parallel code anywhere in a DDM program.

Each DFunction has one input argument, the Context value. Different Context structures (Context, Context2D and Context3D) are provided based on the type of the DThread class. Notice that the DFunctions of the DThreads with Nesting-0 do not have input arguments since the Context value is always zero.

3.1.5 Supporting Recursion

The envelopment of DThreads' code in DFunctions enables us to create parallel DThreads for function calls. This is useful for supporting recursion in the DDM model. FREDDO is the first DDM implementation that supports recursion. Our approach for providing recursion support in DDM was firstly presented in [19]. FREDDO provides three different classes for recursion support:

- ***RecursiveDThreadWithContinuation***: a special template class which provides functionalities for algorithms with multiple recursion. It is used when the number of instances of a recursive function is known at compile-time.
- ***RecursiveDThread***: a special class that allows parallelizing recursive functions that their number of instances is not known at compile time. This class allocates/deallocates the arguments and the return values of the instances at runtime. It can be used for different types of recursion, such as, linear, tail, and so on.
- ***ContinuationDThread***: it can be used in combination with RecursiveDThread to implement algorithms with multiple recursion (or any similar algorithms).

3.1.6 Automatic computation of the RC values

In the previous DDM systems the RC value of each DThread was required to be specified by the programmer. In this work we allow two different approaches for specifying the RC values of the DThreads:

1) Like the previous DDM systems, the RC values are given by the programmers. For this purpose the basic DThread classes have to be used.
2) The RC values will be computed at runtime based on the producer-consumer relationships of the DThreads. For supporting this new feature we have introduced a special data-structure in the TSU, the Pending Template Memory (PTM).

For the latter approach we are providing the FutureDThread Classes (FutureSimpleDThread, FutureMultipleDThread, FutureMultipleDThread2D and FutureMultipleDThread3D). These classes are derived-classes of the basic DThread classes which do not require an RC value to be specified in their constructors. Initially, the Thread Templates of the future DThreads will be stored in PTM and their RC values will be computed at runtime by the TSU module.

3.2 The FREDDO Architecture

Our implementation allows data-driven execution on top of any commodity OS. This allows the execution of DDM and non-DDM applications simultaneously. This key-feature is supported by the Kernels and the Runtime system. The overall architecture of the FREDDO framework is depicted in Figure 2.

3.2.1 The Thread Scheduling Unit (TSU)

a) The TSU's storage units: The TSU uses four main units for the storage, the Template Memory (TM), the Pending Template Memory (PTM), the Graph Memory (GM) and the Synchronization Memory (SM). The TM contains the Thread Template of each DThread. The PTM contains the Thread Templates that their RC values will be computed at runtime using the consumers of each DThread. The GM contains the consumers of each DThread. The SM contains the RC values of the different instances of the DThreads. A DThread that implements a loop has multiple instances, one for each iteration.

FREDDO supports static and dynamic SM. Static SM is used when the number of a DThread's instances is known at compile time (accessing an RC entry is a direct operation). Dynamic SM is used when there is no information about the number of instances of a DThread (accessing an RC entry is an associative operation). A different SM is allocated for each DThread in order to use exactly the amount of RC values that is required for each DThread.

b) TSU-Kernels Communication: The communication between the TSU and the Kernels is done through the Output Queues (OQs), the Input Queues (IQs) and the Unlimited Input Queues (UIQs). A triplet of IQ, UIQ and OQ is attached in each Kernel. The TSU dispatches the ready DThreads to the Kernels, through the OQs.

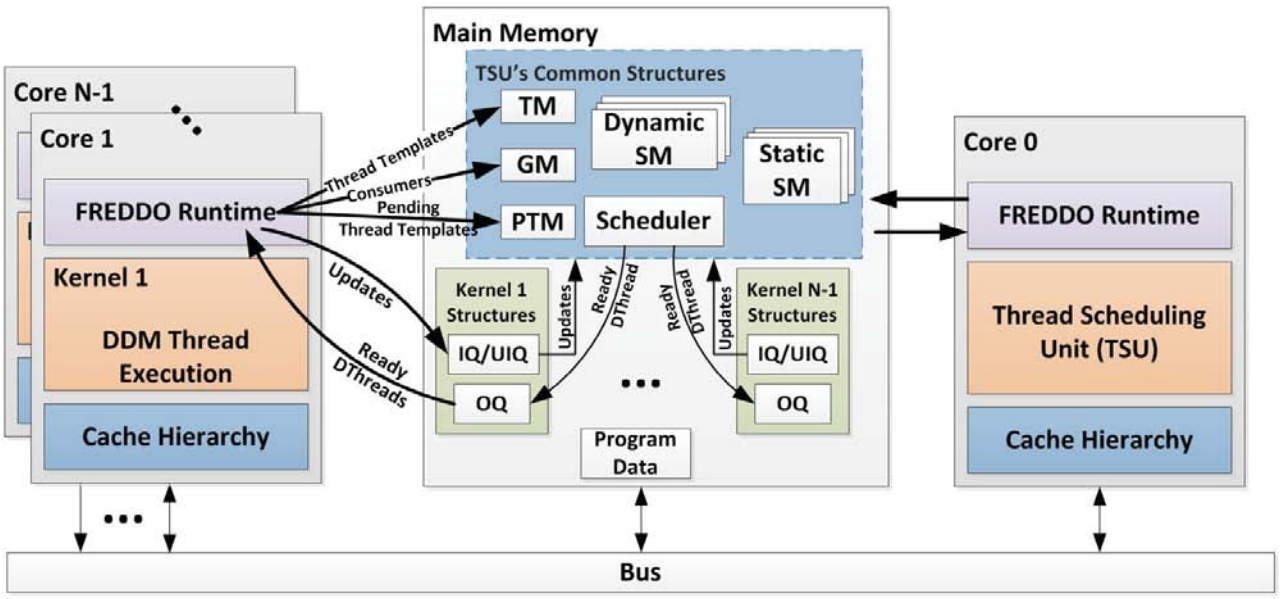


Fig. 2: The FREDDO's overall architecture.

After a Kernel completes the execution of a DThread, it sends Update commands to the consumers of the completed DThread. An Update consists of the Thread ID (TID) and the Context of the DThread that is going to be updated. Particularly, an Update operation indicates that the RC value that corresponds to the TID and Context attributes will be decreased by one. The Updates are stored in the IQs. If an IQ is full, then the Updates are stored in the associated UIQ. The UIQ is an efficient variable-length queue.

c) The TSU's Updating and Scheduling operations: The TSU fetches the Updates from the IQs/UIQs in a round-robin fashion. For each Update, the TSU locates the Thread Template of the DThread from the TM and decrements the RC value in the SM which is associated with the DThread (Static or Dynamic). If the RC value of any DThread's instance reaches zero, then it is deemed executable and it is sent to the Scheduler. An instance of a DThread that is ready for execution is called Ready DThread and consists of the TID, the IFP and the Context attributes.

The Scheduler is responsible for assigning the ready DThreads to the Output Queues (OQs). The DThread invocations are distributed to the Kernels in order to achieve load-balancing.

Prior the DDM scheduling, the TSU fetches the Pending Thread Templates (PTTs) from PTM and it computes their RC values based on the producer-consumer relationships of the DThreads. More specifically, the TSU performs three algorithmic steps:

- **Step 1:** For each PTT allocate an RC Value and set it to 0
- **Step 2:** Get the consumers of each DThread from GM. For each consumer of each DThread:
  - If the consumer is located in PTM, increase its RC Value by 1
  - If the consumer is not located in PTM, ignore it
- **Step 3:** For each PTT of PTM
  1) If the PTT's RC Value=0, set it to 1
  2) Remove the PTT from PTM and store it in TM
  3) If the PTT's RC Value $>$ 1, allocate a new SM (static or dynamic)

3.2.2 The Kernels

A Kernel is a POSIX Thread (PThread) that is pinned in a specific core until the end of the DDM execution. This eliminates the overheads of the context-switching between the Kernels in the system. The Kernel is responsible for executing the ready DThreads that are stored in its Output Queue (OQ). In FREDDO, *m* Kernels are created, where *m* is the maximum number of DThreads that can be executed in parallel in a system. Usually, *m* is equal to $N-1$, where *N* is the number of cores of the system. This is because one of the cores is reserved for the execution of the TSU code.

3.2.3 The Runtime System

The Runtime system enables the communication between the Kernels and the TSU through the Main Memory. It enqueues the Update commands in the IQ/UIQ pairs and it dequeues the ready DThreads from the OQs and forwards them to the Kernels. The Runtime is also responsible for loading the Thread Templates, the Pending Thread Templates and the Consumers of the DThreads, for creating and running the Kernels, and for deallocating the resources allocated by DDM programs.

4. Programming Methodology

In this section we show how the program of Figure 1 can be implemented in DDM using FREDDO. Listing 1 depicts one possible implementation of the algorithm. In this example we show a combination of basic and future DThread classes.

```
1  #include <dthreads.h>
2  #include <future_dthreads.h>
3  using namespace ddm; // Use the ddm name-space
4
5  // Declate DThread objects
6  FutureSimpleDThread *t1;
7  FutureMultipleDThread *t2; MultipleDThread2D *t3;
8  FutureMultipleDThread3D *t4;  SimpleDThread *t5
9
10 // Declare Global Variables (Arrays, etc.)
11
12 void t1_code(){ // Initializing Arrays ...
13  // Update the instances of consumers
14  t2->update(0, 63); // Multiple Update
15  t3->update({0,0}, {15,15}); // Multiple Update
16  t4->update({0,0,0}, {7,7,7}); // Multiple Update
17 }
18
19 void t4_code(Context3D c){
20  auto x = c.Outer, y = c.Middle, z = c.Inner;
21  D[x][y][z] = E[x][y][z] * F[x][y][z];
22  t4->updateAllCons();
23 }
24
25 void main(){ // Initializiations goes here ...
26  ddm::init(NUM_KERNELS);
27  // DTheads declarations using standard functions
28  t1 = new FutureSimpleDThread(t1_code);
29  t4 = new FutureMultipleDThread3D(t4_code);
30
31  // DTheads declarations using Lambda expressions
32  t2 = new FutureMultipleDThread([&](Context cntx){
33   C[cntx] = A[cntx] + B[cntx];
34   t5->update();
35  });
36
37  t3 = new MultipleDThread2D([&](Context2D cntx){
38   auto j = cntx.Outer, k = cntx.Inner;
39   R[j][k] = L[j][k] * M[j][k];
40   t5->update();
41  }, 1); // 1 at this point is the RC value
42
43  t5 = new SimpleDThread ([&](){
44   // Print Results ...
45  }, 832); // 832 at this point is the RC value
46
47  // Set the consumers of each DThread
48  t1->setConsumers({t2, t3, t4});
49  t2->setConsumers({t5});
50  t3->setConsumers({t5});
51  t4->setConsumers({t5});
52
53  t1->update(); // Decrease the RC of T1
54  ddm::run(); // Start the DDM scheduling
55  delete t1; ...; delete t5;
56  ddm::finalize();  // Deallocate Resources
57 }
```

Listing 1: DDM code of the program presented in Figure 1.

Additionally, we place the DThreads' code in standard C++ functions (for T1 and T4) and in Lambda expressions (for T2, T3 and T5) in order to show the ability of our framework to allow the DThreads' code to be placed in different callable targets.

The *init* runtime function (line 26) initializes the DDM execution environment and it starts NUM_KERNELS Kernels. For each DThread object we specify at least its DFunction. For instance, in line 29, we specify only the DFunction for DThread T4. Since we don't specify the maximum indexes of the loops the Dynamic SM will be used. Notice that FREDDO provides additional constructors that allow the users to use Static SMs [18].

Each DThread object can call several different types of Update commands. An object is able to Update itself or all its consumers. In both cases a user is able to specify a Context or a range of Contexts if the DThread which is going to be updated has Nesting>0. In the case of Nesting=0, there is no need to specify the Context since it's always zero. For example, in line 40, T3 sends an Update to T5.

A Multiple Update decrements a range of RC values of a DThread's instances. For example, in line 14, T1 updates the RC values of T2's instances from Context=0 to 63. In the T4's code (line 22), T4 updates all its consumers. In this example, T4 has only one consumer, the DThread T5.

After the DThreads' declarations, the consumers of each DThread are defined, using the *setConsumers* function (lines 48-51). The *run* function is responsible for computing the RC values of the pending Thread Templates (Future DThreads) and for starting the DDM scheduling.

When the scheduling finishes, the DThreads are removed from the TSU (line 55) using the *delete* operator of C++. Finally, all the resources allocated by the FREDDO framework are deallocated by executing the *finalize* runtime function (line 56).

5. Experimental Results

5.1 Experimental Setup

To evaluate FREDDO we have used a 32-core (with Clustered Multi-Threading) HP server machine equipped with 2 x 1.4GHz AMD 8-core Opteron 6276 processors and 48GB of DDR3 RAM clocked at 1333MHz. Out of the 32 cores, one is used to run the TSU, while the rest are used for executing DThreads. Our benchmark suite has ten applications. Figure 3 illustrates the characteristics of all the benchmarks. The problem sizes are separated into three categories: *Small, Medium and Large*. The last three columns depict the sequential execution time (in seconds) of each problem size of all benchmarks. The execution time measurements were collected using the *gettimeofday* system call. For the performance evaluation, all the experimental results are reported as speedups. Speedup represents how many times a certain parallel execution is faster than the corresponding sequential execution. The baseline for the speedup is the original sequential one, without any FREDDO overheads.

Benchmark	Description	Granularity	Problem Sizes			Sequential Execution Time (sec)		
			Small	Medium	Large	T_{Small}	T_{Medium}	T_{Large}
BMMULT	Blocked Matrix Multiplication	32x32 block	1Kx1K	2Kx2K	4Kx4K	4.26	34.07	273.34
LU	Blocked LU Decomposition	32x32 block	1Kx1K	2Kx2K	4Kx4K	1.26	10.11	80.91
Cholesky	Blocked Cholesky Factorization	64x64 block	1Kx1K	2Kx2K	4Kx4K	0.28	2.23	17.94
Blackscholes	Financial analysis application	variable	16K options	64K options	10M options	2.11	8.55	1359.60
Conv2D	9x9 convolution filter	32x32 block	1Kx1K	2Kx2K	8Kx8K	1.47	5.88	93.72
Swaptions	Financial analysis application	102400 Simulations	256 Swaptions	512 Swaptions	4096 Swaptions	283.63	567.23	4537.16
Trapez	Trapezoidal rule for integration	variable	2^{29} steps	2^{30} steps	2^{31} steps	1.88	3.76	7.52
Fibonacci	Recursive implementation of the Fibonacci algorithm	variable	N=40	N=45	N=48	1.12	12.59	140.69
Knights-Tour	Recursive implementation of the Knight's tour problem	variable	4x4 board	5x5 board	6x6 board	0.01	10.49	202179.00
NQueens	Recursive implementation of the NQueens problem	variable	N=14	N=15	N=16	9.66	65.76	469.75

Fig. 3: The benchmark suite characteristics.

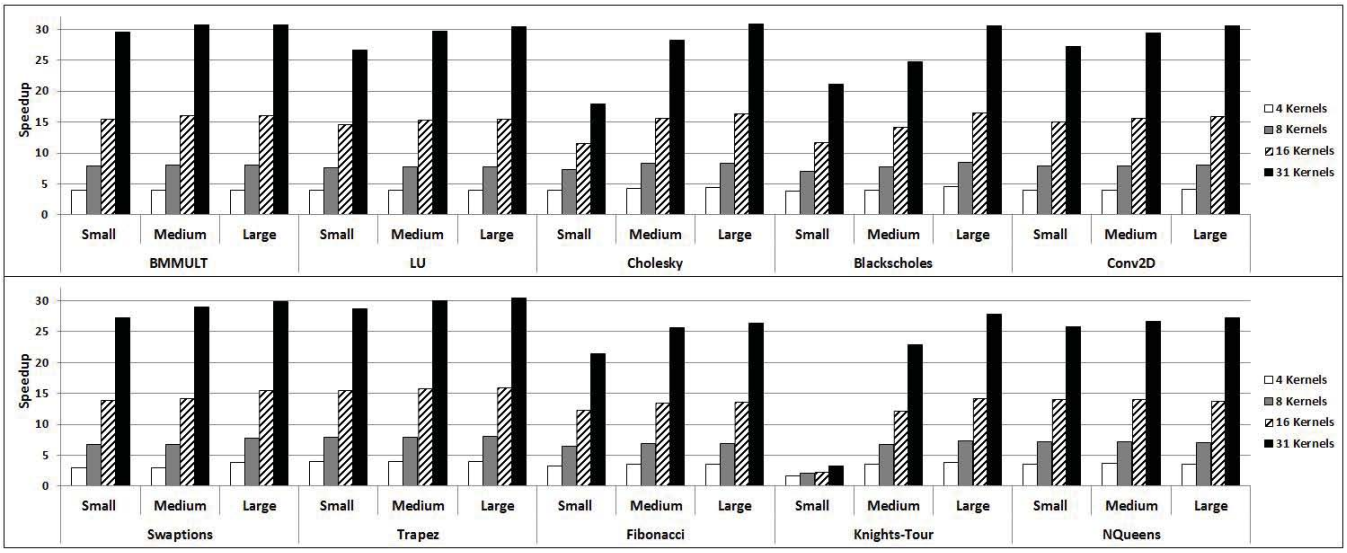


Fig. 4: Performance scalability of FREDDO for different number of computation cores (Kernels) and problem sizes.

5.2 Performance Evaluation

In this work we performed a scalability study of the performance for applications with different characteristics. Our benchmark suite includes applications that are embarrassingly parallel (like Swaptions and Blackscholes [20]), applications that are compute-bound like Trapez and applications that have a combination of memory-bound and compute-bound nature (like BMMULT). Additionally, benchmarks with complex data-dependencies are selected (like LU and Cholesky) as well as benchmarks with recursion. The benchmarks with recursion were implemented using the RecursiveDThreadWithContinuation class.

We have evaluated the performance of our framework on different number of cores and problem sizes. The evaluation is shown in Figure 4. Four different Kernel configurations are used: 4, 8, 16 and 31. The evaluation shows that FREDDO scales very well across the range of the benchmarks and it achieves very good speedups, especially in the *Large* problem size. This is justified by the fact that, as the benchmark's execution time increases, the parallelization overhead is amortized.

For the *Large* problem size, the blocked algorithms (BMMULT, Cholesky, Conv2D and LU), Swaptions, Blackscholes and Trapez achieved almost the theoretical peak in each configuration. For instance, the LU benchmark, achieves the following speedups: 3.95 out of 4, 7.84 out of 8, 15.57 out of 16 and 30.41 out of 31. The recursive algorithms ended up with lower speedups due to their complexity. Particularly, they achieved an average speedup of 3.62 out of 4, 7.08 out of 8, 13.84 out of 16 and 27.15 out of 31.

For the *Small* and *Medium* problem sizes, FREDDO achieves also very good performance, especially in the cases of the blocked and compute-bound algorithms. To conclude, the results achieved for all benchmarks show that FREDDO effectively leverages the decoupling of synchronization and execution for the maximum tolerance of synchronization overheads.

5.3 Comparisons with other frameworks

FREDDO is compared with three different frameworks, the DDM-VM [8], [9], the OpenMP (version 3.1) and the

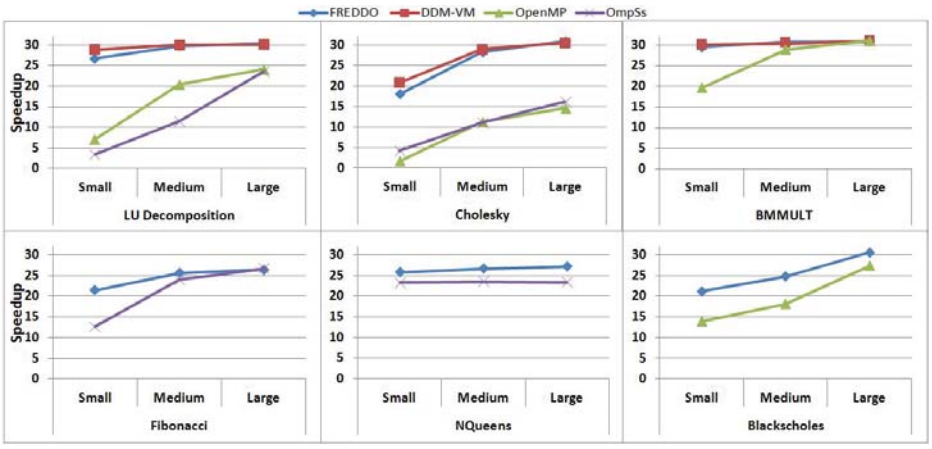


Fig. 5: Framework comparisons on benchmarks with different characteristics.

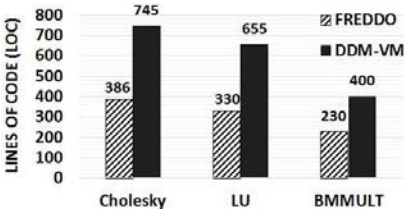


Fig. 6: FREDDO vs. DDM-VM on lines of code (LOC).

OmpSs (version 15.04 [11]), for benchmarks with different characteristics (Figure 5).

In the case of Cholesky and LU benchmarks which have strict data-dependencies, the DDM systems achieve better performance results compared to the OpenMP and OmpSs implementations. This is due to the DDM ability to tolerate synchronization latency by decoupling the synchronization from computations, and allowing the TSU to operate asynchronously from the computation cores.

Additionally, we observed that OpenMP and OmpSs need much larger sizes (for example 8Kx8K matrix sizes) to achieve similar results with the DDM systems. This is because of: (i) the low thread switching overheads of the DDM frameworks and (ii) the static dependency resolution that is used in our case which incurs less overheads. For the same reasons, FREDDO achieves better performance than OpenMP for the Blackscholes and BMMULT (for the *Small* and *Medium* problem sizes) benchmarks which are embarrassingly parallel.

Furthermore, we compare FREDDO with OmpSs for the Fibonacci and NQueens benchmarks which are recursive algorithms. In the case of the Fibonacci algorithm FREDDO is better than OmpSs for the *Small* problem size while they achieve similar results for the rest of the problem sizes. In the case of NQueens, our framework is about 10% faster than the OmpSs framework.

DDM-VM achieves slightly better results than FREDDO, for the *Small* problem size due to three factors:

- The DDM-VM uses only static memory in contrast with FREDDO which uses both static and dynamic memory.
- In DDM-VM, all DThreads are declared in the same function using *label* statements. The DDM-VM's runtime executes the ready DThreads by simply jumping to their *labels* using the *goto* statements. Our framework places the DThreads' code in any callable target (like functions and Lambda expressions). The FREDDO's runtime executes the ready DThreads by calling the callable targets which incurs more overheads than the DDM-VM's approach.
- In FREDDO, functionalities such as creating, updating and removing DThreads are provided by creating and accessing objects. DDM-VM provides the same functionalities by using C macros which incurs less overheads.

However, the factors above, restrict the programmers from writing productive DDM-VM applications (see Sections 3.1.2 and 3.1.4). For the larger problem sizes (*Medium* and *Large*), the overheads derived from the new features of our framework are amortized. Thus, FREDDO and DDM-VM achieve very similar results. It is worth noting that the lines of code (LOC) of the DDM benchmarks in FREDDO is reduced to half compared to DDM-VM (Figure 6).

6. Related Work

OmpSs [11], [12] is a programming model that provides the features of the StarSs [21] framework using OpenMP directives. OmpSs allows to express data-dependencies between tasks using the *in, out and inout* clauses. The Nanos++ runtime system is used to support task parallelism using synchronizations based on data-dependencies. Also, the Mercurium source-to-source compiler is used which recognizes constructs and transforms them to calls to the runtime. OmpSs builds the dependency graph at runtime (dynamic dependency resolution), thus, this approach incurs extra overheads. The DDM model provides support for both static and dynamic dependency resolutions [22].

Gupta and Sohi [6] provide a C++ runtime library that allows data-flow/data-driven execution of sequential imperative programs on multi-core systems. Their framework exploits Functional-Level Parallelism (FPL) by executing functions on the cores in a data-flow fashion. In contrast, DDM applies its techniques at non-blocking threads.

SWARM (SWift Adaptive Runtime Machine) [23] is a software runtime that uses an execution model based on

codelets [3]. SWARM divides a program into tasks, with runtime dependencies and constraints that can be executed when all runtime dependencies and constraints are met. The runtime schedules the tasks for execution based on resource availability. Also, SWARM utilizes a work-stealing approach for on-demand load-balancing.

Thread Building Blocks (TBB) is an API developed by Intel that relies on C++ templates to facilitate parallel programming [24]. It provides a set of data-structures and algorithmic skeletons that supports the execution of tasks. TBB also provides a set of concurrent containers (hash-maps, queues, etc.) and synchronization constructs (mutex constructs, atomic operations). The TBB runtime implements a tasks-stealing scheduling policy and adopts a fork-join approach for the creation and management of tasks. On the contrary, DDM relies on data-dependencies for the scheduling of threads.

7. Conclusions

In this work we presented FREDDO, an object-oriented software implementation for the Data-Driven Multithreading (DDM) model. The proposed framework targets multi/many-core systems and it provides improvements and new functionalities over the previous DDM implementations.

FREDDO performance has been evaluated on a 32-core server with ten benchmarks and compared with the performance achieved from OpenMP, OmpSs and DDM-VM frameworks. The evaluation shows that FREDDO scales well and tolerates scheduling overheads and memory latencies effectively. Our framework outperforms the OpenMP and OmpSs especially in the case of high-complexity applications. We also showed that FREDDO achieves similar results with DDM-VM despite the fact that the former provides more functionalities. The size of the DDM applications implemented in FREDDO is reduced to half, compared to the size of the DDM-VM applications, due to the improvements in the programming interface.

Future work will be focused on a distributed implementation of FREDDO. Also, we plan to support DDM execution in other object-oriented languages like Java and Scala.

Acknowledgment

This work was partially funded by the IKYK foundation through a scholarship for George Matheou.

References

[1] S. H. Fuller and L. I. Millett, "Computing performance: Game over or next level?" *Computer*, no. 1, pp. 31–38, 2011.

[2] Arvind and R. A. Iannucci, "Two fundamental issues in multiprocessing," in *4th International DFVLR Seminar on Foundations of Engineering Sciences on Parallel Computing in Science and Engineering*. New York, NY, USA: Springer-Verlag New York, Inc., 1988, pp. 61–88.

[3] S. Zuckerman, J. Suetterlein, R. Knauerhase, and G. R. Gao, "Position paper: Using a codelet program execution model for exascale machines," in *EXADAPT Workshop*. Citeseer, 2011.

[4] G. Matheou, P. Evripidou, and C. Kyriacou, "Paradigm shift for exascale computing," in *Proceedings of the 3rd International Conference on Exascale Applications and Software*. University of Edinburgh, 2015, pp. 109–114.

[5] P. Kogge, "Next-generation supercomputers," *IEEE Spectrum, February*, 2011.

[6] G. Gupta and G. S. Sohi, "Dataflow execution of sequential imperative programs on multicore architectures," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 59–70.

[7] C. Kyriacou, P. Evripidou, and P. Trancoso, "Data-driven multithreading using conventional microprocessors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 17, no. 10, pp. 1176–1188, Oct. 2006.

[8] S. Arandi and P. Evripidou, "Programming multi-core architectures using data-flow techniques," in *SAMOS-2010*. IEEE, 2010, pp. 152–161.

[9] ——, "DDM-VMc: the data-driven multithreading virtual machine for the cell processor," in *Proc. of the 6th Int. Conf. on High Performance and Embedded Architectures and Compilers*, 2011, pp. 25–34.

[10] O. Board, "Openmp application program interface version 3.0," in *The OpenMP Forum, Tech. Rep*, 2008.

[11] BSC, "The ompss programming model," 2015. [Online]. Available: https://pm.bsc.es/ompss

[12] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "Ompss: a proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011.

[13] K. Stavrou, M. Nikolaides, D. Pavlou, S. Arandi, P. Evripidou, and P. Trancoso, "TFlux: a portable platform for data-driven multithreading on commodity multicore systems," in *Parallel Processing, 2008. ICPP'08. 37th International Conference on*. IEEE, 2008, pp. 25–34.

[14] G. Michael, S. Arandi, and P. Evripidou, "Data-flow concurrency on distributed multi-core systems," in *Proceedings of the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13)*, 2013.

[15] G. Matheou and P. Evripidou, "Verilog-based simulation of hardware support for data-flow concurrency on multicore systems," in *SAMOS XIII, 2013*. IEEE, 2013, pp. 280–287.

[16] ——, "Architectural support for data-driven execution," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 52:1–52:25, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2686874

[17] Arvind and Gostelow, "The u-interpreter," *Computer*, vol. 15, no. 2, pp. 42–49, Feb. 1982.

[18] G. Matheou and P. Evripidou, "Freddo: an efficient framework for runtime execution of data-driven objects," Department of Computer Science, University of Cyprus, Nicosia, Cyprus, Tech. Rep. TR-16-1, January 2016. [Online]. Available: https://www.cs.ucy.ac.cy/docs/techreports/TR-16-1.pdf

[19] G. Matheou, I. Watson, and P. Evripidou, "Recursion support for the data-driven multithreading model," 2015, will be published in Fifth Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM 2015).

[20] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[21] J. Planas, R. M. Badia, E. Ayguadé, and J. Labarta, "Hierarchical task-based programming with starss," *International Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 284–299, 2009.

[22] S. Arandi, G. Michael, P. Evripidou, and C. Kyriacou, "Combining compile and run-time dependency resolution in data-driven multithreading," in *Data-Flow Execution Models for Extreme Scale Computing (DFM), 2011 First Workshop on*. IEEE, 2011, pp. 45–52.

[23] C. Lauderdale, M. Glines, J. Zhao, A. Spiotta, and R. Khan, "Swarm: A unified framework for parallel-for, task dataflow, and distributed graph traversal," *ET International Inc., Newark, USA*, 2013.

[24] J. Reinders, *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media, Inc., 2007.

ISBN: 1-60132-444-8, CSREA Press ©

SESSION

SAFE, SECURE AND DEPENDABLE INFORMATION SHARING NETWORK SYSTEMS AND SERVICES

Chair(s)

Prof. Hiroaki Nishikawa
Prof. Hiroshi Ishii

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

A Study on the Information Content Leaked from Queries to Search Engines and Its Reduction

Hiroshi Yamamoto[1] **and Yusuke Hiraide**[2]
[1]School of Information and Telecommunication Engineering, Tokai University,
2–3–23 Takanawa Minato-ku, Tokyo, Japan
[2]Hitachi, Ltd. 1–6–6 Marunouchi Chiyoda-ku, Tokyo, japan

Abstract— *Recently, the opportunity to use search engines has increased due to the increased variety and number of Internet-capable devices. Search engines have become indispensable services for many users. Users are sending a vast amount of information as input to search engines. The risk when information related to user privacy is stored and analyzed by search engine providers has recently been recognized. There is existing research regarding protecting user privacy from search engines that try to extract related information from users' query strings. The searchable encryption technique is effective when searching encrypted queries. This technique is useful when users search their own data stored in external cloud storage. However, search engine providers make a profit based on the query strings of their many users, therefore, they are not expected to adopt this approach. Private Information Retrieval (PIR) is a known technique that ensures no information is leaked to the search engine. However, PIR is a technique based on a model with strict limitations on retrieval and is hardly a practical method. In this paper, we define a measure for the amount of information that is leaked during a search activity. The distinctive feature of this paper is that the amount of information is defined using entropy. In addition, a practical search scheme that reduces the amount of leaked information is proposed. The proposed scheme is simple, and we implement the system using a typical personal computer (PC). We evaluate the system by experiment and confirm that the proposed scheme works correctly and is of acceptable usability.*

Keywords: search engine, leak of information, privacy

1. Introduction

Recently, the use of smartphones with the ability to connect to the Internet using a cellular or wireless network is increasing. Many users have become able to use the Internet anywhere. The web search is one of the most popular services on the Internet [1]. Users send search keywords to search engines every day to perform web searches. Keyword searches from a user may relate to the user's interests, lifestyle or job. There are obviously issues of privacy in sensitive keywords sent to search engines. Search engine providers actually store queries from users and analyze them, and they use the collected information to provide advertisements or recommendations. It is argued that there are privacy risks from the information stored by the search engines. It is reported that, from the user's web search queries, the user's gender can be calculated with 84% accuracy and the user's age can be calculated with an absolute error of 7 years [2].

A number of studies have been performed to avoid the risk caused by search engines. One study is on searchable cryptography technology [3]. Using a homomorphic encryption function, we send encrypted queries to the search engine and perform searches for the encrypted queries among the encrypted data in the search engine. Searchable cryptography ensures that the search engine cannot know the plain text version of the queries. Searchable cryptography technology is effective when we search for our own data stored in the cloud. It is not suitable for a general web search service because search engine providers make a profit from the content of query strings, using this information for advertisements or recommendations.

PIR (Private Information Retrieval)[4], [5] is another technology to protect query privacy. It has been shown that we can send search data to a search engine using PIR while giving no information to the search engine. We explain the details in section 2. However, PIR is based on a mathematical model, and the constraints and assumptions are too strict for practical use.

In this paper, we define a measure for the amount of information that is leaked during a search activity. We propose a practical scheme in which we protect a certain amount of privacy while allowing a small amount of information to leak. We evaluate the proposed scheme in terms of the measure of leaked information and from a practical viewpoint.

2. Existing Technique

Because search engine providers make a profit from analyzing search keywords, we have to make the following assumption.

Assumption 1: Search engines store the plain text version of the query strings.

This is why we cannot expect search engine providers to adopt an approach using searchable cryptography.

ISBN: 1-60132-444-8, CSREA Press ©

2.1 PIR

PIR is a secure search scheme using plain text. In PIR, the search is defined as follows;

- The number of searchable items is n.
- Each $A_i (1 \leq i \leq n)$ is a binary digit.
- The user wants to know the value of $A_k (1 \leq k \leq n)$ for a particular k.

In this model, an array $(A_1, A_2, \ldots, A_n)$ represents a database and is stored in a search engine. An integer k represents a search keyword and the value of A_k represents the search result.

In this model, an ordinary search action corresponds to the following procedure;

1) The user sends the value of k to the search engine.
2) The search engine retrieves the value of A_k and sends it to the user.

The information we want to keep secret in this model is the index k. Obviously, that same index k is known by the search engine in this procedure. We assume that all the information about the search keyword is leaked to the search engine in this procedure. The goal of PIR is to get the value of A_k without giving any information about the value of k to the search engine. PIR provides an information-theoretically secure way to retrieve the value of A_k. A trivial form of PIR is defined in the following procedure;

1) The user requests that the search engine send *all* the data $A_1, A_2, \ldots A_n$.
2) The search engine sends the user the requested data.
3) The user retrieves A_k and abandons the other values.

The procedure gives no information about the value of k to the search engine because the search engine can simulate the input of the user independently. Therefore, this trivial PIR is an information-theoretically secure scheme. On the other hand, the search engine must send the user all the data $A_1, A_2, \ldots, A_n$. This procedure is not a practical scheme because of the unrealistically large amount of communication traffic it requires. It has been demonstrated that if we use a single database, trivial PIR is the only way to achieve an information-theoretically secure scheme.

2.2 Feasibility of PIR

Schemes to reduce communication complexity by using multiple servers have been proposed [5]. These schemes make the following assumptions:

- The mirror servers have exactly the same contents.
- The mirror servers are not conspiring.

Implementing the first assumption strictly is difficult due to the technical problems of time lag or failure. Mirror servers cooperate with one another to retrieve the searched data. It is difficult for users to believe that mirror servers are not conspiring.

PIR is a scheme that guarantees information-theoretic security, but there are difficulties in its practical implementation.

3. Measure of Leaked Information

From the discussion in the previous section, the practical implementation of a information-theoretically secure web search is infeasible. The goal of this research is not perfect privacy but to protect a certain amount of privacy, tolerating a small amount of information leakage to the search engines. Quantification of the amount of information that is leaked to search engines is crucial in this approach. We define the amount of leaked information using the entropy of the query strings.

3.1 Terminology

In this paper we use the following terms;

Input alphabet:

The set of characters that a user can input into the search box of a search engine.

- Note that the space character that separates keywords is also an element of the input alphabet.
- In many case, the input alphabet consists of alphabetic characters, numeric characters and the space character.
- We denote the input alphabet by Σ.

Search string:

A string using characters from the input alphabet Σ.

- A search string corresponds to the whole string that a user inputs into a search box.
- A search string may represent multiple search keywords separated by space characters.

Search string space:

The set of all the possible search strings.

- We denote the search string space by S.

Real search string:

The search string that the user actually wants to input.

- The real search string s is an element of S.

Search action:

A series of search strings containing the real search string.

- In order to reduce the quantity of leaked information, a user may execute searches including the real search string as well as dummy search strings.
- A search action is an array σ that consists of elements in S containing the real search string.

For example, consider the case of a user executing a search for the two keywords "apple" and "orange" in the search box of a search engine that accepts twenty letters of alphabetic, numeric or space characters. We assume that alphabetic characters are represented internally as lowercase characters in the search engines. In this case, the input alphabet Σ is the set $\{a, \ldots, z, 0, \ldots, 9, [\text{SPACE}]\}$. A search string is a string from Σ with length 20. We assume that if

an input string is shorter than 20 characters, [SPACE]s are used as padding in the rest of the search string. The search string space S is the twentieth extension of the input alphabet Σ^{20}, the set of possible search strings. The real search string is "apple+orange++++++++", where '+' stands for a [SPACE] character. The search action is a series of search string that are a mixture of dummy search strings and the real search string, as in the following example;
$\sigma = s_1, s_2, s_3$, where
s_1 = "peach+grape+++++++++",
s_2 = "apple+orange++++++++",
s_3 = "banana+apple+apricot".

3.2 Quantity of the Leaked Information

We define the quantity of the information leaked to a search engine as follows.

Definition 1:

$$\begin{bmatrix}\text{The amount of information leaked}\\ \text{from a search action}\end{bmatrix} = \begin{bmatrix}\text{Ambiguity of the real search string}\\ \textit{before}\text{ the search action}\end{bmatrix} - \begin{bmatrix}\text{Ambiguity of the real search string}\\ \textit{after}\text{ the search action}\end{bmatrix} \quad (1)$$

In this paper we use Shannon entropy to quantify the ambiguity of the search strings.

Shannon entropy

- $S = \{s_1, s_2, \ldots, s_q\}$: The information source
- $P_i \ \ (1 \le i \le q)$: The probability that s_i occurs
- $H(S)$: The (binary) Shannon entropy of S, where

$$H(S) = -\sum_{i=1}^{q} P_i \log_2 P_i. \quad (2)$$

In this paper we may omit the base number for the base 2 logarithm.

Using Shannon entropy, the ambiguity of the real search string before the search action is $H(S)$ and the ambiguity of the real search string after a search action σ is the conditional entropy $H(S|\sigma)$. From definition 1, the amount of information that is leaked to the search engines is

$$H(S) - H(S|\sigma). \quad (3)$$

3.3 Models and Leaked Information Content

We now discuss several cases in which various dummy search strings are assumed to be used.

Case 1: Ordinary search

We assume that the input alphabet Σ is the set $\{a, \ldots, z, 0, \ldots, 9, [\text{SPACE}]\}$. The cardinality of the input alphabet $|\Sigma|$ equals 37. Also, we assume that the length of the search box is 100. Thus the search string space S is Σ^{100}. For simplicity, we assume that each possible search string $s_i \in S$ occurs with equally likely probability. In this case, each P_i equals $(1/37)^{100}$ for $i = 1, 2, \ldots, 37^{100}$ and the entropy $H(S)$ equals $100 \log 37$

In an ordinary search, the search action σ consists of the real search string only. After the search engine receives σ, the search engine can determine the real search string with the probability of 1. This means that the ambiguity of the real search string after the search action $H(S|\sigma)$ equals zero.

We conclude that the leaked information $H(S) - H(S|\sigma)$ equals $H(S)$. This is the case where all the information is leaked to the search engine.

Case 2: Search all the possible strings

Assume that Σ, S and P_i are the same as those in **Case 1**. In this case we assume that the search action σ consists of all the possible search strings. Because the search engine has zero knowledge about the real search string, the ambiguity of the real search string after the search action is the same as the ambiguity of the real search string before the search action. This means that $H(S|\sigma)$ equals $H(S)$, and therefore the leaked information $H(S) - H(S|\sigma)$ is zero. This case corresponds to the PIR scheme and is an information-theoretically secure search action.

Case 3: Search n strings

This case is one of the more practical cases. We do not use all the possible search strings but instead n search strings as σ. Again, Σ, S and P_i are assumed to be the same as those in **Case 1**.

We assume that σ consists of n distinct search strings containing the real search string. In other words, σ is a series of $n-1$ distinct dummy strings and the real string in arbitrary order. In this case, because one of the n search strings is the real search string, the search engine gets n candidates for the real search string and the probability that each candidates is the real search string equalls $1/n$. This means that the ambiguity after the search engine receives the search action σ is

$$H(S|\sigma) = \sum_{1}^{n} \frac{1}{n} \log n = \log n. \quad (4)$$

We conclude that the leaked information $H(S) - H(S|\sigma)$ equals $100 \log 37 - \log n$.

If n equals $\frac{1}{2^i}$ of all the possible search strings, the leaked information equals

$$\log 37^{100} - \log \frac{37^{100}}{2^i} = i.$$

If one half of all the possible search string are searched, the leaked information to the search engine is 1 bit.

Case 4: Word unit model

In this case, we consider another model in which the unit is a word instead of a character. In this model, we regard each English word that represents a single search keyword as a literal. Let $\mathcal{E}$ be the set of English words. A search string is a subset of $\mathcal{E}$. If there is no limit on the length of the input search string, the search string space S is $2^{\mathcal{E}}$,

the power set of $\mathcal{E}$. Let w be the number of English words ($|\mathcal{E}| = w$).

We now consider the number of search keywords in a single query. In the real world, large numbers of search keywords are seldom used at once. We expect that the distribution of the number of search keywords has a certain statistical tendency. In this case, let $P(K = i)$ be the probability that i keywords are used in a single query. Consider the case where the number of keyword is fixed at i. The number of subsets of $2^{\mathcal{E}}$ with i keywords is $\begin{pmatrix} w \\ i \end{pmatrix}$. We assume these subsets occur with equal probability. Therefore the probability of a particular search string with i keywords is

$$\frac{P(K=i)}{\begin{pmatrix} w \\ i \end{pmatrix}} \tag{5}$$

From (2), the entropy of the real search string is

$$H(S) = \sum_{i=1}^{w} \sum_{i\text{words query}} \frac{P(K=i)}{\begin{pmatrix} w \\ i \end{pmatrix}} \log \frac{\begin{pmatrix} w \\ i \end{pmatrix}}{P(K=i)}$$

$$= \sum_{i=1}^{w} P(K=i) \log \frac{\begin{pmatrix} w \\ i \end{pmatrix}}{P(K=i)} \tag{6}$$

In this case, the amount of leaked information is evaluated using the distribution of $P(K = i)$. The leaked information is as follows:

- If we execute an ordinary search, that is, σ consists of only the real search string, the leaked information equals
$$\sum_{i=1}^{w} P(K=i) \log \frac{\begin{pmatrix} w \\ i \end{pmatrix}}{P(K=i)}.$$
- If we execute all the searches in $2^{\mathcal{E}}$ as σ, the leaked information equals zero.
- If we execute n searches including $n-1$ dummy search strings as σ, the leaked information equals
$$\sum_{i=1}^{w} P(K=i) \log \frac{\begin{pmatrix} w \\ i \end{pmatrix}}{P(K=i)} - \log n.$$

4. Proposed Method

In this section we propose practical schemes to reduce the information content that is leaked to the search engines.

4.1 Design Over View

We begin by explaining the proposed system briefly. The main components of the proposed system and interactions between these components are shown in Figure 1 and Figure 2.

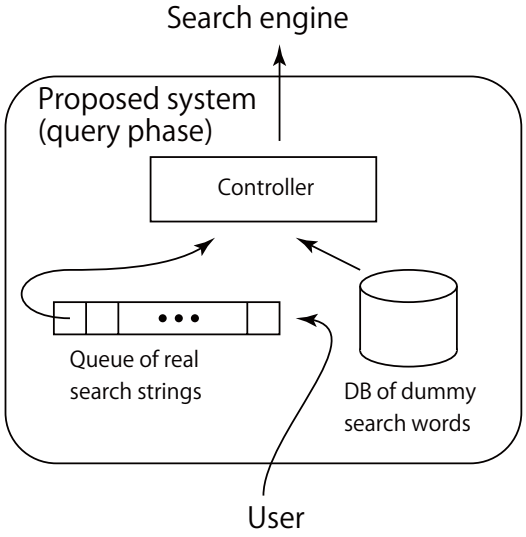


Fig. 1: Query phase of the proposed system.

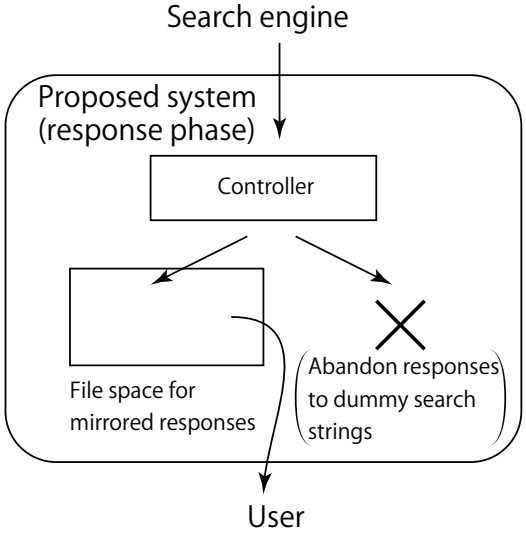


Fig. 2: Response phase of the proposed system.

The proposed system primarily consists of the "Queue of real search string", "Database of dummy search words", "File space for responses" and "Controller". The proposed system is placed in a trusted organization and acts generally as a web proxy server. When the user requests a search with a real search string, it is enqueued into the real search string queue. The controller checks for items in the queue at fixed intervals. If there is an item at the head of the queue, the controller executes a search for the item otherwise the controller choose a dummy search string constructed from the database and searches for this instead.

The controller stores the responses to queries for real search strings into the file space. Responses to dummy queries are simply abandoned. The system periodically

checks if there is a response in the file space after the user has sent the query to the proposed system.

4.2 Detailed Design

The proposed system consists of two modules: the user query module and the server query module. Both are installed on a server that is trusted by the user. The two modules are executed by performing the following procedures repeatedly;

User query module

1) Listen for the transmission of a real search string from the user.
2) When a real search string arrives, enqueue it into the queue of real search strings.
3) Check the file space periodically for the mirrored response, until the contents created from the response to the search string appear.
4) Return the response to the user using the contents in the file space.

Server query module

1) The controller checks the queue of real search strings for items at fixed intervals.
2) If there is an item in the queue of real search strings, the controller dequeues the item on the head of the queue and sends the search engine the dequeued item. If the queue is empty, the controller makes a dummy search string from the database of dummy search words and sends the dummy search string to the search engine.
3) If the query sent in the previous step was a search string from the queue (a real search string), the controller stores the mirror contents of the response to the search string. If the query was a dummy search string, the controller simply ignores the response.

Information from the user query module to the server query module is transmitted by the queue of real search strings. The queue must store additional information to identify the particular query. Information from the server query module to the user query module is transmitted by the file in the file space for mirrored responses. The server query module must store files according to identifiable rules.

4.3 Multiuser Expansion

The proposed scheme is easily expanded to accept multiple users. The expansion is enabled by allowing the queue of real search strings in Figure 1 to accept search strings from multiple users and adding adequate information to identify distinct users and queries. This expansion is effective in practice because real search strings can be used as dummy search strings for other users. Therefore, the traffic caused by dummy search strings is reduced when one system is shared by many users.

5. Experiment

We implemented the proposed system including the multiuser expansion in Section 4.3. The environment of the experiment is shown in Table 1.

Table 1: Experimental environment

CPU	Intel Xeon X3430(8M Cache, 2.40GHz)
Operating system	Ubuntu Linux 14.04
Development language	PHP5.5.9
Tools	Apache2.4.7, MySQL 14.14

The server query module checks the queue for items at fixed intervals (Section 4.2). The interval is vitally important because if it is set for too long a period, the time lag to receive the response may become unacceptable. On the other hand, if the interval is set to be too short, heavy traffic caused by the searches of dummy search strings may become a problem. In some cases, the search engine may regard such traffic as a kind of DOS attack and may cut off system access. In this experiment, the interval is set to 5 seconds, considering practical usage levels.

Experiments were performed using 10 people, and we confirmed that the system works correctly. There were 31 searches in total, and the time lag of the response ranged from 14 to 57 seconds.

6. Conclusions

There are privacy risks when using search engines. It is possible to predict the users' attributes such as sex, age, preference, interests, and so on from the users' queries. Searchable cryptography using homomorphic functions may not solve the problem. Search engine providers make a profit from analyzing search keywords, therefore they will maintain the search keywords in plain text.

In this paper we proposed a measure for the amount of information content that leaked from queries to search engines, using entropy. We defined the amount of information leaked by a search action as the ambiguity of the real search string before the search action minus the ambiguity of the real search string after the search action. We analyzed this measure of the leaked information in several cases. Then we demonstrated a practical scheme to reduce the leaked information, consisting of a queue of real search strings, a database of dummy search words, a file space for mirrored responses and a controller. The proposed scheme is especially effective in a multiuser implementation. We implemented the proposed scheme with inexpensive equipment. We confirmed that the system works correctly and the time lag is acceptable.

Using the measure proposed in this paper, we can quantify the information leaked to search engines during a web search, and we can reduce the leaked information using the proposed scheme.

We will continue experiments and analyses with more participants. We will refine the source code of the system to reduce the time lag of the response. There is still a gap between the model used in the evaluation of the leaked information and the implemented system. Narrowing the gap by refining the model of leaked information is an issue to be addressed in the future.

References

[1] Sören Preibush "The value of privacy in Web search," WEIS 2013
[2] Rosie Jones, et al., "I know what you did last summer- query logs and user privacy," CIKM 2007
[3] Craig Gentry, "A Fully Homomorphic Encryption Scheme," STOC 2009
[4] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private Information Retrieval," In Proceedings of the 36th Annual Foundations of Computer Science. IEEE Computer Society Press, 1995
[5] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private Information Retrieval," Journal of the ACM, Vol. 45, No. 6, November 1998

A Study on Approximation of the Processing Time of a Model of Cloud Computing

Hiroshi Yamamoto[1], **Yutaro Kuriyama**[2] **and Hiroshi Ishii**[1]
[1]School of Information and Telecommunication Engineering, Tokai University,
2–3–23 Takanawa Minato-ku, Tokyo, Japan
[2]DOCOMO Systems, Inc. Kokusai Akasaka Bldg. 5F 2–4–5 Akasaka Minato-ku, Tokyo, japan

Abstract— *Recently, cloud services have been used not only as storage but also for computing power. Cloud computing systems perform computations using distributed computation with a large number of computers connected to the Internet. Therefore, cloud computing systems are designed to compute correct results even if failures occur on some of the computers. Many cloud computing systems have mechanisms for redundant calculation for this purpose, which make it difficult to estimate the computing power of cloud services. In this paper, we propose a method to estimate the computing power of cloud computing systems. First, we define a simple model of cloud computation. Assumptions about hardware and software control of the model are shown. We introduce a concept called the "round number". We show that the round number is crucial for the overall computation time, and we derive a formula for the distribution function of the round number. We show that the overall computation time of cloud computing systems can be approximated using the distribution function of the round number combined with an approximation of the frequency of the computation time for each round number. Simulation of distributed computation is performed for an experimental problem and the validity of the proposed method is confirmed.*

Keywords: cloud computing, discrete probability distribution

1. Introduction

Cloud computing is defined as the kinds of application technologies or services that use virtual resources through the Internet [1]. The meaning of "resource" is widely-varied, i.e., infrastructure (IaaS), platform (PaaS), software (SaaS), etc. At the inception of cloud services, simple Internet storage services were popular [2]. Recently, cloud services have begun to be used as computing power because of their low initial cost and high flexibility. There are many cloud service providers offering computing power, and it has been become important to evaluate the performance of the computing power of cloud computing systems for the purpose of choosing adequate cloud service providers to match users' needs. Cloud computing systems use a large number of general PCs as calculating resources, therefore they are designed to be able to accomplish correct computation even if some of the PCs are not functioning. A retransmission and recalculation technique is used for a fail-proof mechanism, and this causes a time lag in the overall computation. This is one of the reasons why evaluation of overall computing power of a cloud computing system is difficult. The failure probability of each calculating node, the time lag to detect failure, and the delay of recalculation affect the overall computation time. Therefore these factors make precise evaluation difficult.

In this paper, we propose an evaluation method for the computing power of a cloud computing system using a simple model of cloud computing systems. We make assumptions about the hardware and software control of the model. Then, we introduce a concept called the "round number," which is the maximum number of times to invoke calculation, including recalculation, among all the subproblems divided by the controller node of the system. The overall computation time is strongly affected by the distribution of the round number. We derive a formula for the distribution function of the round number. Then we approximate the frequency of the overall computation time for each round number using normal distribution. We approximate the total frequency of the overall computation time by summing the approximation of the computation time for each round multiplied by the distribution function of the round number. Then, we execute a distributed computer simulation corresponding to the proposed model of cloud computing systems. Comparing the approximation using the distribution function and the approximation of frequency of the overall computation time for each round number with the simulation result, we conclude that the overall computation time of cloud computing systems can be estimated by the proposed method.

2. Model of Cloud Computing

Real cloud computing systems have a complicated control and recalculation mechanism that is affected by the random occurrence of failure. Therefore, estimation of the total computing time of a real cloud computing system is difficult. In this section, we explain the assumptions we made to establish a simple model of a cloud computing system for the purpose of estimating the overall computation time.

Assumptions about hardware

- The cloud computing system consists of one control node and a sufficient number of calculating nodes. All the calculating nodes have the same processing speed.
- When the control node orders the calculating nodes to start calculation, they either stop with failure probability p or carry out the calculation and return correct answers.
- The computing time of the control node is negligible compared to the computing time of calculating nodes.
- The communication time between nodes is also negligible compared to the computing time of calculating nodes.

Assumptions about control

- The overall problem given to the cloud computing system is divided into n subproblems by the control node.
- The control node distributes the divided subproblems to calculating nodes, checks if each calculating node is working, orders other calculating nodes to recalculate failed problems if needed, and aggregates the results of subproblems.
- Calculating nodes return ACK responses when the control node checks if they are working.
- The control node decides that a calculating node is out of order if the control node fails to receive an ACK responses from the calculating node within T_e after the calculation order. In this case, the control node immediately orders another node to start the recalculation of the subproblem that was distributed to the failed node. The control node repeats this until it has obtained the results of all the subproblems.

Let $\mathcal{P}$ be the original problem given to the cloud computing system. The control node divides $\mathcal{P}$ into n subproblems. We call the subproblems $\mathcal{P}_1, \mathcal{P}_2, \ldots$ and $\mathcal{P}_n$, and we call the answers to the subproblems $\mathcal{A}_1, \mathcal{A}_2, \ldots$ and $\mathcal{A}_n$ respectively. The control node sends each $\mathcal{P}_i$ $(1 \leq i \leq n)$ to a calculating node and receives the result $\mathcal{A}_i$ if the calculating node works. The control node combines the results $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ after it has received all of them and creates the result of the original problem $\mathcal{P}$.

For example, let the original problem $\mathcal{P}$ be "How many prime numbers exist that are less than one billion?" The control node may divide $\mathcal{P}$ into the following ten subproblems;

$\mathcal{P}_1$: How many prime numbers are there from 1 to 100,000,000?

$\mathcal{P}_2$: How many prime numbers are there from 100,000,001 to 200,000,000?

$\vdots$

$\mathcal{P}_{10}$: How many prime numbers are there from 900,000,001 to 1,000,000,000?

After the control node has received all the results $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{10}$, it simply adds the number of prime numbers in each section and obtains the result of the original problem $\mathcal{P}$.

3. Computation Time in the Proposed Model

We introduce a concept called the round number, which strongly affects the overall computation time. We show a formula for the probability distribution function of the round number. Then, we show an approximation of the overall computation time including recalculation time caused by failures.

3.1 Round Number

If the failure probability p equals zero, all of the $\mathcal{P}_i$ $(1 \leq i \leq n)$ are calculated only once until the control node has received the results of all the subproblems. In the case that a failure occurs when a calculating node is ordered to calculate $\mathcal{P}_i$, the control node detects the failure after T_e, then it orders another calculating node to recalculate the subproblem $\mathcal{P}_i$.

Assume that the failure probability p is given and consider a trial calculation from start to finish where the control node obtains all the results $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ including the process to check if calculating nodes are working and the time for the recalculating process. Let R_i be the number of times that the control node has ordered calculating nodes to calculate the particular subproblem $\mathcal{P}_i$ in the trial. If p equals zero, $R_1 = R_2 = \cdots = R_n = 1$ because each subproblem is calculated only once. Consider the case where q is nonzero. First, we describe the process for a particular subproblem $\mathcal{P}_i$. If the calculating node that is ordered to calculate the "first" calculation of $\mathcal{P}_i$ succeeds, R_i equals 1. If a failure occurs at the first calculation of $\mathcal{P}_i$, the control node orders another calculating node to recalculate $\mathcal{P}_i$. If the "second" calculation succeeds, R_i equals 2. In this manner, if the subproblem $\mathcal{P}_i$ is succeeded at r-th calculation for the first time, R_i equals r. In general, the values of R_i $(1 \leq i \leq n)$ differs among various i for the trial. We define the round number for a trial as the maximum value of R_i among $1 \leq i \leq n$.

The round number is crucial for the estimation of overall computation time. Consider the case where the time required by successful calculation of a subproblem is a constant T_c. If the round number of a trial is r and one of the problems for which the control node has ordered the calculation r times is $\mathcal{P}_j$ $(1 \leq j \leq n)$, one of the most time consuming problems is $\mathcal{P}_j$. Therefore, the overall computation time is dominated by the overall time required to receive the result of $\mathcal{P}_j$. The time required until the control node has received the result of $\mathcal{P}_j$ is the time for $r-1$ occurrences of failure detection followed by one successful calculation. Therefore, the overall computation time where the round number is r is:

$$(r-1)T_e + T_c \tag{1}$$

The distribution of the overall computation time is given by (1) and the discrete probability distribution of the random variable of the round number $P(R = r)$. The overall computation time is $(r-1)T_e + T_c$ with the probability $P(R = r)$.

In the more general case where the time required by the successful calculation of a subproblem fluctuates, this evaluation is still useful to approximate of the overall computation time.

3.2 Distribution of the Round Number

We derive a formula for the probability distribution function for the random variable of the round number $P(R = r)$. The overall computation time under the proposed model of cloud computing is exactly provided in the case where the time required by the successful calculation of a subproblem is constant.

First of all, we consider a particular subproblem $\mathcal{P}_i$. R_i stands for the number of times times the calculation for $\mathcal{P}_i$ was ordered before the calculation completed. The case where R_i is equal or smaller than r is the complementary case where all of the first r consecutive orders for $\mathcal{P}_i$ fail. The probability that all of the first r consecutive orders for $\mathcal{P}_i$ fail is p^r, therefore, the probability that R_i is equal or smaller than r for a trial is

$$(1 - p^r). \tag{2}$$

The case where the round number for a trial is equal or less than r is the intersection of the cases where R_i is equal or less than r for all i $(1 \leq i \leq n)$. The cases where R_i is equal or smaller than r for all i are independent. Therefore, the probability that the round number for a trial is equal or less than r is given by the following expression.

$$P(R \leq r) = (1 - p^r)^n. \tag{3}$$

The probability that the round number for a trial is exactly r is "the probability that the round number for a trial is equal to or less than r" minus "the probability that the round number for a trial is equal to or less than $r - 1$", therefore the following theorem holds.

Theorem 1: The probability that the round number for a trial is exactly r is

$$P(R = r) = (1 - p^r)^n - (1 - p^{r-1})^n. \tag{4}$$

3.3 Approximation of the Overall Computation Time

Under the proposed model of cloud computing, the overall computation time in the case where the time required by the successful calculation of a subproblem is constant is given by (1) and the distribution function (4). In a realistic case, the time required by the successful calculation of a subproblem is not constant. In such cases, the probability distribution function (4) still holds but T_c of the overall computation time in (1) fluctuates.

We propose an approximation method where we use the distribution function (4) and a type of probability density function to approximate the value of (1). Let $f_r(x)$ be the probability density function to approximate (1) for a particular round number r, where x is the parameter of the overall computation time. In this paper, we approximate (1) by a normal distribution function

$$f_r(x) = \mathcal{N}(\mu, \sigma^2) \tag{5}$$

where $\mathcal{N}(\mu, \sigma^2)$ is the probability density function of the normal distribution;

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{6}$$

We approximate the overall computation time by summing $f_r(x)$ multiplied by (4) for all the positive integers r.

$$\begin{bmatrix}\text{The probability density function of the} \\ \text{overall computation time}\end{bmatrix} = \sum_{r=1,2,\ldots} P(R = r) f_r(x) \tag{7}$$

Estimating the parameters of $\mathcal{N}(\mu, \sigma^2)$, by simulation for example, we can approximate the overall computation time of the cloud computing system.

4. Experiment

We performed a distributed simulation experiment to confirm that the proposed method is effective. We explain the simulation system and the example problem we used for the experiment.

4.1 System Used for the Experiment

The environment of the experiment is shown in Table 1.

Table 1: Experimental environment

CPU	Intel Core2Duo CPU P8600 2.40GHz
Operating system	Windows 7 Professional
Development environment	C, Visual Studio
Network environment	100MHz switch, directly connected
The number of calculating nodes	10 nodes

The distributed simulation software was implemented almost exactly according to the assumptions described in section 2. However, we devised an implementation for recycling calculating nodes that have acted as failed nodes. In the implementation, calculating nodes return a "failure signal" instead of stopping entirely in case of failure, and the nodes are used for subsequent calculations. This contrivance reduces the number of required calculating nodes to n no matter how many failures happen.

The parameter n, the number of divisions, was 10. The failure probability p in the experiment was 0.1. We assumed that T_e, the time required to detect failure, was 0.303

seconds. This value is determined as one-twentieth of the largest actual calculating time of a subproblem. We repeated 4,988 trials of the example problem described below.

4.2 Problem Used for the Experiment

We performed an experiment of distributed computation to find the number of prime numbers in an interval. The original problem that is the input of the cloud computing system is defined as the following;

Original problem

$\mathcal{P}$: Find the number of prime numbers from 1,000,000,000 to 1,000,499,999.

The length of the interval to search for prime numbers is 500,000 in the original problem.

The original problem $\mathcal{P}$ was divided into ten subproblems, each with an interval length of 50,000. The subproblems are as follows:

$\mathcal{P}_1$: Find the number of prime numbers from 1,000,000,000 to 100,049,999.

$\mathcal{P}_2$: Find the number of prime numbers from 1,000,050,000 to 100,099,999.

$\vdots$

$\mathcal{P}_{10}$: Find the number of prime numbers from 1,000,450,000 to 100,499,999.

The results of the subproblems are combined by simply adding them together, and thus the result of the original problem is obtained.

The algorithm we used to check for primality of an integer k is a simple algorithm performed by dividing k by small positive integers up to $\sqrt{k}$.

5. Results and Consideration

First of all, we must confirm that Theorem 1 holds in the simulation. Theorem 1 says that the appearance probability of the round number is given by (4). Table 2 shows the comparison between Theorem 1 and the simulation results. The second column of Table 2 is calculated by (4), and the third column is the actual appearance probability in the experimental simulation for round numbers $1, 2, \ldots, 6$. The higher the round number, the smaller the appearance probability exponentially; however, we can confirm that Theorem 1 holds approximately in the experimental simulation.

Table 2: Appearance probability of round numbers

Round number	Theorem 1	Simulation
1	3.49×10^{-1}	3.64×10^{-1}
2	5.56×10^{-1}	5.38×10^{-1}
3	8.57×10^{-2}	8.82×10^{-2}
4	8.96×10^{-3}	9.82×10^{-3}
5	9.00×10^{-4}	4.01×10^{-4}
6	9.00×10^{-5}	0

According to Section 3.3, we approximate the frequency of the overall computing time of each round number with a normal distribution. Table 3 shows the average μ and the standard deviation σ when regarding the frequency of each round number as a normal distribution. We show the parameters only up to the round number 3 because the appearance probability is exponentially small when the round number is large. The differences of μ between each round number are approximately 0.3. The value of T_e equals 0.303 in the experiment, therefore this suggests that (1) roughly holds when T_c is not constant.

Table 3: Parameters of Normal Distribution

Round number	Average(μ)	Standard deviation(σ)
1	3.275	0.3957
2	3.545	0.3746
3	3.849	0.3616

Then, we compare the approximation by a normal distribution using the parameter in Table 3 and the simulation result for each round number. Figures 1, 2 and 3 are comparisons of the frequency of the overall computation time when the round number is 1, 2, and 3 respectively.

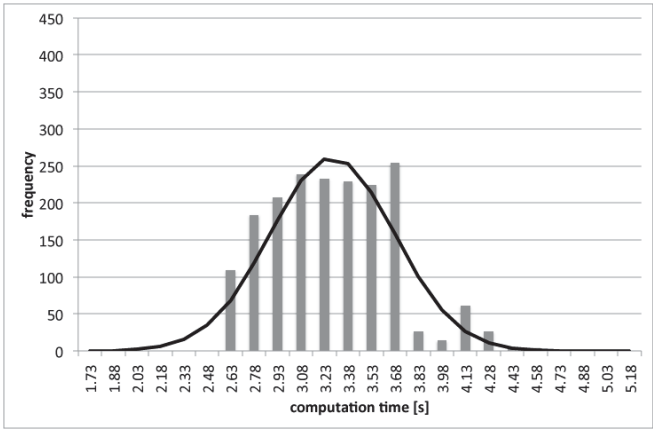


Fig. 1: Approximation and simulation (round number = 1).

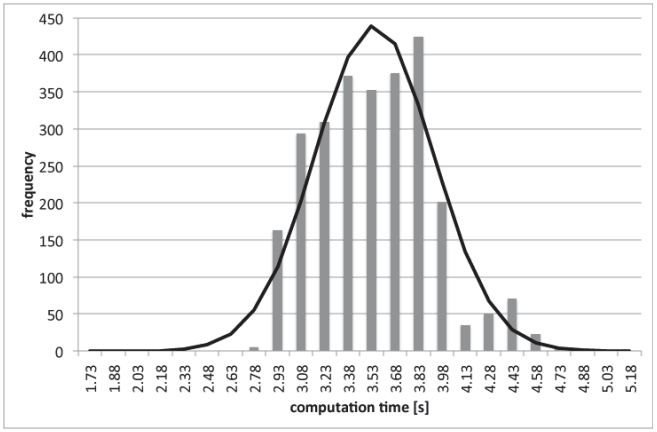


Fig. 2: Approximation and simulation (round number = 2).

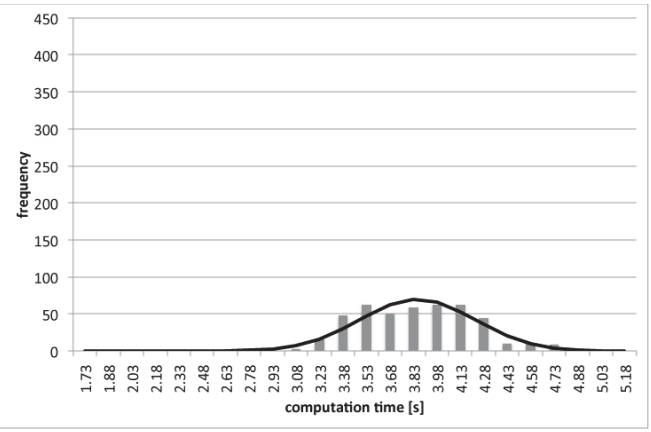


Fig. 3: Approximation and simulation (round number = 3).

The vertical axis shows the frequency and the horizontal axis shows the overall computation time in Figures 1, 2 and 3, where the lengths of intervals of the horizontal axis are 0.15 seconds and the labels are the medians of sections. The lines on the graphs indicate approximations by normal distributions using the parameters in Table 3. The bars on the graphs indicate the actual frequencies calculated by the simulation. To correspond to the vertical axis of the frequency in the simulation, the values of the lines are the interval integrals of normal distributions multiplied by the number of total trials (4,988) and by the appearance probabilities (the second column of Table 2).

Figure 4 is the comparison of the total overall computation time between the approximation by the proposed method and the actual frequency calculated by the simulation. The

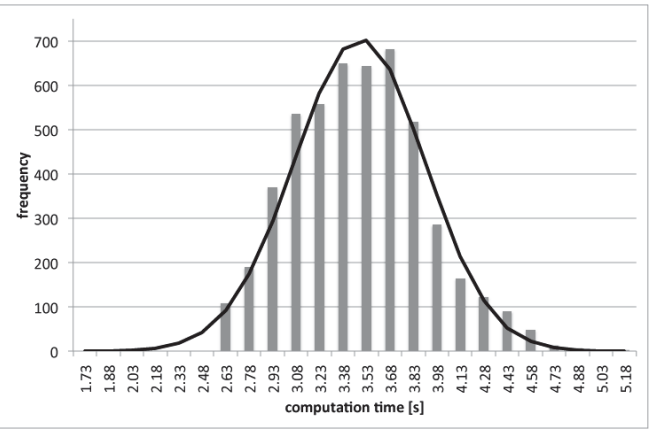


Fig. 4: Approximation and simulation (total).

approximation by the proposed method is given by (7); however, we used only $r = 1, 2$ and 3 for the summation in Figure 4 because events with larger round numbers are exponentially unlikely and we couldn't get enough data in the simulation. Therefore, the line in Figure 4 is calculated by simply adding the lines of Figures 1, 2 and 3. The bars in Figure 4 show the frequency of the total overall computation time of all the trials. From Figure 4, it can be stated that the approximation by the proposed method is a good approximation visually, whereas the approximation of each round number by normal distribution shown in Figures 1, 2, and 3 is not. This suggests that the effect of Theorem 1, the distribution of the round number, is dominant to the overall computation time and the effect of precision in the approximation of each round number shown in Figures 1, 2 and 3 is limited.

6. Conclusion

Cloud computing systems are designed to allow for failures of the participating calculating nodes. To overcome the impact of node failures, cloud computing systems have a mechanism to manage detection of failure and order recalculation. Such a process makes estimation of the computing power of cloud computing systems difficult.

In this paper, we proposed a method to approximate the computing power of cloud computing systems. We proposed a simple model of a cloud computing system that represents the fail-proof mechanism of cloud computing. We introduced the important concept of the round number, which strongly affects the overall computation time. We proved a theorem that provides a formula to represent the probability that the round number r occurs in a trial. If the time required by the successful calculation of a subproblem is constant, the distribution of the overall computation time is strictly given by the formula we derived. In the more general case where the time required by successful calculation of a subproblem is not constant, we proposed a method to approximate by normal distributions for each round number. Correctness of the theorem and effectiveness of the proposed method was demonstrated by distributed simulation. The result of the experiments suggests that Theorem 1 results in a strong effect even in the general case where the time required by successful calculation of subproblems fluctuates.

7. Future Problems

As mentioned in section 5, it is difficult to analyze events with a large round number because the appearance probability is too small to get enough statistical data. Therefore, it is difficult to determine the parameter of average and standard deviation of normal distribution for cases using large round numbers. We predict that the average parameter of normal distribution increases by T_e as the round number increases, and therefore the average for large round numbers is estimated by the average for small round numbers and the value of T_e. If the standard deviation parameter for large round numbers can be predicted using the standard deviation for small round numbers, an approximation of large round numbers becomes possible. To establish a way to estimate these parameters for large round numbers is an important future problem.

The distribution of the frequency of the total computation time for each round number strongly depends on the problems. We are planning to examine problems other than the problem used in this paper. Another future problem is to examine a distribution other than the normal distribution to approximate the distribution of each round number. We are planning to examine log normal distribution for this purpose.

The validity of the assumptions we made in the proposed model of the cloud computing system should be discussed using comparisons to real cloud computing systems.

References

[1] Amazon Web Services, "What is Cloud Computing?," [Online]. Available: http://aws.amazon.com/what-is-cloud-computing/

[2] Amazon Web Services, "Amazon Simple Storage Service," [Online]. Available: http://aws.amazon.com/s3/

[3] Y. Kuriyama, H. Yamamoto, "A Study on Modeling and Simulation of Cloud Computing," The 37th Symposium on Information Theory and its Applications, pp.311–314, 2014 (in Japanese).

[4] Y. Kuriyama, H. Yamamoto, "Approximation of the Processing Time for Cloud Computing and its Evaluation," The 38th Symposium on Information Theory and its Applications, pp.628–630, 2015 (in Japanese).

Prototype Development of a Twitter-Based Safety Confirmation System for Disaster Situations

Keisuke Utsu[1], **Akio Ogata**[2], **Kunihiko Sakurai**[1], **Mana Tsutsumi**[1], **Ayaha Suzaki**[1], **Rie Abe**[1], **Ayami Manaka**[2], **Hiroshi Ishii**[1], **and Osamu Uchida**[3]
[1]Department of Communication and Network Engineering, School of Information and Telecommunication Engineering, Tokai University, Minato City, Tokyo, Japan
[2]Course of Information and Telecommunication Engineering, Graduate School of Information and Telecommunication Engineering, Tokai University, Minato City, Tokyo, Japan
[3]Department of Human and Information Science, School of Information Science and Technology, Tokai University, Hiratsuka City, Kanagawa, Japan

Abstract - *Since the Great East Japan Earthquake of March 11, 2011, disaster-related information through sharing social media has been gathering attention. In this study, we develop a Twitter-based safety confirmation system as a web application. Then, we deploy the system on an Internet server and a multi-hop wireless LAN constructed by access point devices using micro-computer Raspberry Pi boards.*

Keywords: Disaster, Safety confirmation, Social media, Twitter

1 Introduction

After the occurrence of a large-scale disaster, many people try to confirm the safety of their family and friends. The confirmation is usually attempted through telephone using fixed-line phones and cellphones or by e-mail using cellphones [1].

Safety confirmation using information devices requires the communication infrastructure to be running normally. However, we have experienced many cases where the communication infrastructure has been disabled for a long time due to large-scale disasters. The Great East Japan Earthquake of March 11, 2011, seriously damaged the communication infrastructures. After the earthquake, many phone calls using fixed-line phones and cell phones were attempted, but could not connect to the network; this situation continued for a long time in many places [1]. Communication carriers took control of the traffic to prevent traffic congestion due to several phone calls attempting safety confirmation. A similar situation occurred during the Kumamoto Earthquake of April 2016.

Many alternative safety confirmation services are available; for example, communication carriers provide Business-to-Consumer (B2C) services and information service providers and security companies provide Business-to-Business (B2B) services. However, since these services rely on functioning communication infrastructures, they cannot be used for prolonged periods under lengthy failure conditions.

On the contrary, immediately after the Great East Japan Earthquake, although phone calls were unavailable due to traffic congestion, many people were able to communicate through the Internet. In addition, Twitter [2], one of the social media, was also used to share information on the damage caused at the disaster site. On Twitter, users can post short messages within 140 characters in length and pictures. The reason why many people used Twitter was that the exchanged data size on Twitter was rather small. Although smartphone use was not widespread at the time, Twitter was easily accessible with feature phones (conventional cellphones). The most important feature of Twitter is that users can distribute information easily to their followers. The following is an example where Twitter was used beneficially. An isolated victim sent an e-mail to her family abroad using her cell phone, which was the only available means of communication, and her family posted a rescue request on Twitter. Then, a vice-president in Tokyo who was not in the disaster site found the post. Finally, a rescue team from outside the disaster area was dispatched, and the rescue succeeded [3]. Similarly, many posts requesting rescue and relief supplies were posted after the Kumamoto Earthquake [4][5].

Recently, the use of information on SNS and Twitter for disaster situations has been gathering attention. Various studies have focused on systems that gather and utilize information across various SNSs [6][7][8]. We have also studied a web system that enables users to post disaster-related information in a useful format using smartphones on Twitter [9][10]; the posted information is linked to an online map.

Taking into account the above background, we are studying safety confirmation systems provided by social media. This study develops a prototype of a Twitter-based safety confirmation system named "T-@npi" for consumers. "T-@npi," the name of the proposed system, is after the

Japanese word "安否" which is pronounced *anpi* in Japanese and means the degree of safety of someone. Then, this study deploys T-@npi on the Internet server. In addition, assuming that communication infrastructures will be unavailable during disaster situations, this prototype system is deployed on access-point (AP) devices controlled by Raspberry Pi [11] microcomputer boards for multi-hop wireless local area network (LAN).

2 Existing Safety Confirmation System

This paper assumes the following three types of users:

- Sending user: User is sending a safety information message.
- Confirming user: User is confirming the safety information message of their family or friends.
- Supporting user: Users, such as staff of local governments, are confirming the safety of the disaster site.

Disaster Emergency Message Dial 171 [12] and Disaster Emergency Message Board Web171 [13] are services for consumers in Japan. These services are provided by communication carriers when excessive traffic congestion results due to the large number of safety confirmation phone calls.

Disaster Emergency Message Dial 171 is a service to record voice messages. The users can use the system by dialing 171 using fixed-line phones or cell phones. Sending users in the disaster area must input the fixed-line phone number of the confirming user. Then, the sending users can record a message for the confirming users. The problem with this system is that the user must know the fixed-line phone number of the recipient. In addition, since only fixed-line phones in the disaster areas are allowed to use the system, the users who do not have fixed-line phones cannot use the system.

Disaster Emergency Message Board Web171 is a service for registering text messages. The users can use the system with any type of information device. The system uses the phone number for identification. Here, the phone numbers are not limited to fixed-line phone numbers; cell phones are also allowed to use the system.

In the Great East Japan Earthquake, both 171 and Web171 were not used by many people because the communication infrastructure had been unavailable for a long time at the disaster site [1]. A notable reason is that the systems were not generally used by the people.

Google Person Finder is a web service to help people reconnect after a disaster [14]. The service was first provided in the aftermath of the Great East Japan Earthquake, and 670,000 calls were registered.

Facebook [15], the most popular SNS in the world, also provides a safety confirmation service. The service asks the user in the disaster site to reply to confirm their safety, and the response is sent to his/her friends on their timeline. Facebook is used by many for communicating among individuals rather than with an unspecified number of people because the users generally use their real name. Twitter is more effective than Facebook for communicating with an unspecified number of people. In addition, the LINE communication system [16] is better than Facebook for confirming one's safety to close friends.

Regarding Twitter, a system that estimates safety based on posts has been proposed [17]. In addition, a system that estimates safety based on life log data has been studied for supporting safety confirmation using SNS [18].

3 Twitter-based Safety Confirmation System, T-@npi

This section describes the development of a safety confirmation system using Twitter. The system is named "T-@npi." We adopted Twitter because it enables users to easily post and distribute information. In the existing 171 or Web 171 systems, sending users can only communicate their safety information. These systems cannot support sending users requesting rescue; the T-@npi system should support rescue requests. In addition, since Twitter is an open SNS with anonymity unlike Facebook and LINE, a system using Twitter is helpful for mutual and public assistance.

3.1 Outline of the system

User devices are assumed to be smartphones, tablet PCs, or general PCs. The system is deployed on the Internet server. The information devices are connected to the server by mobile network lines such as Long Term Evolution (LTE), 3G, or fixed network lines such as Fiber to the Home (FTTH) or xDSL. The system is constructed by programs and web pages of PHP (Hypertext PreProcessor) and JavaScript. They are deployed on the server. Twitter API is called by PHP script posts and searches the information on Twitter.

The outline and use cases are shown in Fig. 1. The sending users send their safety information to the confirming users, and the information is posted on the timeline of account @anpi_test on Twitter by Twitter API [19]. The confirming user can find the safety information of the sending user from the timeline of account @anpi_test. Also, the confirming user can distribute the information on Twitter. The supporting user can grasp the damage status of the disaster area where they support and see the list of the safety information submitted by the sending user from the timeline of account @anpi_test.

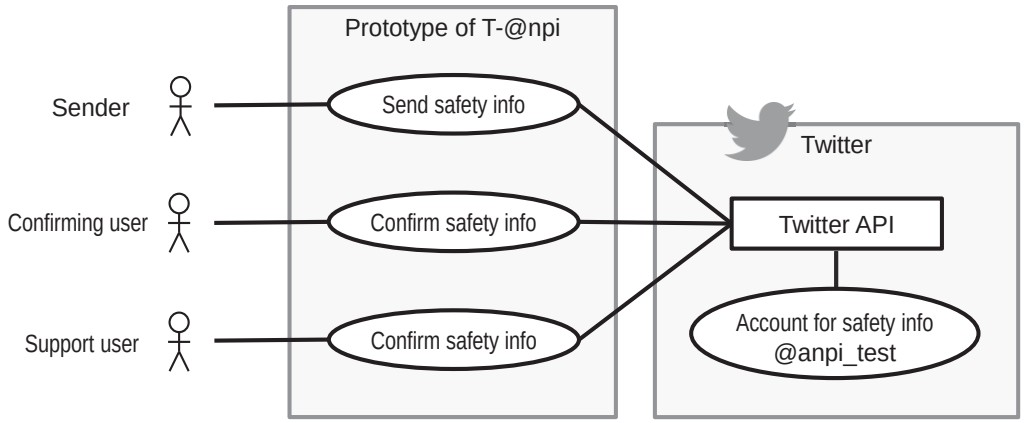


Fig.1. Outline and use cases of the system

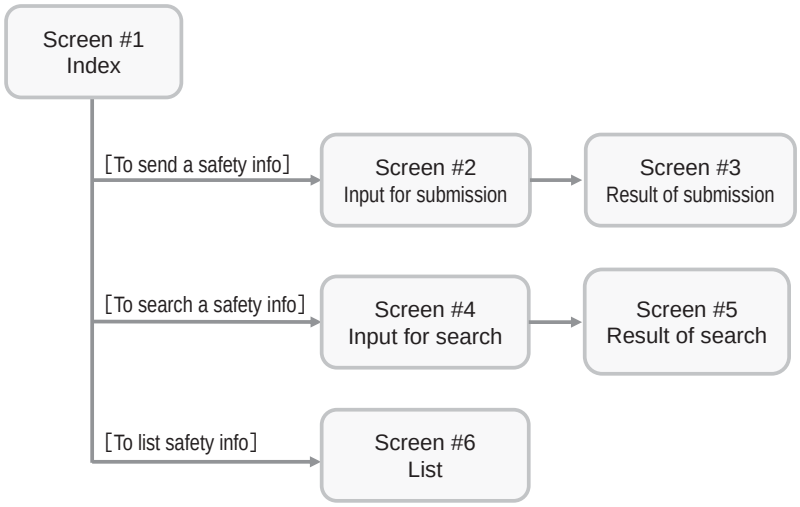


Fig. 2. User page transitions

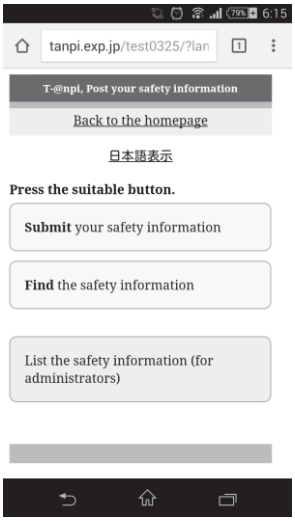


Fig. 3. Screen #1 (Index)

The advantages of using Twitter are as follows. Since Twitter is a very popular service, users are experienced and can use the system easily. Since users do not use the existing services such as 171 and Web171 daily, they are not familiar with the system operation; this has been a bottleneck for the acceptance of these systems. Our proposed system will be a solution to this problem. On the other hand, if systems need to have dedicated applications installed on a user's device, then the application must be distributed by the application store's website beforehand. However, it is not easy to register applications at application stores. In addition, some users are reluctant to install these applications because they are not used frequently. For supporting users, the proposed system has no cost. Since information exchange takes place on Twitter, the proposed system does not require large-scale hardware. Furthermore, since the identification of individuals is distinguished by Twitter IDs, the system manager does not need to maintain individual information.

3.2 User Interface and Operation

The user page transition is shown in Fig. 2. The screen shots are shown in Fig. 3-9. These figures were captured on Xperia J1 compact with Android 4.4 sold by Sony mobile communication Japan Inc. We have confirmed that similar screens were displayed on iPhone 6 or 5s with iOS9 sold by Apple Inc. The user can switch the language of the instructions between English and Japanese. Functions and operations on each screen are explained as follows.

Screen #1 (Fig. 3): Index

First, when a user accesses the system via a web browser, this page appears. The screen has three buttons. The user presses the appropriate button to select the operation. The button of "Submit your safety information" is for the sending user to submit a safety information. When the button is pressed, the screen transits to Screen #2. The button of "Find the safety information" is for the confirming user to find the safety information of the sending user. When the button is

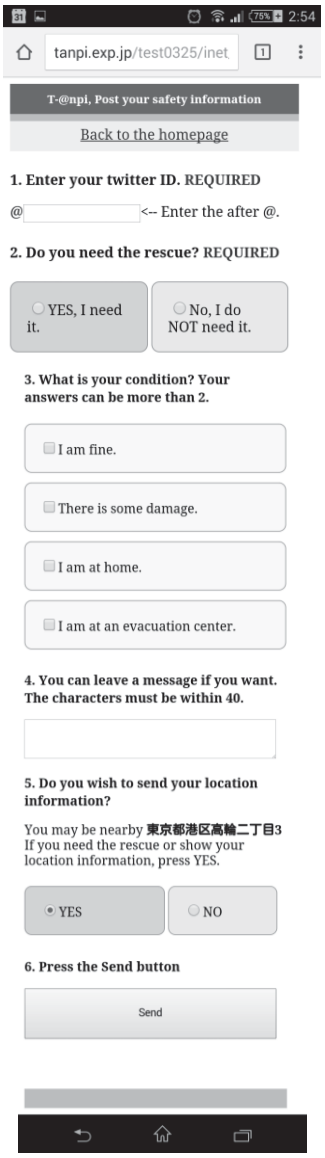


Fig. 4. Screen #2
(Input for submission)

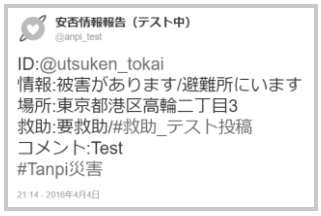


Fig. 5. Example tweet

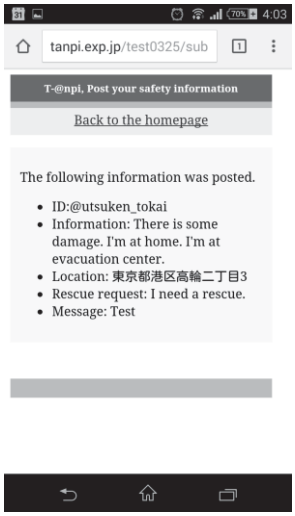


Fig. 6. Screen #3
(Result of submission)

pressed, the screen transits to Screen #4. The button of "List the safety information (for administrators)" is for supporting users to see the list of safety information submitted by the sending users. When the button is pressed, the screen transits to Screen #6.

Screen #2 (Fig. 4): Input for submission

This screen is for a sending user to send the safety information. Since the contents on the screen do not fit in the display size, some screenshots are combined in Fig. 4.

2-1) input of Twitter ID

The sending user enters one's own twitter ID; login on Twitter is not needed.

2-2) selection of the rescue request

If sending users need a rescue, they select "YES."

2-3) selection of the safety status

The sending users check more than one appropriate check box of their safety status among four choices: "I am fine," "There is some damage," "I am at home," and "I am at an evacuation center." The status choices are based on those in Web171.

2-4) additional message

A user may enter a message within 40 characters.

2-5) user's current location and the selection of whether to send it.

If the user accesses the system via the Internet, the web browser asks permission to provide the location information. When the user allows to provide the location information, the latitude and longitude information is obtained by GPS and a geolocation system, which are called by Java Script. The information is transformed into the address by the reverse geocoding service [20] and displayed. When the "Yes" radio button is checked in response to the question "do you wish to send your location information?," the address is sent with the safety information. Otherwise, "情報なし," which means "no location information," is sent. On the other hand, if the user's device cannot obtain its location coordinates or the user does not want to provide location information, then "情報なし (no location information)" is sent. In Fig. 6, the location information is displayed as "東京都港区高輪二丁目 3," which means "2-3 Takanawa, Minato City, Tokyo."

If a user accesses the system via the multi-hop wireless LAN, the question of "do you wish to send your location information?" does not appear; instead, the preset address of the access point device is sent automatically.

2-6) "Submit" button

Once the "Submit" button is pressed, a tweet described in Japanese, as shown in Fig. 5, is posted on Twitter. The description of the sample post in Fig. 5 is as follows.

ID: @utsuken_tokai
Information: There is a damage./I am at an evacuation center.
Location: 2-3 Takanawa, Minato City, Tokyo.
Comment: Test
#Tanpi-Disaster

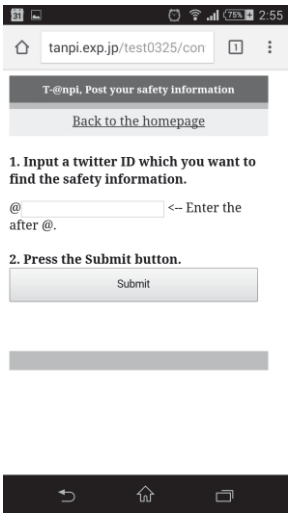


Fig. 7. Screen #4
(Input for search)

Fig. 8. Screen #5
(Result of search)

The screen then transits to Screen #3 (Fig. 6).

Screen #3 (Fig. 6): Result of submission

When the submission is completed, the submitted message is displayed. A sample of the submitted message is shown in Fig. 6. In the example, the hashtag #救助_テスト投稿 is included. Here, "救助" means "a rescue request" and "テスト投稿" means "a test submission."

Screen #4 (Fig. 7): Input to find information

4-1) Input of Twitter ID

Fig. 9. Screen #6
(List view)

The confirming user enters the ID of the sending user for which the confirming user wants to search. Here, the confirming user is not required to login on Twitter.

4-2) "Submit" button

The user presses the button after entering the information, and the screen then transits to Screen #5.

Screen #5 (Fig. 8): Result of search

The safety information of the confirming user can be found by searching the information submitted on @anpi_test by the GET method of Twitter API.

Screen #6 (Fig. 9): List view for supporting users

The safety information submitted on @anpi_test is listed by the GET method of Twitter API. The list is useful for supporting users to collect safety information and to support victims. Users can filter the information by location and sort and view the information. If the information includes a rescue request (#救助_テスト投稿) or "被害があります (There is some damage)," the background color of the information is highlighted.

4 Deployment of T-@npi

First, the prototype system of T-@npi is deployed on the Internet server to support users who can connect to the Internet. Next, the prototype system is deployed on a multi-hop wireless LAN comprising AP devices controlled by Raspberry Pi boards. Each access point operates as a web server; an outline of the entire network is shown in Fig. 10.

4.1 Deployment on the Internet server

The prototype system is deployed on the rented server on the Internet. As an operation test, post submissions are tried from March 11, 2016, to April 4, 2016. The test submissions are made both outdoors and indoors (at the stations and on the trains). The device is a smartphone Xperia J1 compact with Android version 4.4 produced by Sony Mobile Communications, Inc, was and is connected to the Internet by LTE or 3G lines. We submit 45 posts with the geolocation function enabled, and all 45 posts are successfully submitted with the address information. However, the address in some posts has position errors. Since some posts are submitted on the trains, the past addresses may be included. To mitigate this error, the system should be improved as follows. If a user finds that the obtained address is obviously incorrect, a message is displayed on Screen #2 to recommend the user to reload the page. Alternatively, the button to renew the address information is added on the page.

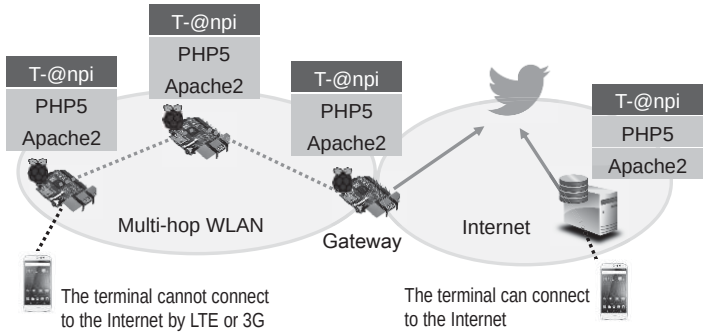


Fig. 10. Outline of the network system

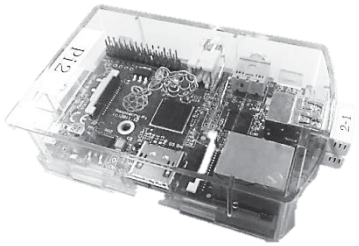


Fig. 11. Raspberry Pi Type B board

4.2 Deployment on the access point devices of the multi-hop wireless LAN

We develop AP devices for the multi-hop WLAN using Raspberry Pi boards [21]. The board is shown in Fig.11, and the specifications are shown in Table 1. Since the board is inexpensive, Linux-compliant, and consumes low energy, we can easily implement the AP functions. The OS on the board is Raspbian OS [22], on top of which necessary functions are implemented. Two USB-connected wireless LAN adaptors are installed on the board. One adaptor runs in the ad hoc mode and uses IEEE802.11g as the MAC layer, and the other runs in the infrastructure mode by hostapd [23] and uses IEEE802.11g as the MAC layer. Web server Apache [24] and PHP are installed in each AP. The static routing is configured in each AP to construct the multi-hop wireless LAN as shown in Fig. 10. In real situations, a dynamic routing protocol will be required to achieve flexibility and reliability. The selection or development of a suitable routing protocol will be addressed in further studies. The distance between the adjacent APs and between an AP and a user terminal is 10 m. One AP runs as a gateway node to connect to the Internet via the campus network. The user terminal connecting to the AP receives IP addresses for the terminal, default gateway, and IP address of Domain Name System (DNS) server by Dynamic Host Configuration Protocol (DHCP). For the test operation, we connect the smartphone Xperia J1 compact to each AP and test to submit the information five times. All submissions are successful. In addition, searching and finding the posted information is also successful.

Table 1 Spec of Raspberry Pi Type B

CPU	BCM2835 (700MHz)
RAM	512MB
USB ports	2
Board size	85.6x53.98x17 [mm]
Price	USD 35.00

5 Summary and future issues

This paper describes the prototype development of the T-@npi safety confirmation system using twitter. We first explain the operation and user interface of the system. Then, we explain the development of the system on the Internet server and the multi-hop wireless LAN constructed with APs made with Raspberry Pi boards. Future issues are as follows. In terms of the application of the prototype, sending users can post information without user logins. In the future, we plan to introduce user logins to confirm identities. In terms of network composition, the test network consists of one AP running as a gateway node with a connection to the Internet. In an actual network, the gateway would be a redundant configuration. In addition, we must study the provision for the case where a gateway node is disabled. After improving the system, we plan to collaborate with local governments to adopt the system.

6 Acknowledgments

This study has been supported by the COC (Center of Community), Ministry of Education, Culture, Sports, Science and Technology, Japan

7 References

[1] Internet white paper 2011, http://iwparchives.jp/iwp2011

[2] Twitter, Twitter Inc., https://twitter.com/

[3] SHUCHI PHP Online, http://shuchi.php.co.jp/article/943?p=1

[4] Asahi Shimbun Company , http://www.asahi.com/articles/ASJ4J34R6J4JULZU003.html

[5] Sankei Shimbun & SANKEI DIGITAL, http://www.sankei.com/west/news/160421/wst1604210120-n1.html

[6] DISAster-information ANAlyzer (DISAANA), http://www.nict.go.jp/en/index.html, National Institute of Information and Communication Technology

[7] Takahiro Okuma, Kayoko Yamamoto, "Study on a Social Media GIS to Accumulate Urban Disaster Information : Accumulation of Disaster Information during Normal Times for Disaster Reduction Measures", Socio-Informatics, Vol.2, No.2, pp.49-65, 2013 (in Japanese)

[8] Takuma Murakoshi, Kayoko Yamamoto, "Study on a Social Media GIS to Support the Utilization of Disaster Information : For Disaster Reduction Measures from Normal Times to Disaster Outbreak Times", Socio-Informatics, Vol.3, No.2, pp.17-30, 2014 (in Japanese)

[9] Osamu Uchida, Masafumi Kosugi, Gaku Endo, Takamitsu Funayama, Keisuke Utsu, Sachi Tajima, Makoto Tomita, Yoshitaka Kajita, Yoshiro Yamamoto, "A Real-Time Disaster-Related Information Sharing System Based on the Utilization of Twitter", The Fifth International Conference on Social Media, Technologies, Communication, and Informatics (SOTICS 2015), pp.22-25, IARIA, 2015

[10]Masafumi Kozugi, Hiroto Funakoshi, Keisuke Utsu, Sachi Tajima, Makoto Tomita, Yoshitaka Kajita, Yoshiro Yamamoto, Osamu Uchida, "Introduction of MGRS code into Disaster-Related Information Sharing System", IPSJ SIG Technical Report, 2016-GN-98, No.14, pp.1-8, 2016 (in Japanese)

[11]Raspberry Pi Foundation, https://www.raspberrypi.org/

[12]Disaster Emergency Message Dial 171, https://www.ntt-east.co.jp/en/saigai/voice171/ , NTT EAST

[13]Emergency Message Board Web171, https://www.ntt-west.co.jp/dengon/web171/english/ , NTT WEST

[14]Google person finder, https://www.google.org/personfinder/

[15]Facebook, https://www.facebook.com/

[16]LINE, http://line.me

[17] Masakazu Kyokane, Yukio Hori, Yoshiro Imai, "Twitter-based Information Collecting System for Safety Confirmation", IEICE Technical Report, pp. 7-11, KBSE2013-54, 2013 (in Japanese)

[18] Yuji Ikebata, Koji Tsukada, "A Proposal for Safety Confirmation System using Lifelog Data", IPSJ Kansai-Branch Convention, E-20, 2013 (in Japanese)

[19]Twitter API, https://dev.twitter.com/overview/documentation

[20]Reverse geocoding service, National Agriculture and Food Research Organization, Japan, http://www.finds.jp/wsdocs/rgeocode/index.html.ja

[21]Akio Ogata, Hirohide Matsuzaka, Hayato Taniguchi, Masaya Nomoto, Ayami Manaka, Koichi Saito, Minoru Fukuzaki, Hiroshi Ishii, Yasuhiro Nozawa, Keisuke Utsu, "Prototype Development and Performance Evaluation of Wireless LAN Access Points for Community Information Network", IEEE TENCON 2015, PID:524, 2015

[22]Raspbian, Raspberry pi foundation, https://www.raspberrypi.org/downloads/raspbian/

[23]hostapd, https://w1.fi/hostapd/

[24]The Apache HTTP Server Project, https://httpd.apache.org/

A Simulation Study of Broadcast Voice Streaming using BBISS over a Multi-hop Wireless LAN

Ayami Manaka[1], **Chee Onn Chow**[2], **Hiroshi Ishii**[3], **and Keisuke Utsu**[3]
[1]Course of Information and Telecommunication Engineering, Graduate School of Information and Telecommunication Engineering, Tokai University, Minato City, Tokyo, Japan
[2]Department of Electrical Engineering, Faculty of Engineering, University of Malaya, Kuala Lumpur, Malaysia
[3]Department of Communication and Network Engineering, School of Information and Telecommunication Engineering, Tokai University, Minato City, Tokyo, Japan

Abstract - *We have been studying a community network made of a multi-hop wireless LAN to inform citizens of daily or emergency information. On the other hand, we have proposed an information delivery method named BBISS (Broadcast Based Information Sharing System) for ad hoc communication environments. The system is suitable for the multi-hop wireless LAN. Also, we have proposed a broadcast voice streaming method named VoBBISS (Voice over BBISS) which can distribute the stream with high quality and efficiency. This study evaluates a performance of the broadcast voice streaming using BBISS for inter access points communications on the multi-hop wireless LAN by a network simulation.*

Keywords: Wireless LAN, Multi-hop, Voice streaming, Simulation

1 Introduction

Japan has a well-developed communication infrastructure. Smart devices such as smart phones and tablet PCs have been widely spread. However, it is difficult to support daily life for citizens by using smart devices. Since daily life support services using telecommunications carrier's lines may be expensive, they are not easily provided by local government. To provide the services at a low costs, we have studying a development of a multi-hop wireless LAN which is constructed by multiple access point (AP) devices. We have reported a development of the AP devices which is made by microcomputer board "Raspberry Pi" in [1].

On the other hand, large-scale disasters frequently happen in Japan. When the communication infrastructure is disabled due to disasters, emergency alerts and evacuation instructions are broadcast by local governments, police, or firefighters using microphones and speakers as usual. Since the audio broadcast is often poorly-heard for the citizens, it has problems in terms of immediacy and reliability. Furthermore, people still use radios and handy televisions to obtain the information. Even if the citizens have such devices, they have difficulty in getting information specific to the local area by them. To obtain the local information, the citizens have to wait for mass media coverages. However, the mass media may not cover local information in the disaster site. Therefore, it is necessary a way to distribute the local information in the community.

Regarding the above issues, we have studied the method to distribute the daily information, event announcements, and emergency notifications to the user devices under the multi-hop wireless LAN (WLAN). The network was assumed to be deployed in Minato city, Tokyo, Japan, and named Community Information Network (CIN), in [2]. To broadcast non-real-time contents such as text and image data, that paper studied to apply the Broadcast Based Information Sharing System (BBISS) [3] to information distribution among APs. Then, the effectiveness of the system was evaluated through the network simulations. On the other hand, to enable the emergency notifications and the evacuation instructions in voice, a broadcast voice streaming method is required. Regarding the broadcast voice streaming, we studied voice streaming over BBISS (VoBBISS) on ad-hoc communication environment and showed the effectiveness in [4].

This paper focuses on the broadcast voice streaming applying VoBBISS on the multi-hop WLAN based on [2]. Then the distribution performance is evaluated by the network simulation. Section 2 describes assumptions and requirements of the multi-hop WLAN and broadcast voice streaming on the network. Section 3 introduces VoBBISS. Section 4 evaluates the effectiveness of applying VoBBISS to multi-hop WLAN inter APs distribution of voice streaming by the network simulation. Lastly, Section 5 describes conclusions of the paper.

2 Assumption and Requirements

2.1 Assumptions of multi-hop WLAN

As locally launched communication network services, APs have been equipped to provide tourists with public Wi-Fi services in Japan [5][6]. On the other hand, as a technology, IEEE802.11s mesh network has been studied [7].

We consider that there are following two requirements to develop the locally launched network.

1. The network provides the residents and/or tourists with the internet connection service by deploying Wi-Fi APs. As a result, the network 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. The network is available even when the infrastructures of communication carriers are unavailable due to disasters, and the citizen can use the network for an alternative infrastructure.

To satisfy the above requirements at low costs and at the same time, we are studying a multi-hop WLAN as shown in Fig. 1. Each sub-area is composed with multiple APs. 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 network is composed of a number of sub-areas, each of which is composed of several APs. An AP in each sub-area is a gateway to the other sub-areas and/or the external network (Internet). How to realize the connections among sub-areas and between the network and the internet needs further study.

The inter-AP communication on the multi-hop WLAN was studied in [2]. In the paper, we studied the broadcast distribution of the download and play media such as text and images. The information source is assumed to have two cases: the one is specific point(s) such as city office and its brunches. The other one is unspecific point(s) under APs. In both cases, to reach generated packets to the user terminals, they must be delivered to each AP. Here, unicast communication is not appropriate for distribution to many destination. That is because link capacity is small in the wireless network and it results in low packet reachability. To enable the broadcast distribution effectively achieving high packet reachability, we discussed the applying of BBISS which uses broadcast communications to the inter-AP distribution in [2]. The method BBISS utilizes the characteristics of broadcast communication: broadcast communication on wireless media has redundancy and it has better fault tolerance than the unicast communication.

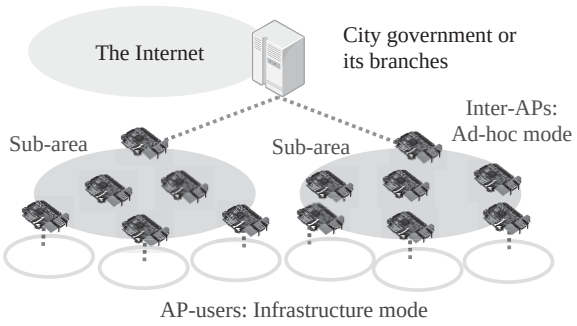


Fig. 1 Composition of the multi-hop wireless LAN

2.2 Requirements of broadcast voice streaming of multi-hop WLAN

As mentioned above, we have already studied the broadcast distribution of the download and play media such as text and images by BBISS. However, we have not yet studied the broadcast voice streaming performance on the network. Therefore, this study focuses on broadcast voice streaming for the emergency notifications.

This study focuses on the broadcast voice streaming for the user terminals which is connecting to the multi-hop WLAN. Here, this study is not concerned with voice codecs. The codec G.711 [8] which is a general codec is assumed to be used below.

For reference, voice streaming using existing flooding [9][10][11] is shown in Fig. 2. A packet includes information such as sequence number which is added by Real time Transfer Protocol (RTP) or another protocol is transferred to UDP layer, IP layer, and link layer. On the contrary, at the receiver side, receiving voice packets are reassembled. However, since packets are generated at a high rate in the voice streaming, the above methods are insufficient. The reduction of the packet losses and complement them are required in the distribution method.

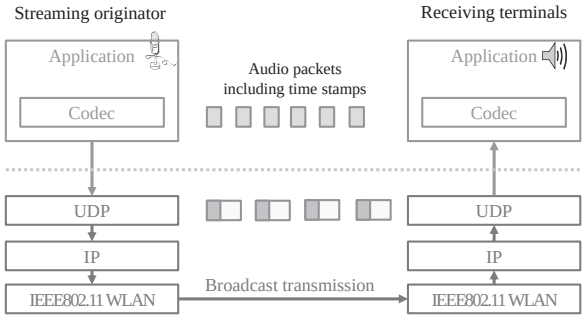


Fig. 2 Voice streaming by existing flooding methods

The broadcast voice streaming should meet the following requirements.

(a) Quality degradation due to collisions should be saved:

Since ad-hoc and multi-hop communication environment are lossy, and streaming applications generates packets at a high rate, packet losses significantly occur. Since we assume a one-way voice streaming of emergency notifications and evacuation instructions, the percentage of unreached packets at the receiving terminal should be 5.00% or less, which is sufficient quality in the assumed environments, taking into account of [12]. For the same reason, a few seconds buffering delay is acceptable. Therefore, the system can adopt retransmissions of the unreached packets. Based on the above, the system aims for 5.00% or less of unreached packets.

(b) Redundant packet exchanges and overhead should be saved:

The saving the redundant packet exchanges is effective to save quality degradation due to collisions. In addition, since the battery capacity of the node is limited, the redundant packet exchanges should be saved. By the same reason, overhead of under UDP layer which is generated in packets transfer should be reduced.

(c) Overhead of HELLO packets should not be generated:

HELLO packets which are generated to collect and distribute the topology information should not be generated to reduce traffic load and the number of packet exchanges.

3 VoBBISS

We have proposed VoBBISS which is a broadcast voice streaming method on the ad-hoc communication environment in [4]. The system VoBBISS and voice codec are independent, and this study does not take into account of packet loss concealment (PLC). The details of the system are shown in the following.

3.1 Broadcast voice streaming by VoBBISS

Figure 3 shows the outline of proposed system. The BBISS was primarily assumed to be adopt to the download and play media such as text and images which are non-real-time data sized tens of kBytes to hundreds of kBytes. Therefore, BBISS itself cannot be applied to the voice streaming that generates packets at a high rate. For the streaming application in VoBBISS, the voice packets are conjunct at the streaming originator. Finally, conjunct packets are transferred by BBISS. Then, the conjunct packet are reassembled to the voice packets at the receiver side.

(A) Conjunction and disjunction of voice packets

At the streaming originator, the voice packets including sequence numbers are combined with a certain length of time. As the number of packets to be combined get larger, the length of a buffering time which is waiting time for the playback start will be longer. The number of packets to be combined must be larger values than multiple number of the BBISS packet payload size, and it should be similar values of multiple number of the BBISS packet payload size. In an example shown in Fig. 3, the packets for 0.5s (i.e. 25 voice packets) are combined. The conjunct packets are transferred to the BBISS layer. At the receiver, conjunct packets which are received at the BBISS layer are reassembled to the voice packets. Then, voice packets are transferred to a playback application.

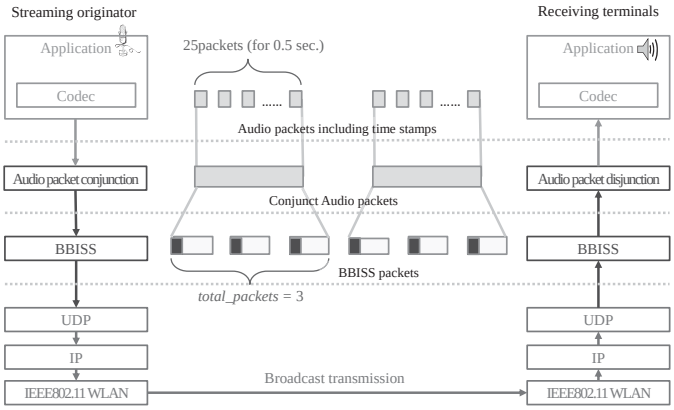


Fig. 3 Voice streaming by BBISS

(B) Packets distribution by BBISS

Broadcast Based Information Sharing System operates on the UDP/IP layer. The detailed explanation of BBISS operation and its features is described in [2]. The outline of BBISS is as follows. Below "send" means "broadcast" at the MAC layer in this section.

The streaming originator divides the information (a conjunct audio packet) into multiple packets (BBISS packets). The division number is determined by *total_packet* which is a preset value. An example of the case where *total_packet* = 3 is shown in Fig. 3. Here, the setting is depends on the number of packets combined in (A). How to set the value will be mentioned later. Then, the node sends the packets sequentially with a fixed time interval which is determined by *send_interval* (as Sending state in Fig. 4). 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

(as Relay decision state in Fig. 5). 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. 6).

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. 7. The number of trials to send retransmission request packet is limited to a predetermined threshold (*req_threshold*).

3.2 Features of VoBBISS and solution for the requirements

If voice packets are transferred as they are, the lower layer contentions are serious. That is because, the packet size of the voice packet is smaller than that of other applications. In addition, traffic load is heavy due to the overhead under the UDP layer. To solve the problems, VoBBISS can reduce the overhead as below. The audio packets are conjunct at first, and then the conjunct audio packets are divided into the BBISS packets which can be delivered using BBISS. The operation can reduce the quality degradation of the playback, and contributes the requirements (a) and (b).

The retransmission operation, which is using within several seconds buffering, is taken to reduce the quality degradation of playback in VoBBISS. When some packets are not reached the receiving node, the NACK based retransmission control is taken. The operation can reduce the quality degradation of the playback, and contributes the requirement (a).

In addition, in Relay decision state of BBISS, the relay transmission is canceled when the receiving node detects the number of relaying node is reached *relay_threshold*. The operation can reduce the redundant relay transmissions, and contributes the requirement (b). Also, the operation does not require HELLO packet exchanges, therefore it can contribute the requirement (c).

In the previous study [3], VoBBISS can deliver in better packet reachability than existing methods, on the ad hoc communication environment. The quality is sufficient for voice streaming. Furthermore, since the number of transmitted packets are reduced, it can consume the energy efficiently.

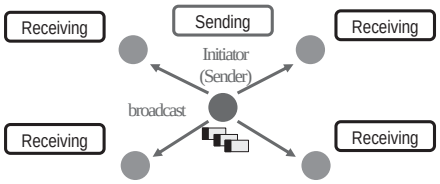


Fig. 4 Sending and Receiving states in BBISS

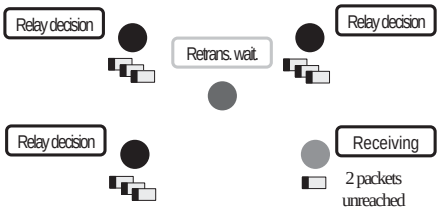


Fig. 5 Relay decision state in BBISS

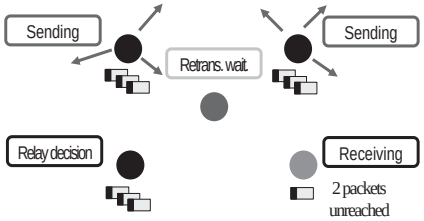


Fig. 6 Relay decision and Sending states in BBISS

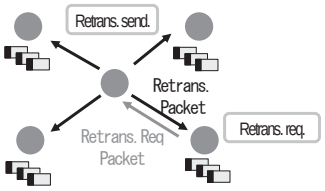


Fig. 7 Retransmission operation for complementing unreached packets in BBISS

4 Performance evaluation of VoBBISS on the multi-hop WLAN

In this section, the performance evaluation of broadcast voice streaming by VoBBISS on the multi-hop WLAN which is assumed to be applied in the Merry Load Takanawa Street, Takanawa area, Minato City, Tokyo, Japan, as a model case. Here, the evaluation focuses on communication performance of the inter-APs. The communication among AP-user terminals is a further issue.

4.1 Simulation environment

We used OPNET Modeler 17.5 as a network simulator [13]. The simulation area size could cover Merry Road Takanawa Street. The APs were assumed to be equipped there covering the street in about 100m intervals, as shown in Fig. 8. The radio communication range (transmission radius) was 150m. The MAC layer protocol was IEEE802.11a, and the data rate was 54Mbps. AP1, 6, and 8 were in one hop radio range of AP0. AP2 and 7 were two hops away from AP0. AP3 was three hops away from AP0. AP4 and 5 were four hops away from AP0. The radio channel for inter-APs and that for the AP to the user terminals were assumed not to interfere each other. In the simulation, traffic was assumed to be generated from the AP.

We assume the voice stream as G.711, and voice playback time (length) was 30s, i.e. 50 packet/s that was equivalent to a total of 1500 packets. The sending packet size was 200Byte (Payload 160Byte + UDP/IP header that was equivalent to 40Byte).

The numbers of APs generating the information simultaneously (source APs) were 1 (AP_a, in Fig. 8), 2 (AP_a and AP_b), or 4 (AP_a, AP_b, AP_c, and AP_d). As larger the number was set, the traffic load got heavier. The voice stream was distributed by SF or VoBBISS. The number of trials for each distribution method was 10.

For SF simulations, *send_interval* was set 20ms, and waiting time for the packet relaying was set a value in the range of (20, 200) ms. For BBISS simulations, *send_interval* was set 166ms, and waiting time for the packet relaying was set the value in the range of (16.6, 166) ms, the payload size of the BBISS packets was 1480Byte. The other parameters for BBISS were set according to [14]: *relay_threshold* was set 2, and *req_threshold* was set 3. The following describes the number of packets constructing a BBISS packet (*total_packet*). The number of combined voice packets must be less than or equal to the value multiple of the payload size of the BBISS packet (The number of combined voice packets * The size of voice packets ≤ *payload_size* * *total_packet*). In addition, it should be close to a multiple of the payload size of the BBISS packets. This study was simulated as the number of conjunct packets were 25 (0.5s), that was equals *total_packet* = 3.

Although the streams were generated at nodes connected to the APs in real situations, the voice streams were assumed to be generated at APs in the simulations. That was because, the channels among APs and that for the user terminals to APs were independent in the study. The numbers of APs generating the information simultaneously (source APs) were 1 (AP_a, in Fig. 8), 2 (AP_a and AP_b), or 4 (AP_a, AP_b, AP_c, and AP_d).

Fig. 8 Assumption of the APs' placement

4.2 Evaluated items

Since this study focuses on the one-way voice streaming in the disaster situations, a few seconds of buffering time is acceptable. The evaluated items were the following (i) and (ii).

(i) Percentage of receiving packets [%]

The percentage of voice packets received at the APs of the generated voice packets (1500 packets) in each stream. Here, streaming originator APs were excluded. The percentage values were averaged in all APs. To deliver the packets to user terminals under the multi-hop WLAN, as many as possible of generated packets should be reached to the APs. Therefore, as the percentage gets larger, the performance will be better.

(ii) End-to-End delay time of voice packets [s]

The average end-to-end packet delay between the originator AP and the receiver APs was evaluated. Then, the values were averaged in all generated packets. The generated packets should be reached as early as possible. Therefore, as the time gets shorter, the performance will be better.

4.3 Evaluation result

(i) Percentage of receiving packets [%]

The simulation result for average of 10 trials is shown in Fig. 9. The deviation bars in the graph show the range of average values +/- standard deviation values.

In SF, the number of source APs got more, the traffic loads and packet collisions were increased. Therefore, the percentage of receiving packets was decreased. In VoBBISS, even though the number of source APs was 4 (that was the maximum case), the percentage was 95.8%. This means that almost all of voice packets are reached the APs. The reason is as follows: in SF, the nodes relay the packet every time when the nodes receive it. On the other hand, in BBISS, the nodes relays the conjunct packet after the node receiving it. Therefore, BBISS can reduce the traffic load and the data frame collisions, compared with SF. In addition, the retransmission control in BBISS contributes to improve the packet reachability.

(ii) End-to-End delay time of voice packets [s]

The simulation result for average of 10 trials is shown in Fig. 10. The deviation bars in the graph show the range of average values +/- standard deviation values.

The end-to-end delay time values for VoBBISS were about 1s, and those were larger than those for SF in all cases of the number of source APs. The reason is as follows: the retransmissions were operated in BBISS at the case where some packets were unreached. In addition, since BBISS combined and divided voice packets, the delay time was expanded. However the end-to-end delay time is equivalent to the minimum waiting time for the playback start at the user terminal. That delay time for BBISS is negligible for the emergency broadcast to the user terminals.

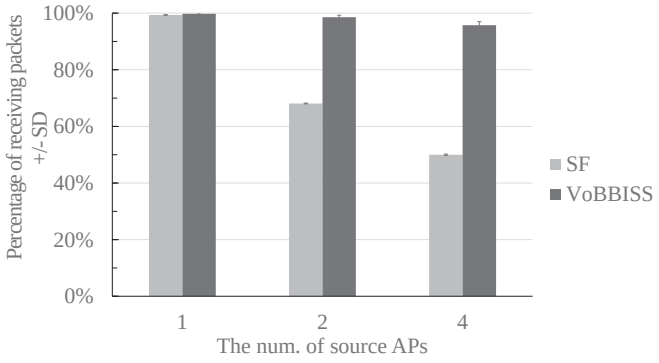


Fig. 9 Simulation result for percentage of receiving packets

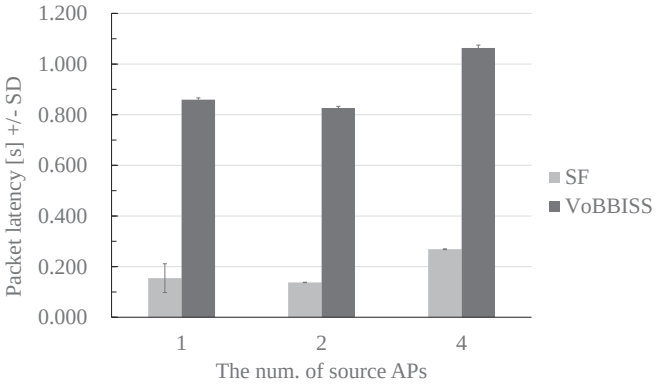


Fig. 10 Simulation result for percentage of packet latency

5 Conclusions

This study evaluated a performance of the broadcast voice streaming using VoBBISS for the multi-hop wireless LAN which was assumed to be applied in Takanawa, Minato city, Tokyo. The performance for inter-APs distribution was evaluated by the network simulation. The simulation result showed that the distribution by VoBBISS shows the better packet reachability than that by SF. In addition, even though in the case of the number of source APs was 4, the packets end-to-end delay time was the small values (about 1s). In the future, we plan to evaluate on multi-hop LANs with other topologies. In addition, we plan to implement VoBBISS on the communication devices.

6 Acknowledgments

This study has been supported by the Telecommunications Advancement Foundation, Japan.

7 References

[1] Akio Ogata, Hirohide Matsuzaka, Hayato Taniguchi, Masaya Nomoto, Ayami Manaka, Koichi Saito, Minoru Fukuzaki, Hiroshi Ishii, Yasuhiro Nozawa, Keisuke Utsu, "Prototype Development and Performance Evaluation of Wireless LAN Access Points for Community Information Network", IEEE TENCON 2015, PID:524, 2015

[2] 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", Proceedings of the 2015 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA' 15), pp.539-545, 2015

[3] Keisuke Utsu, Chee Onn Chow, Hiroaki Nishikawa, Hiroshi Ishii, "A Novel Information Sharing Architecture Constructed by Broadcast Based Information Sharing System (BBISS)", Proceedings of the 2014 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'14), pp.534-540, 2014

[4] Keisuke Utsu, Chee Onn Chow, Hiroshi Ishii, "A Voice Streaming Technique achieving High Reliablity and Efficiency by Broadcast Based Information Sharing System", The IEICE Transactions on Communications, Vol. J98-B, No.7, pp. 582-590, 2015 (in Japanese)

[5] Kobe City, "KOBE Free Wi-Fi," http://www.city.kobe.lg.jp/information/press/2014/07/20140704142001.html

[6] Fukuoka City, "Fukuoka City Wi-Fi," http://www.city.fukuoka.lg.jp/wi-fi/

[7] Shiro Sakata, Akira Yamada, Hiroyuki Iizuka, Tetsuya Ito, "Trend on Wireless LAN Mesh Network," The Journal of the Institute of Electronics, Information and Communication Engineers, Vol.92, No.10, pp.841-846, 2009 (in Japanese)

[8] ITU-T Recommendation G.711, "Pulse code modulation (PCM) of voice frequencies"

[9] Jorjeta G. Jetcheva, David A. Malts, "A Simple protocol for Multicast and Broadcast in Mobile Ad Hoc Networks", IETF MANET Working Group InternetDraft, <draft-ietf-manet-simple-mbcast. txt>, 2001

[10] 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

[11] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, JangPinig-Sheu, "The Broadcast Problem in a Mobile Ad Hoc Network", Wireless Networks Volume 8, Springer, pp. 153-167, Kluwer Academic Publishers, 2002

[12] J.H. James, Bing Chen, Laurie Garrison, "Imple-menting VoIP: A Voice Transmission PerformanceProgress Report", IEEE Communications Magazine, Vol. 72, Issue. 7, pp.36-41, July 2004

[13] The network simulator, OPNET Modeler (Riverbed Modeler), http://www.riverbed.com/products/steelcentral/steelcentral-riverbed-modeler.html

[14] Sayuri Wada, Hiroshi Ishii, Hiroaki Nishikawa and Keisuke Utsu, "An Optimization Study on Broadcast Based Information Sharing System (BBISS)", Proceedings of the 2014 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'14) , pp.485-489, 2014

Greedy Forwarding Prolonging the Network Lifetime Based on Two-hop Information over MANET

P. Phoummavong[1], K. Utsu[2], C. Chow[3], H. Nishikawa[4], 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] Faculty of Engineering , University of Malaya, Kuala Lumpur, Malaysia
[4] Graduate School of Systems and Information Engineering, University of Tsukuba, Ibaraki, Japan

Abstract – *Recently, there happened several disasters around the world. Taking the situation into account, we have to consider the possible type of network topology. Here, a Mobile Ad-hoc network (MANET) is very powerful role in case the infrastructure is out of order due to disaster. Among MANET routing mechanisms, location-based one is superior to the topological ones. This paper addresses the keeping or prolonging time for commutation using location-based routing between source by saving energy consumption of battery-driven nodes over MANET. For this purpose, we introduce the modification of greedy forwarding protocol based on two-hop neighbor information. This approach determines the appropriate path to guarantee that path takes low energy consumption. Moreover, this algorithm can swap the relay nodes that are located between source and destination to satisfy the minimum energy consumption and prolong network lifetime. We compare our approach with other greedy forwarding algorithm such as Most Forward with Radius (MFR), Greedy Random Selection (RS) and Greedy Nearest with Forward Progress (NFP) through computer simulation. The evaluation results show that our approach can prolong network lifetime than existing algorithms.*

Keywords: Two-hop neighbor information, Swapping the relay nodes, Virtual destination.

1 Introduction

As commonly known, A Mobile Ad-hoc Networks (MANET) [1] is an infrastructureless network that is organized by collections of self-organizing mobile nodes. Recently, the technologies of MANETs have been widely used to support various organizations, including industry, and especially emergency services. Let us consider the MANETs are deployed in disaster area. In this case, a topology-based routing protocol which needs IP address of each node is not desirable to be adopted because IP address assignment is difficult without a central server through access points and gate way. STo respond this problem, we introduce a location-based routing protocol that can operate in a disaster area without specific IP addresses. Normally, a location-based routing protocol is designed for MANET by using the global positioning system (GPS), and location services instead of IP-addresses. When the mobile nodes are deployed in disaster area with limited battery, they should not waste much energy for activities such as data transmission from source (*S*) to destination (*D*). Obviously, the network lifetime is affected by how batteries of nodes are consumed. Hence, we observe the relay nodes and swap them with other nodes to send the data packet as long as possible by saving the batteries of nodes. Before discussing our proposed algorithm, let us describe existing approaches and their critical problems. The approach described in [2] is a typical geographical unicast routing, usually called Greedy forwarding (GF). From the literature review, we can classify improved forwarding methods into three types according to next hop node selection criteria to save more energy of mobile nodes considering how much they can save more energy than the existing GF algorithms. See Fig.1.

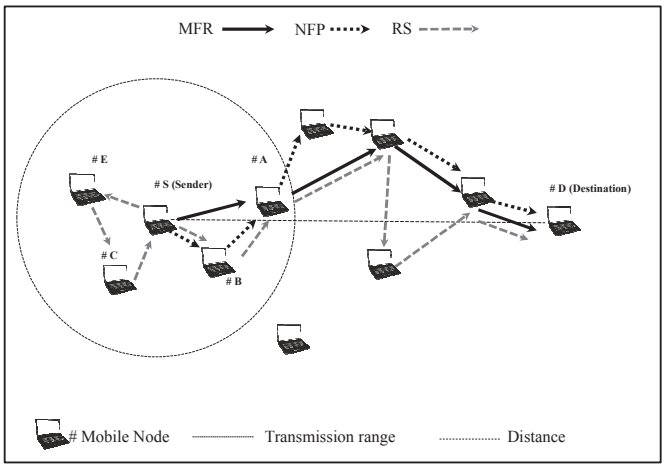


Fig. 1. The selections criteria for greedy forwarding

The first type is: 1) Greedy Nearest with Forward Progress (NFP) [3]. A next hop is chosen as the nearest neighbor with positive progress. In Fig. 1, node B is chosen. This idea tries to reduce the energy by adjusting transmission power according to the distance of two nodes and also reduce the probability of packet collision. However, this method cannot prolong the network lifetime because of redundancy of

the relay nodes. The second type is: 2) Greedy Random Selection (RS) [4]. This method uses random selection criterion. One node from the set of the senders with positive progress is randomly selected via probability properties. In Fig. 1, a node from the set of neighbor nodes of *S* (A, B, C, and E) is randomly chosen. This method tries to trade off progress and transmission reliability performance and the transmission is adjusted to reduce the consumption of energy. However, RS may give a very bad path by selecting unsuitable next relay nodes because a forwarder randomly selects the next relay node. Moreover, since RS may select redundant relay nodes, this algorithm cannot save the energy and prolong network life time. The last type is: 3) Greedy Most Forward with Radius (MFR) [5]. This algorithm processes the baseline as the orthogonal projection between *S* and *D*. A neighbor is in forward direction if the progress is positive in Fig. 1, node A is chosen. This criterion tries to minimize the number of hops while a packet has to traverse in order to reach *D* and is related to performance such as packet delay as well. However, this algorithm cannot optimize the energy consumption because the same relay nodes always work hard when same pair of *S* and *D* are attempted.

To solve the problems of these three existing location-based routing protocols (NFP, RS and MFR), we propose Greedy Forwarding with swapping the relay nodes based on two-hop information. By swapping the choice of forwarder, our proposals can reduce the energy consumption. The rest of this paper is organized as follows. Section 2 shows the evaluation criteria; next, section 3 proposes Greedy Forwarding with swapping the relay nodes based on two-hop information and details of its algorithm. Section 4 presents the performance and results. Section 5 concludes this paper.

2 Evaluation Criteria

In this section, we show the assumption and criteria to be used in our proposal. Apparently, the energy consumption is the major concern for the MANETs because the mobile nodes are driven by the internal batteries which are mainly consumed to send, receive and relay the packets. Firstly, nodes are assumed to be deployed in the static network topology in free space. We consider actions occur in discrete time sequence, "t_n" ($n=1,2,\ldots$). In other words, all the packets are assumed to be generated, sent, or received in the network at time "t_n". And the initial energy is given for each node i (where i =1, 2…N, N is the total number of nodes in the network). To calculate energy consumption, the energy functions (for non-chargeable device) are very important factors. The non-chargeable battery (cost energy consumption) for each node can be summarized by modifying functions given in [6] as equations below.

$$\partial_H(t_n) = \left(\sum_{\lambda=0}^{\lambda=H_i(t_n)} C_P(\lambda) + \mathrm{C}_T(\lambda) \right) \quad (1)$$

$$\partial_R(t_n) = \left(\sum_{\lambda=0}^{\lambda=R_i(t_n)} C_P(\lambda) + \mathrm{C}_R(\lambda) \right) \quad (2)$$

$$\partial_T(t_n) = \left(\sum_{\lambda=0}^{\lambda=T_i(t_n)} C_P(\lambda) + \mathrm{C}_T(\lambda) + \mathrm{C}_R(\lambda) \right) \quad (3)$$

$$\beta_i(t_n) = \beta_i(t_{n-1}) - \left(\partial_H(t_n) + \partial_R(t_n) + \partial_T(t_n) \right) \quad (4)$$

In eq.(1), the $\partial_H(t_n)$ represents the cost of energy consumption when a mobile node broadcasts hello messages. $\partial_R(t_n)$ refers to the cost of energy consumption when mobile node receives hello messages as shown in eq.(2). $\partial_T(t_n)$ in eq.(3) represents the cost of energy consumption when mobile node relays the data packets. $H_i(t_n)$ is the number of Hello packets that are generated by node i at time t_n; $R_i(t_n)$ is the number of Hello packets that are received by node i. C_p represents the processing power cost of the packet λ, C_T represents the transmitting power cost of the packet λ and C_R introduces the receiving power cost when a mobile node receives the packet λ. In eq.(4), $\beta_i(t_n)$ is the remaining energy of mobile node i when it broadcasts and receives Hello messages and relays data at time t_n.

3 Greedy Forwarding with Swapping the Relay node Based on Two-hop information

3.1 Exchange Hello message

Sharing the information on two-hop neighbors: Before entering the main algorithm, we propose how to get two-hop neighbor information by referring to the literature.

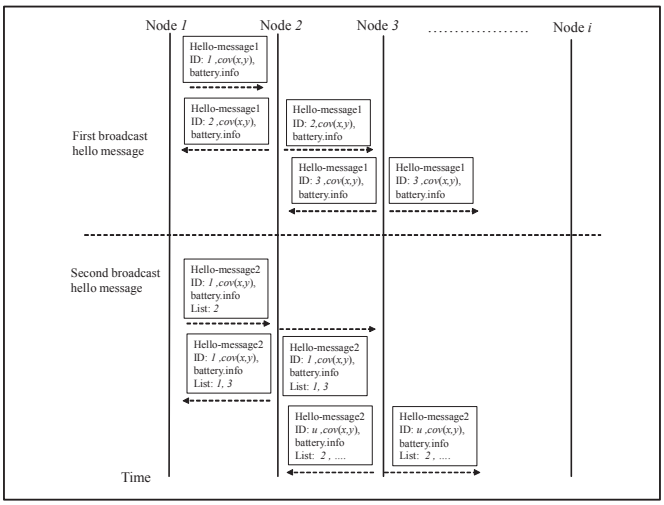


Fig. 2. The selections criteria for greedy forwarding

The hello message based on two-hop information is defined in [7]; the author shows how to optimize the number of forwarders by using two-hop information. Importantly, the sender collects two-hop neighbor information and so it has more information than just collecting one-hop neighbor information before relaying the data packet. Unfortunately, this algorithm is implemented only for a wireless sensor network and the energy consumption is not considered. According to expediency of two-hop information, we propose the way of route selection when the packet is sent from *S* to *D*. Each node exchanges the hello messages by broadcasting with their one-hop neighbors. The initial hello message contains the location and battery information of the node. After receiving the first hello message, every node knows the location and battery information of its one-hop neighbors. Next, every node broadcasts the second hello message, which contains the information on its one-hop neighbors. After that, every node knows the location and battery information on its two-hop neighbors as well as one-hop neighbors in Fig.2. We call the status after collecting two-hop neighbor information "the normal mode". In the normal mode, each node periodically broadcasts the hello message to its one-hop neighbors.

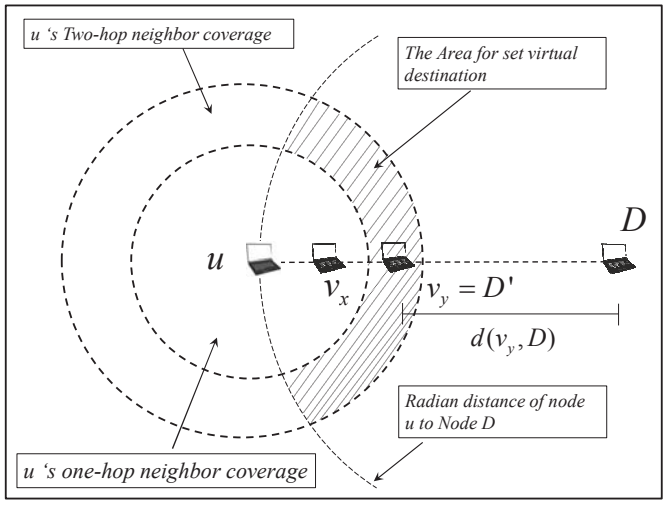


Fig. 3. Set virtual destination

3.2 Virtual Destination and inclusive node

Virtual destination and inclusive nodes: Before sending data packets, all the nodes in the network have same remaining battery capacities. We assume communication is made between node *u* and *D*. The node *u* needs to find appropriate next forwarder (relay node) to relay a packet to node *D*. Before *u* selects the next relay node, we assume the node u must set a virtual destination as (*D'*), which is a two-hop neighbor that is closest to real destination (*D*) among all the two-hop neighbors. Fig.3 shows the area where two-hop neighbor nodes of node *u* exist that may be set to the virtual destination node. The area is shown as dark area in Fig.3. We define node v_x ($x \in \{1, 2, 3,\ldots, n(u)\}$) as the neighbor of one-hop of node *u* and node v_y ($y \in \{1, 2, 3,\ldots, n(v_x)\}$) as the one-hop neighbor of v_x and the two-hop neighbor of *u*. Here, the notation "*n*(*foo*)" means the number of neighbors of node *foo*. In Fig.3, the node v_y will become a virtual destination (*D'*) if it is closest to *D* among all the two-hop neighbors. In this case, the node v_x is called an "inclusive node" which is the one-hop neighbor of both nodes *u* and *D'*. On the other hand, after sending a data packet, the relay node will consume battery. Therefore, we have to consider the remaining battery capacity of relay nodes to select the route with longer lifetime.

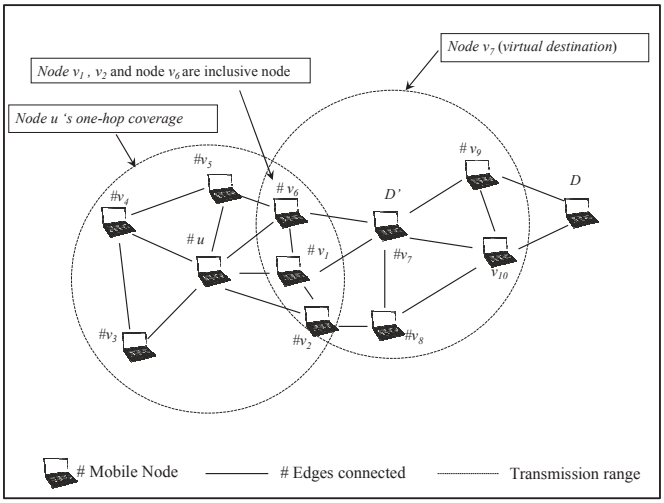


Fig. 4. Example set virtual destination

Here, we assume the node *u* can set the virtual destination by calculating the probability about the average distance of node v_y ($y \in \{1, 2, 3,\ldots, n(v_x)\}$) to *D* and the probability about the maximum remaining battery capacity of node v_y or *D*'. At first, we find the total average distance from all the virtual destination candidates, i.e., node v_y ($y \in \{1, 2, 3,\ldots, n(v_x)\}$) to *D* by eq.(5); where d_T is the total average distance and $d(v_y,D)$ is the distance between node v_y and *D*. Then we calculate the probability by eq.(6); where P_{avg} is the probability of average distance. If node v_y is closest to *D*, it will have the lowest probability. In the second phase, we calculate the total of maximum battery capacity of node v_y by eq. (7); where β_T is the sum of remaining battery capacities of all the nodes v_y ($y \in \{1, 2, 3,\ldots, n(v_x)\}$) and β_{vy} is the battery capacity of a specific node v_y at time t_n. Then, we can find the probability of maximum battery capacity by eq. (8). If a node v_y has the maximum remaining battery capacity, it will have the highest probability. Finally, from eq. (6) and eq. (8), we get a probability that is a combination of the probabilities of distance and the probability regarding v_y , which is a neighbor node of v_x and two-hop neighbor of the node *u*. Here, $p_{avg}(v_y)$ and $p_\beta(v_y)$ are considered mutually independent since the average distance of node v_y and their battery's capacity are independent event each other. Therefore, we can calculate the joint probability in eq. (9). Fig. 4 shows an example: node v_7 is closer than node v_8 therefore the sender node *u* sets node v_7 as virtual destination or *D'*. Let's define node v_1 , v_2 and v_6

become to inclusive nodes of sender u and D'. The node u has the battery information of nodes v_1 , v_2 and v_6 that are located in the intersection coverage of sender u and D'. The neighbor nodes to be considered are inclusive nodes between node u and D'. Hence, forwarder node u that can make the decision on which of v_1 or v_6 is selected by classifying the battery information of node v_1 and v_6.

$$d_t = \sum_{y=1}^{n(v_x)} d(v_y, D) \tag{5}$$

$$p_{avg}(v_y) = \frac{d(v_y, D)}{d_T} \tag{6}$$

$$\beta_T = \sum_{y=1}^{n(v_x)} \beta_{v_y}(t_n) \tag{7}$$

$$p_\beta(v_y) = \frac{\beta_{v_y}(t_n)}{\beta_T} \tag{8}$$

$$Pr(v_y) = (1 - p_{avg}(v_y)) p_\beta(v_y) \tag{9}$$

3.3 Algorithm swapping the relay node

We assume the data packets are sent by source to destination through the relay nodes. The processing of the algorithm is shown in Table 1. In the line 1, the sender node checks the location of destination and then if it finds the destination is its neighbor, the operation will be terminated in the line 2. If not, the sender must select a virtual destination. It will sort a list of candidates of virtual destinations in the ascending order about the distance to real destination in line 3. Then, the sender checks the battery capacity of candidates of the virtual destination before choosing the virtual destination in line 4. If the closest virtual destination candidate has lower battery capacity, then the sender will change the virtual destination by considering $Pr(v_y)$. However, if the battery capacity of the closest virtual destination is still maximum, then sender will select it as shown in the line 5. Next, the line 6 shows that the sender always compares the battery capacity of inclusive nodes before selecting the relay node. The sender will select the next relay node considering which node has maximum battery capacity by referring to the value of $\beta_i(t)$in line 7. In line 8, simulation process is repeated and the next relay node will become the sender node and go back to line 1 until the processing is terminated or data packet is received by destination.

Table 1

Algorithm's swapping the relay node

1: Check location of node (D)
2: Check (D) is covered or not
If not, **then** (**go to 3**)
If (D) is covered, **then**
Select (D) End **(Algorithm is terminated)**
3: Sort list of candidates of virtual destinations in the ascending order about the distance to the real destination.
4: Check battery of candidates of virtual destination by considering $Pr(v_y)$
5: Then set virtual destination which has maximum $Pr(v_y)$
6: Find inclusive nodes
Then find a node v_i which has maximum remaining battery $\beta_i(t)$
7: Select (v_i) as the next relay node
8: Change relay node u to v_i, **go to 1** (repeat)

4 Performance and results

4.1 Parameter and network topologies

We evaluate our proposal by comparing with existing methods NFP, RS and MFR using computer simulation based on MATLAB [8], [9]. The simulation parameters and value setting are shown in Table2. Here, we assume the network is static (nodes do not move). For each topology of each simulation scenario, we assume a position for S is located on the left most and the position D is located on the right most, position of the network as shown in an example of Fig.5.

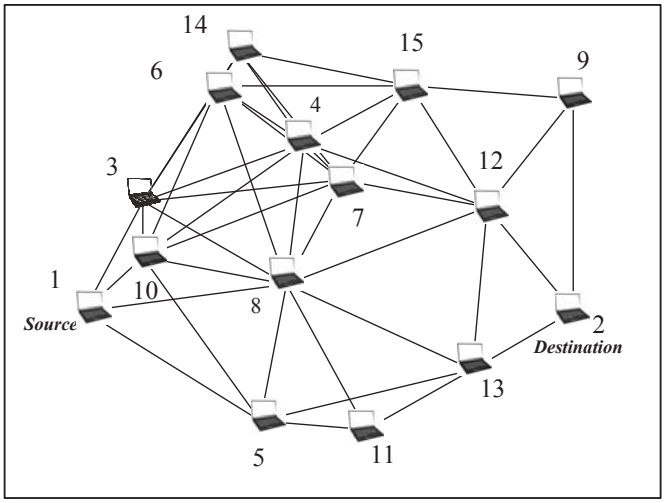


Fig. 5. An example of network topology (300 x 300 m^2)

Table 2

Symbol	Description	Value
t_n - t_{n-1}	The time interval	1s
$\beta_i(0)$	The initial battery capacity for node *i* at time 0	80000 joule
C_P	The processing power cost	0.1 joule
C_T	The transmitting power cost	0.8 joule
C_R	The receiving power cost	0.3 joule
-	The number of mobile nodes	15 nodes
-	Transmission range	150 m
-	Exchange hello message	1 packet /s
-	The number of data packet	1 packet /s
-	Simulation trials	100 topologies
-	Simulation time	1000s

4.2 The simulation results

As for the example 1 in Fig.5, the result is given in Fig.6 and Fig.7. Fig.6 shows the total remaining energy and Fig.7 shows the remaining lifetime per mobile node. Those two values are the metric for simulation. The Fig.6 shows the total remaining energy, comparing the proposed method with MFR, RS and NFP.

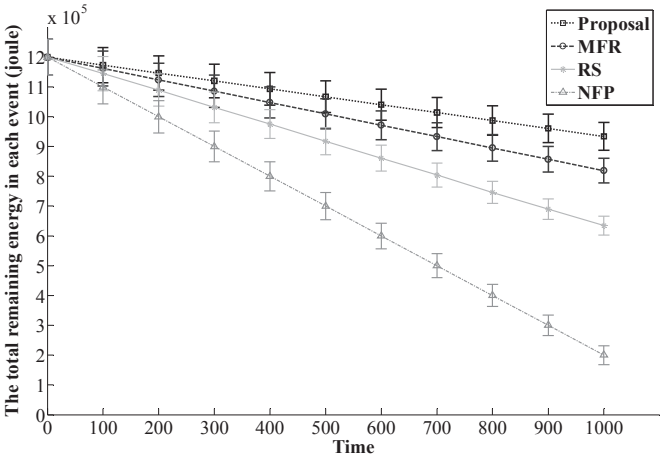


Fig. 6. The total remaining energy in each event for an example

Through four simulation's outputs, NFP consumes the energy much more than any other methods. Since NFP creates the redundant relay nodes when sending the data packet, this method could not guarantee the network lifetime. RS is better than NFP because it uses random selection criterion. One node from the set of the forwarders with positive progress is randomly selected via probability; RS doesn't create the redundancy of relay nodes. However, as the problem shown in section I, RS could not reduce the energy consumption because it gives a very bad path by selecting an inappropriate forwarder which relays the data packet to *D*. Therefore, RS is not best method for saving energy. The other method is MFR, this algorithm uses baseline as the orthogonal projection between *S* and *D*. A neighbor is selected in forward direction to *D* if the progress is positive to minimize the number of hops. However, the result shows MFR could not optimize the energy consumption compared with our proposal because the same relay nodes always relay the data when the same pair of source and destination is attempted. On the other hand, our proposal can guarantee the minimum energy consumption. Total energy is slowly decreasing to the final time slot (t_{1000}) and the total remaining energy is higher than any other existing algorithms.

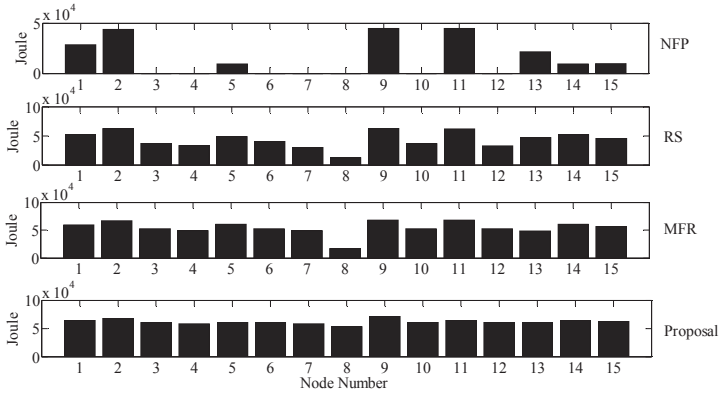


Fig. 7. The remaing lifetime per mobile node for an example

The next reults is the remaning liftetime per mobile node as shown in Fig.7. This is the comparison of our proposalwith MFR, RS and NFP for 15 nodes topology as Fig.5. The result shows our proposal is longer than other three methods. On average, NFP, RS and MFR cannot guarantee a long lifetime of the network. Here, NFP is the worst because many nodes consume much more battery and especially the nodes 3, 4, 6, 7, 8 10, 11 and 12 run out the battery. Hence, choosing the nearest neighbor with positive progress as a next hop is not efficient for saving energy. RS is better than NFP because the battery lifetime per mobile node of RS did not run out the battery like NFP. As shown in the section I, RS could not guarantee the energy consumption. In average, RS is lower than MFR and our proposal. MFR is better than RS because this criterion can minimize the number of hops. However, in this algorithm some relay nodes hardly work and the nodes consume much energy, for example in Fig.7, we can see that node 8 , 4 and node 10 consumes much energy than other nodes.

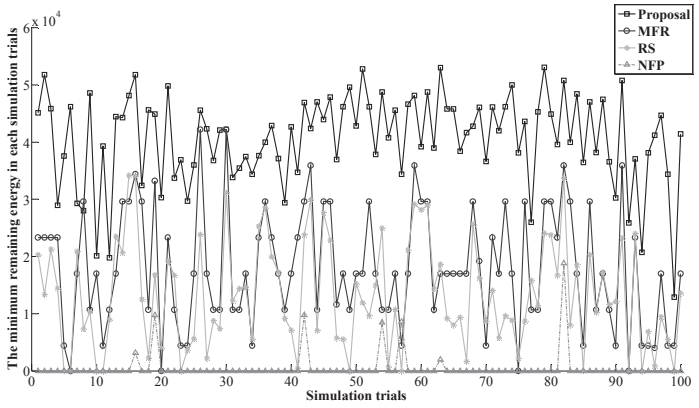


Fig. 8. The minimum remaining engergy of node per each simulation

In Fig. 8 illustrates the measurement of the minimum remaining life node per 100 simulation trials. In average point views, the result shows that the minimum remaining life nodes of our proposal is higher than existing methods. As we showed the problem on above, MFR, RS and NFP cannot guarantee to save the energy. In average, NFP is the worst because the minimum reaming node always runs out of battery. Moreover, NFP is choosing the nearest neighbor as a next hop that is also not efficiency for saving energy. Next method, RS is better than NFP because the battery lifetime per mobile node of RS did not run out the battery like NFP. However, RS could also not guarantee the energy consumption. Same as last result, MFR is better than RS because this criterion can minimize the number of hops. However, in this algorithm, some relay nodes always work hard and the nodes consume much energy.

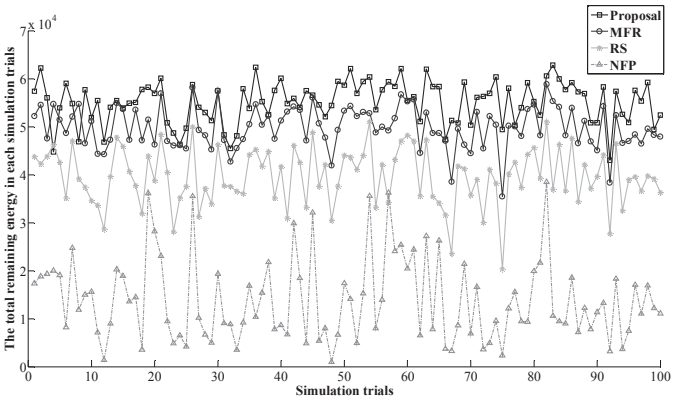


Fig. 9. The remaing lifetime per mobile node for an example

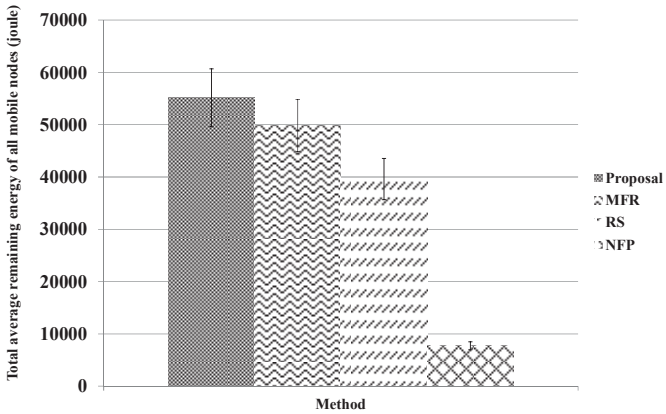


Fig. 10. The total remaining lifetime per mobile node for each method

To make sure for the investigation, we made the measurement of total average remaining energy for 15 nodes and 100 simulation trials as illustrated in Fig. 9 and Fig. 10. The result shows that the average lifetime of our proposal is longer or higher than existing methods. As estimation and prediction in section I, MFR, RS and NFP cannot guarantee the long lifetime of the network. Most important thing is that our proposal can increase lifetime of the network by 10.5 %, 28.6% and 82.2% compared with MFR, RS and NFP, respectively.

5 Conclusion

In this paper, we address the problem of energy reduction in a mobile ad hoc network. Reducing energy consumption and increasing network lifetimes are very difficult in existing location-aided routing algorithms. To overcome these problems, we proposed Greedy Forwarding by swapping the relay nodes based on two-hop information about an ad hoc network inspired by geographic routing. Simulation outputs showed how our proposal can optimize the energy consumption and improve several aspects of performance with time interval and network topologies. Specifically, our proposal has the best and stable performance in terms of increasing lifetime compared with existing approaches. A future study is needed for more detailed evaluations, including loss path energy consumption and mobility.

6 Acknowledgment

This work is partly supported by JSPS KAKENHI Grant Number 26420372.

7 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] I. Stojimenvic, "position-Base Routing in Ad-hoc Network," IEEE Communication Magazine, Vol 40, pp. 128-134, July 2002.

[3] G. G. Finn, "Routing and Addressing Problem in Large Metropolitan-Scale Internet work," ISI research report, ISU/RR-87-180, March 1987.

[4] M. Zorzi and R. R. Rao, "Geographic Random Forwarding (GeRaF) for Ad hoc and sensor Network: Multihop Performance," IEEE Transaction on Mobile Computing, Vol. 2, No. 4, 2003.

[5] T. Hou and V. Li, "Transmission Range Control in Multihop Packet Radio Network," IEEE Transaction on Communication, Vol. 34, No. 1, pp 38-44, January 1986.

[6] N. Paris, B. K. C etin, N. Pratas, F. J. Velez, N. R. Prasad, and R. Prasad, "Cost Benefit Aware Routing Protocol for Wireless Sensor Network with Hybrid Energy Storage System," Journal of Green Engineering, pp 189-204, 2011.

[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] F. Paul and H. Uwe, "The Mathematics of Networks of Linear Systems", Presents a systematic study that develops the functional model approach for multivariable linear systems. ISBN: 978-3-319-16646-9, Springer International Publishing, 2015.

[9] R. Bradley, "Programming for Engineers", A Foundational Approach to Learning C and Matlab. ISBN: 978-3-642-23303-6, Springer Berlin Heidelberg, 2011.

An Experimental Study on Round Trip Time Distribution of the Internet

Akira Sasatani and Hiroshi Ishii
Graduate School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan

Abstract – *The end-to-end delay is one of the important service indicators of the Internet applications. Knowing the characteristics of the end-to-end delay seems very significant to provide users better net-related services. This paper measures end-to-end delay between a fixed site in Japan and several sites outside Japan periodically and clarifies actual delay distributions in shape of histograms. Then, it compares each measured histogram and normal distribution with the same mean and standard deviation as those of histogram and shows most histograms are normally distributed.*

Keywords: End-to-end delay, ping, normal distribution, correlation factor

1 Introduction

People are nowadays able to access information existing all over the world quickly and simply through the internet which spread as infrastructure. The packet transfer time between the service requiring side and offering one is a main service performance ruling factor in the internet. Therefore to get some knowledge about the distribution of the end-to-end delay of this network is necessary to design and achieve a service.

End-to-end delay should be measured as one way delay to a destination from an original sender. However, it is difficult to know one way delay conveniently because there have to install measuring equipment in both of sending side and receiving side and have to be operated synchronously. Therefore it is general that only one side measures the round trip delay instead of one-way end-to-end delay. This round trip delay is called RTT (Round Trip Time). An existing study [1] shows RTT obeys Gamma distribution and another [2] explains that RTT follows Pareto distribution when a network is congested and does lognormal distribution when a network is not in congestion. We however think it necessary to re-examine RTT distribution because those existing study results are obtained in the period when high speed access circuits did not exist and the network traffic was much smaller.

In [3], we have already measured RTT using ping command to some countries and studied resemblances between the measured results and lognormal distribution according to the result of [2] but failed to show the resemblance adequately. The reason of failing getting the resemblance seemed lack of consideration of outliers of measured values.

Based on the estimation in [3], this paper revisits the issue and studies resemblances between this actually measured results and a normal distribution by considering the outliers of measured values using IQR (Interquartile range).

Section 2 describes measuring method and conditions. In section 3, we show experimental results. Section 4 studies resemblances between the measured results after excluding outliers and a normal distribution. Section 5 concludes this paper.

2 Measuring method and conditions

2.1 Measuring method of RTT

We measure RTT using ping command. Ping commands are periodically sent from a PC in our laboratory by use of "ExPing" [5] and the results are logged.

2.2 Measuring specification of RTT

2.2.1 Measurement cycle

10 min. (6161 times in total)

2.2.2 Measurement period

April 29, 2015 at 06:03 p.m. ～ June 18, 2015 at 01:42 p.m. (51 days).

2.2.3 Measurement target

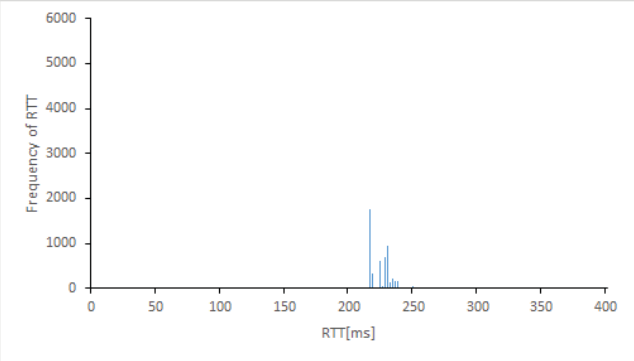
USA free web server (000webhost.com), Germany free web server (Host-ed.net), University of Vienna, Higher University of San Andrés, The University of Sydney, University of Nairobi, Harvard University, The University of British Columbia and Tunghai University.

2.2.4 Measurement place

Ishii laboratory, Tokai University, Minato, Tokyo, Japan.

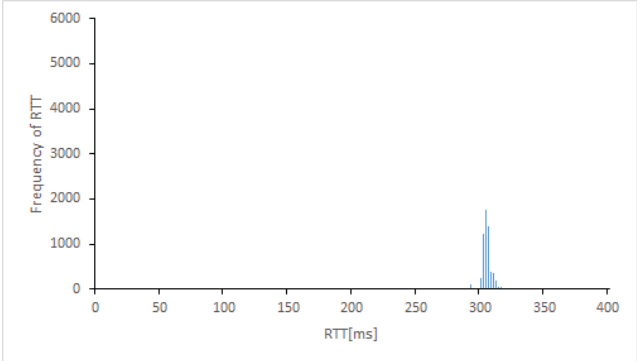
3 Experimental results

The results of measuring experiment are shown as histograms in Figures 1 through 9. The horizontal axis shows the measured RTT and the vertical axis shows the frequency of RTT per 2ms time slot. The average and the standard deviation of all the RTT measured for each destination is also shown in each figure.



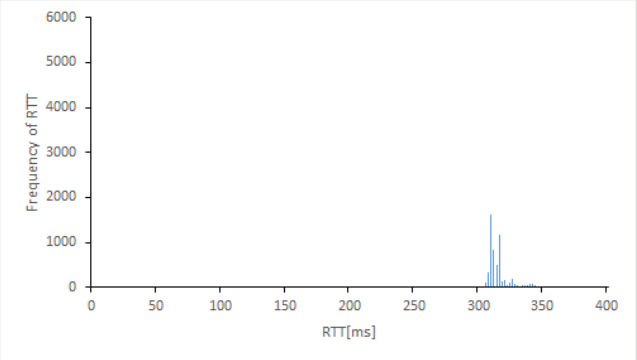
(Average：225.250 Standard deviation：13.721)

Fig. 1. RTT histogram of the U.S.



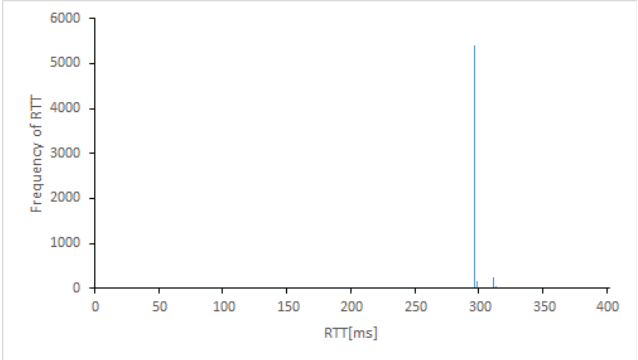
(Average：306.889 Standard deviation：14.642)

Fig. 4. RTT histogram of Univ. of San Andres



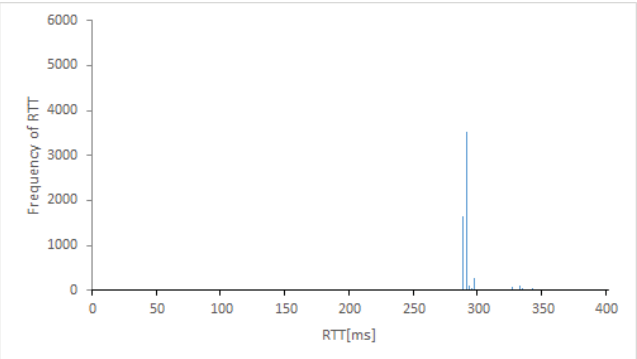
(Average：318.852 Standard deviation：17.961)

Fig. 2. RTT histogram of Germany



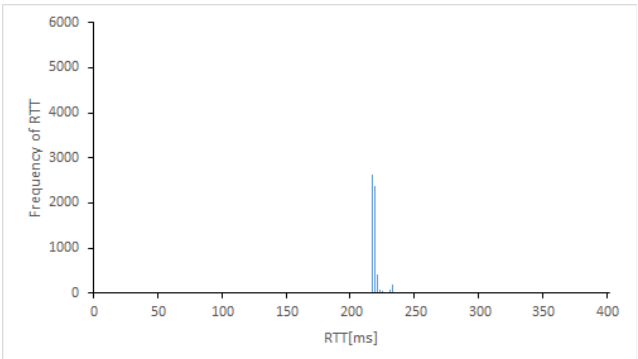
(Average：297.372 Standard deviation：11.428)

Fig. 5. RTT histogram of the Univ. of Sydney



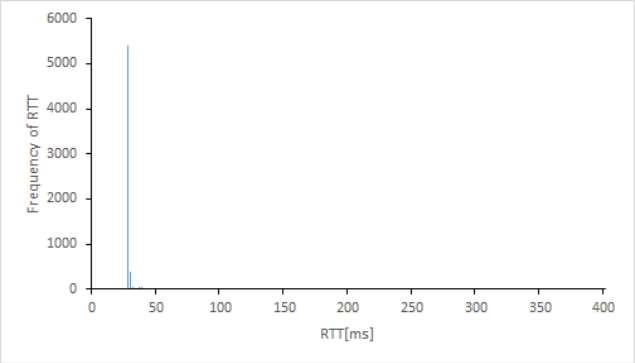
(Average：292.753 Standard deviation：13.659)

Fig. 3. RTT histogram of Univ. of Vienna



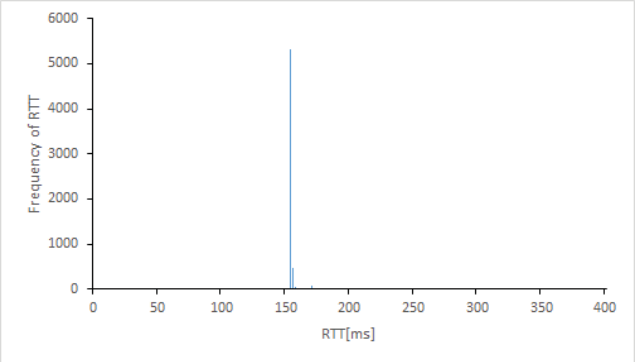
(Average：220.097 Standard deviation：15.326)

Fig. 6. RTT histogram of Univ. of Nairobi



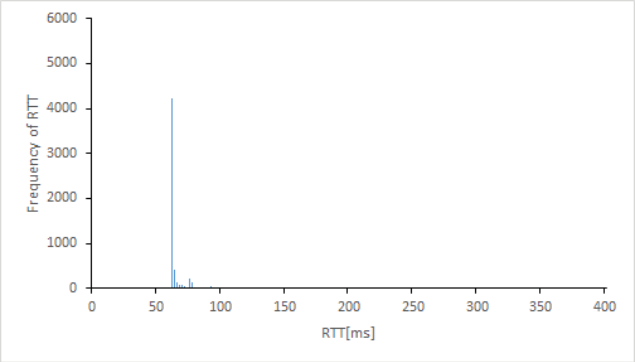
(Average：28.617 Standard deviation：8.399)

Fig. 7. RTT histogram of Harvard Univ.



(Average：155.364 Standard deviation：10.384)

Fig. 8. RTT histogram of the Univ. of British Columbia



(Average：71.657 Standard deviation：28.845)

Fig. 9. RTT histogram of Tunghai Univ.

4 Consideration

4.1 A distribution of RTT and normal distribution

The shape of the RTT histograms seems similar to that of the normal distribution from Fig.1 through 9. Then we research resemblance by use of correlation coefficient between each measured RTT histogram and the normal distribution with the same mean and variance as those of each histogram. The results are shown in Table 1.

Table 1. Correlation coefficient between measured results and normal distribution with same average and variance

	Correlation coefficient
the U.S.	0.566
Germany	0.559
Vienna	0.397
San Andres	0.590
Sydney	0.330
Nairobi	0.426
Harvard	0.389
B.C.	0.358
Tunghai	0.232

As shown in the Table1, the correlation coefficient of any pair of sample and normal distribution is rather small (less than 0.6). Hence, it is difficult to conclude that the normal distribution gives reasonable approximation to the measured RTT histograms. This result is also shown in [3].

4.2 Excluding outliers

Section 4.1 uses the average and standard deviation calculated from all samples. But there are samples whose values are extremely deviated from the average. They are called "outliers". An outlier is a value in a statistical samples which does not fit a pattern that describes most other data points. More specifically, a value that lies r* IQR beyond the upper or lower quartile. Hence, we exclude outliers using quantiles because outliers may affect estimation of the distribution. First, each RTT data is rearranged in an ascending order of the value. Next we find the values bottom 25th percentile (first quartile) Q1 and bottom 75th (top 25th) percentile (third quartile) Q3. Then, according to the following formula, we get the boundary of outliers, O_L and O_U.

$$Outlier\,(Lower)\ \ O_L = Q1 - r \cdot IQR$$
$$Outlier\,(Upper)\ \ O_U = Q3 + r \cdot IQR$$
$$IQR = Q3 - Q1$$

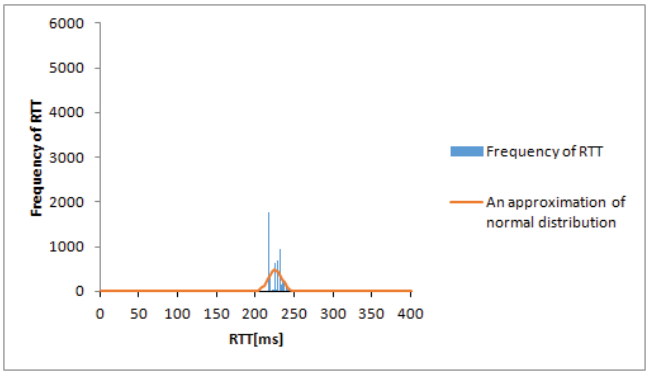
We researched data between O_L and O_U.

Although generally "r" is 1.5, we adopt r=3 to exclude extreme outliers only.

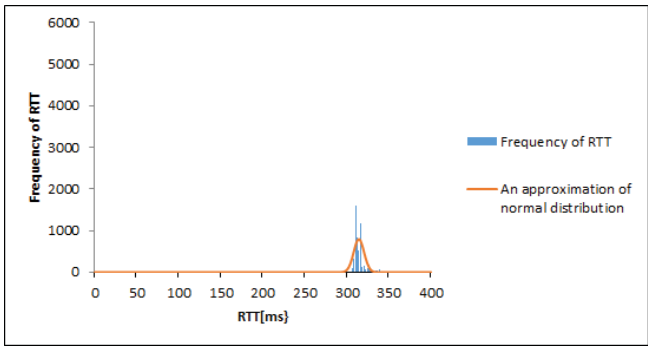
Table 2. Excluding outliers before and after
Average and normal distribution, Rate (Excluding outliers / All data)

		the U.S.	Germany	Vienna	San Andres	Sydney	Nairobi	Harvard	B.C.	Tunghai
All data	Ave.[ms]	225.250	318.852	292.753	306.889	297.372	220.097	28.617	155.364	71.657
	Std. dev.	13.720	17.961	13.659	14.642	11.428	15.326	8.399	10.384	28.845
Excluding outliers	Ave.[ms]	224.032	314.097	288.794	304.251	295.225	216.794	27.308	153.786	62.082
	Std. dev.	8.020	6.163	0.671	3.604	0.535	1.447	0.690	0.673	1.843
Rate		0.990	0.897	0.858	0.944	0.906	0.900	0.947	0.945	0.812

Table2 shows an average and a standard deviation of all measured data and those of data after excluding outliers. Also it shows the rate of number of sample after excluding outliers to that of all elements measured. It shows that the average is unchanged independent of outlier exclusion but the standard deviation becomes much smaller after excluding extreme outliers. Normal distribution with an average and a standard deviation calculated from the data without extreme outliers and measured histogram are shown for each country in figures 10 through 18. In addition, correlation coefficients are summarized in Table3.

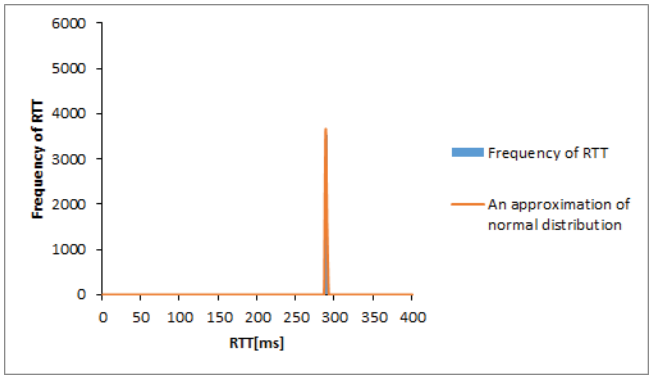


(Correlation coefficient: 0.597)
Fig. 10. RTT histogram and outliers
excluded normal distribution (the U.S.)

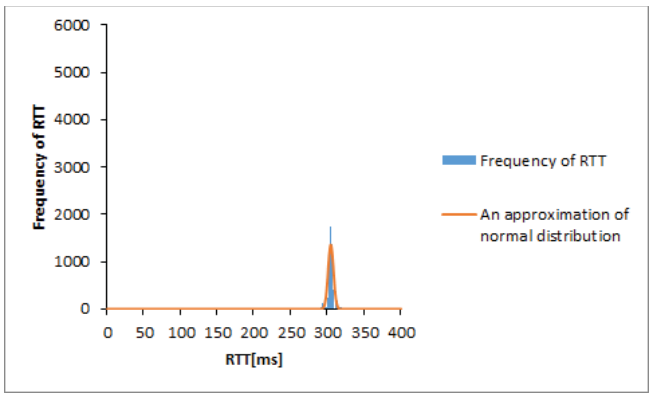


(Correlation coefficient: 0.797)

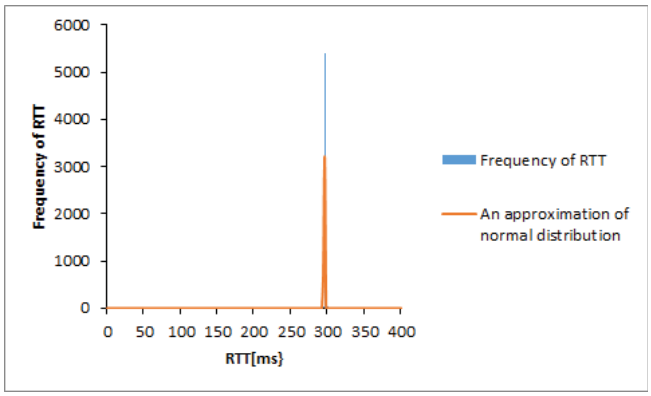
Fig. 11. RTT histogram and outliers
excluded normal distribution (Germany)



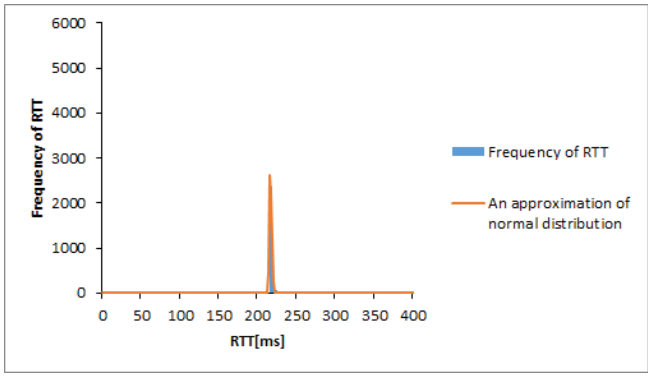
(Correlation coefficient: 0.726)
Fig. 12. RTT histogram and outliers
excluded normal distribution (Univ. of Vienna)



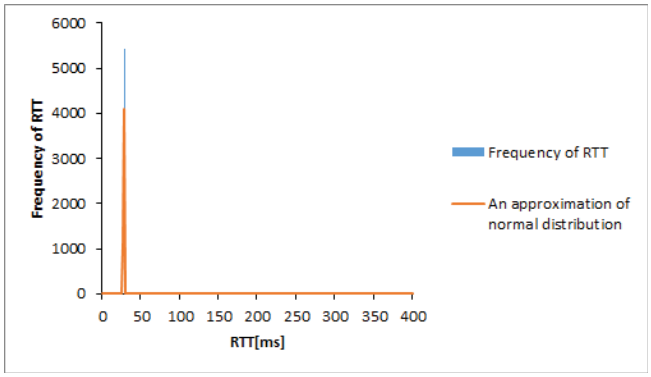
(Correlation coefficient: 0.951)
Fig. 13. RTT histogram and outliers
excluded normal distribution (Univ. of San Andres)



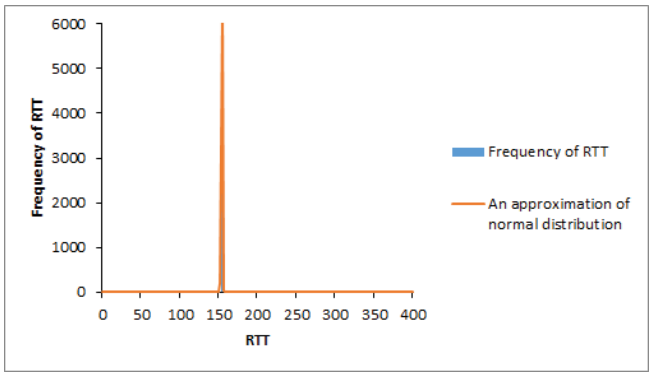
(Correlation coefficient: 0.979)
Fig. 14. RTT histogram and outliers
excluded normal distribution (Univ. of Sydney)



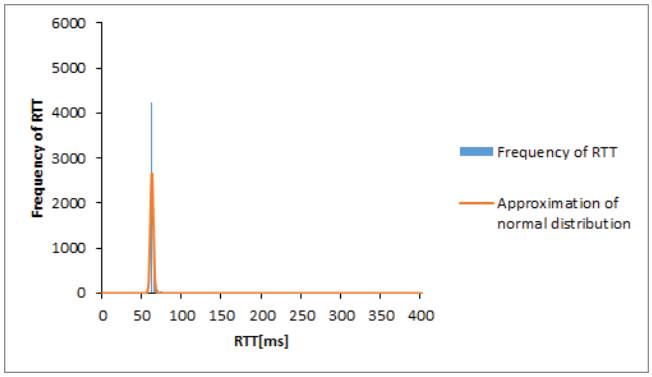
(Correlation coefficient: 0.988)
Fig. 15. RTT histogram and outliers
excluded normal distribution (Univ. of Nairobi)



(Correlation coefficient: 0.962)
Fig. 16. RTT histogram and outliers
excluded normal distribution (Harvard Univ.)



(Correlation coefficient: 0.996)
Fig. 17. RTT histogram and outliers
excluded normal distribution
(the Univ. of British Columbia)



(Correlation coefficient: 0.824)
Fig. 18. RTT histogram and outliers
excluded normal distribution
(Tunghai Univ.)

Table 3. Correlation coefficient between measured histograms without outliers and normal distribution

	Correlation coefficient
the U.S.	0.597
Germany	0.797
Vienna	0.726
San Andres	0.951
Sydney	0.979
Nairobi	0.988
Harvard	0.962
B.C.	0.996
Tunghai	0.824

According to the above Figures 20 through 28 and Table3, most RTT data rather accord with normal distribution owing to excluding outliers. Nevertheless in USA server, even if outliers were excluded, correlation coefficient between measured histogram and the normal distribution is small. In the following, we study this phenomenon.

4.3 Composite distribution

The RTT distribution of the USA server has several peaks (Fig.1). Hence we study whether it can be expressed with the composition of several distributions or not. According to studies until section 4.2, the average almost coincides with the mode in histograms approximated with just a single distribution. Let the delay of the maximum frequency and delay of the 2nd largest frequency be d_1 and d_2, respectively and so forth. We assume that samples in the neighborhood of peak *i* belongs to a normal distribution *i* with the average delay, di, group all the samples into several groups belonging to the different normal distribution and then each standard deviation is estimated.

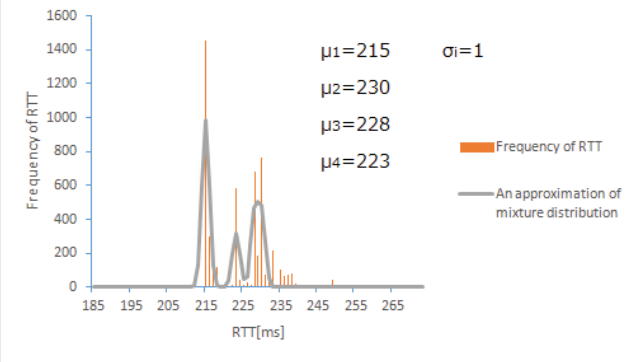
Here let the composite distribution dist(x), then;

$$dist(x) = \sum_i a_i \cdot NORM(x, \mu_i, \sigma_i)$$

$$where \sum_i a_i = 1 \quad \mu_i = d_i$$

$$a_i \ is \ roughly \ estimated \ \ from \ the \ measurements$$

Here we choose 4 values from the highest frequency (i =1, 2, 3, 4) and get a composite distribution. Fig.19 shows measured histogram and the composite distribution. Moreover we get a correlation coefficient of them.



(Correlation coefficient: 0.807)

Fig. 19. RTT histogram and Composite distribution (the U.S.)

It can be said that the shape of measured histogram and one of the composite distribution are similar. We get a high correlation coefficient (0.807). Accordingly the histogram can be approximated as a composite distribution that consists of several normal distributions.

5 Conclusions

This paper has tried to clarify the characteristics of RTT by continuous measurement using ping. We have sent ping commands to and logged the results of USA, Germany, University of Vienna, Higher University of San Andrés, The University of Sydney, University of Nairobi, Harvard University, The University of British Columbia and Tunghai University. As a result, we found out that the RTT has a very stable value. The distributions without extreme outliers have high correlation with the normal distribution.

Further study is needed to consider the resemblance by a distribution besides the normal distribution, quantitative study of resemblance by chi-square testing, consideration of the relation between number of hops and RTT, and more precise investigation of the composite distribution.

6 References

[1] Kaori Kobayashi, Masaru Ohta, Tsuyoshi Katayama. "Analysis and Evaluation of Packet Delay Time on Internet"; IEICE technical report, NS2001-5, pp.27-32 (2001) (in Japanese).

[2] K. Fujimoto, S. Ata and M. Murata. "Statistical Analysis of Packet Transmission Delay in the Internet"; IEICE technical report, IN2000-48, pp.41-48 (2000) (in Japanese).

[3] Sho Tohyama, Kalkattawi Suhail, Alamri Zizo, Go Gun and Hiroshi Ishii. "Analysis of end-to-end delay characteristics of the internet"; IEE-CMN, CMN-15-003, pp.11-16 (2015) (in Japanese).

[4] Akira Sasatani, Nozomu Hirata, Natsuki Okabe and Hiroshi Ishii. "A study on end-to-end delay distribution of the internet"; IEE-CMN, CMN-16-005, (2016) (in Japanese).

[5] "ExPing", http://www.woodybells.com/exping.html

Data-Driven Sensor Networking Processor Tolerating Instantaneously Excessive Load

Shuji SANNOMIYA[1], Yukikuni NISHIDA[2], Makoto IWATA[3], and Hiroaki NISHIKAWA[1]
[1]Faculty of Engineering, Information and Systems,University of Tsukuba, Ibaraki, Japan
[2]Graduate School of Systems and Information Engineering, University of Tsukuba, Ibaraki, Japan
[3]School of Information, Kochi University of Technology, Tosayamada-cho, Kochi, Japan

Abstract— *To deploy sensor networks widely, not only long life-time even with the use of batteries but also innovative reliability is indispensable because human maintenance works for individual device become no longer possible. As already proposed, ultra-low-power data-driven sensor networking systems detect and isolate failure devices autonomously for eliminating human checking works. However, the failures may still arise instantaneously huge input data and make processors inoperative. In this paper, a data-driven sensor networking processor with an autonomously variable input rate mechanism is proposed to realize tolerance against such instantaneously excessive load. The proposed processor makes it possible to accept arriving input data only when the processing load after the data input can be kept within the processing capability by utilizing the predictability of internal processing load. The circuit simulation of the proposed processor shows that the operation of the processors can be guaranteed regardless of the input data arrival timing.*

Keywords: data-driven processor, real-time multiprocessing, self-timed pipeline, sensor network, internet of things

1. Introduction

Toward Trillion-Sensor-Universe which is propounded by Janusz Bryzek and indicates the utilization of more than 100 networked sensors per person for a year on earth, sensor networking system should be innovatively reliable because maintenance cost for individual sensor device increases explosively and becomes unacceptable. Moreover, such sensor devices may be battery-operated to be free from electrical wiring and thus should be long-lifetime. To realize such reliable and long-lifetime sensor networking system, we have proposed data-driven sensor networking system based on a data-driven processor by which the battery-operated lifetime becomes foreseeable and long even when failures occur in sensor devices [1].

The long battery-operated lifetime foreseeability is provided as a result of data-driven processor architecture and its self-timed pipeline implementation in which operation execution is initiated on the arrival of input data as long as computational resources (i.e. pipeline stages) are available and thus real-time multiprocessing indispensable for networking under time constraints defined by communication protocols can be achieved without extrinsic program-execution overheads such as context switching and interrupt handling [2]. This passive and overhead-free execution manner results in not only low power consumption but also the proportional relation between the consumption current and the processing load (i.e. the amount of processing data) of the data-driven processor. Based on this proportional relation, the runtime energy dominating the battery-operated lifetime of sensor nodes can be estimated for a given typical load before the installation of the nodes.

The energy foreseeability 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 human working for battery replacement each time the battery runs down becomes no longer required. Moreover, it makes it possible to detect the failures of devices autonomously after the system installation by the energy self-check of each node because the failure occurred in a node changes the consumption current of the node or its neighbors processing the data transferred from the node, and gets the energy of the node or the neighbors out of the foreseen range. To enhance the reliability of the system, the bad effects of the nodes with failures can be suppressed within the neighboring nodes by dropping the data transferred from the nodes with failures in the neighbors, while the other nodes can operate continuously.

On the other hand, not only the failures but also unexpected situations in which a large amount of events is sensed concurrently may cause overload situation in which the amount of processing load of a node exceeds the processing capacity of the node. To make the system reliable against such situations, we have studied an overload-free data-driven networking platform architecture, in which input data is buffered or discarded when the sum of the observed load and the load to process the input data exceeds the processing capability, by utilizing the direct observability of the processing load via consumption current [3], [4]. However, the observation of the processing load involves delay due to both the signal propagation in the observation circuit and sampling interval of consumption current, and thus instantaneously sudden increase of input may cause overload and make the platform inoperative.

ISBN: 1-60132-444-8, CSREA Press ©

In this paper, a data-driven sensor networking processor with tolerance against such instantaneously excessive load is proposed. Fortunately, the self-timed pipeline has tolerance against excessive load by nature. In the self-timed pipeline, valid data is transferred between adjacent stages locally based on local negotiation between the stages, and each pipeline stage never accept input data when processing. As a result, the input rate is kept within the maximum throughput of the self-timed pipeline. To inherit this natural tolerance, a variable rate input mechanism is proposed for the circular pipeline indispensable to execute operations in the data-driven processor. The proposed mechanism keeps the processing load within the maximum throughput of the processor naturally. The effectiveness of the proposed mechanism is evaluated based on the circuit simulation.

2. Data-driven sensor networking processor

In this section, data-driven sensor networking processor realizing sensor networking systems with energy foreseeability, autonomous failure detection/isolation and long battery-operated lifetime is explained and its requirements against instantaneously excessive load are discussed.

2.1 Ultra-low-power chip multiprocessor architecture

Sensor networking systems provide a variety of services such as security, infrastructure monitoring, and disaster prevention, by networking sensors spread on sensing targets. With increasing the scale of the sensor networking systems, system maintenance cost to keep the sensor devices alive and healthy increases due to the checking of the operation of every device and the replacement or charge of discharged batteries especially for wireless devices. Therefore, the key to realize large scale networking systems is to reduce the work for operation checking and the frequency of the battery replacement/charge as low as possible.

To realize not only ultra-low-power consumption but also autonomous failure detection/isolation to eliminate both the operation checking and the frequent battery replacement/charge, a data-driven processor named ULP-CUE (Ultra-Low-Power CUE) and its chip multiprocessor version named ULP-DDCMP (Ultra-Low-Power Data-Driven Chip Multiprocessor) [5] are introduced.

Figure 1 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 operations. The circular pipeline of the ULP-CUE consists of matching memory (MM) to detect the arrival of operands for binary operations, program storage (PS) to fetch operations, functional processing unit (FP) to execute the operations and memory access (MA) to read and write data. Its circularization is realized by a

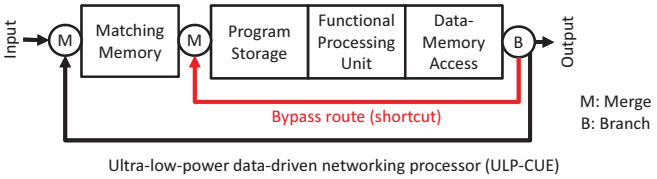


Fig. 1: Circular pipeline to realize ULP-CUE.

merge (M) stage which accepts data from two preceding stages in order of arrival and transfers the accepted data to a succeeding stage and a branch (B) stage which transfer each data to one of the succeeding stages selectively. 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 interrupt handling 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 [5].

2.2 Ultra-low-power self-timed circuit implementation

To inherit the passive and on-demand processing manner of the data-driven manner into circuit implementation, the whole stages of the ULP-DDCMP are realized by using ultra-low-power self-timed elastic pipeline (ULP-STP) [6]. In the ULP-STP, only pipeline stages with valid data are driven exclusively as a consequence of the localized data transfer called handshake. Figure 2 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 [7]. Based on the four-phased handshake, the valid data in the STP are transferred between adjacent stages, as follows.

- (0) Reset: After the assertion of the reset signal, the C negates both its send signal representing transfer request and ack signal representing acknowledge.
- (1) The C asserts its ack signal after its send signal is asserted.
- (2) After the assertion of the ack signal, the preceding C negates its send signal.
- (3) 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 from the succeeding C 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, as shown in figure 3.

This handshake concentrates dynamic consumption current into the pipeline stages with valid data naturally while the empty pipeline stages can be powered off by the power

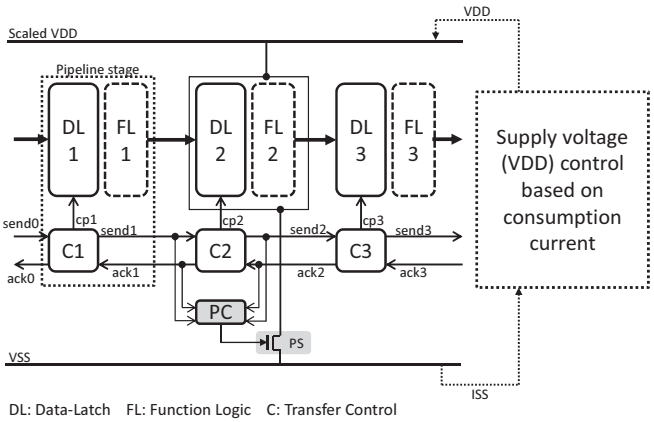


Fig. 2: Self-timed elastic pipeline with run-time voltage scaling and power gating.

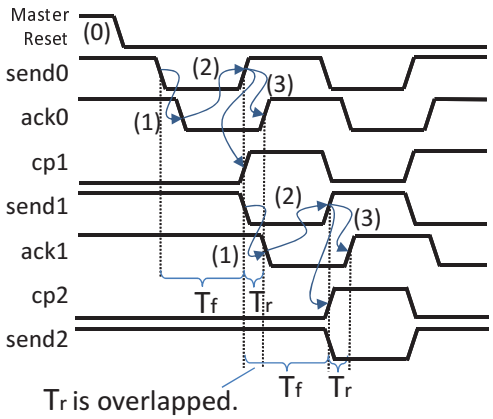


Fig. 3: Handshake timing chart of ULP-STP

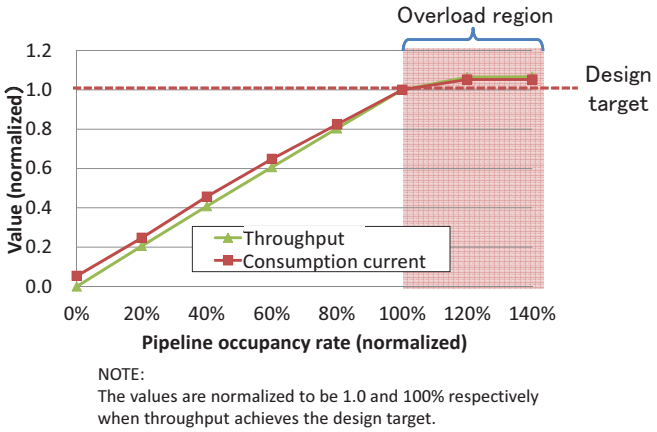


Fig. 4: Proportional relation between throughput and consumption current.

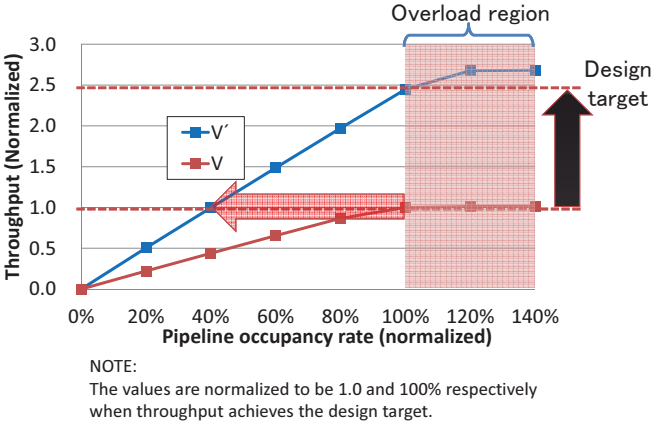


Fig. 5: Run-time scaling of throughput design target.

control and power switch 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 ULP-DDCMP realized by the ULP-STP realizes both ultra-low-power consumption which is demonstrated in [8] and autonomous failure detection/isolation discussed below.

2.3 Energy foreseeability and autonomous failure detection/isolation

Figure 4 shows the measured values of the throughput and current consumption of the ULP-CUE in a prototype LSI chip 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. That is, the power is consumed only for processing. Moreover, the consumption current has a proportional relation with the processing load. This proportional relation 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 [9] and to calculate battery-operated lifetime ($T[sec.] = W/I$) of the 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 not only the economical design of the battery replacement/charge schedules but also run-time detection and isolation of failures. The processing load may be out of the predicted range in operation due to failures such as malfunction of the sensors, and accordingly the consumption current may differ from the predicted amount. In addition, operating environment changes such as the change of temperature also may change the consumption current. Therefore, the actual battery-operated lifetime may differ from the predicted value. Fortunately, by virtue of the proportional relation between the processing load and consumption current, such failures and

contingencies in operation can be autonomously detected by measuring the consumption current. 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 t and the measured run-time consumption current within the t, is compared to the W_t. The deviance of the d from the range of the W_t indicates that the failures or contingencies occur. In operation, devices whose d becomes out of the W_t alerts the failures/contingencies occurrence to the other devices or the center of management with a request to settle the failures/contingencies. This autonomous detection of the failures leads to the reduction of the health checking operations.

Moreover, the failure detection leads to the isolation of the nodes with failures because the alerting nodes can be recognized by their neighboring nodes and the neighbors can isolate the failure nodes by ignoring the packets sent from the failure nodes in the sensor network. This failure node isolation can confine the bad effects of failures within the neighbors of the failure nodes and preserve the other nodes.

2.4 Requirement for tolerance against instantaneously excessive load

In addition to the energy foreseeability and failure detection/isolation, the ULP-DDCMP provides a natural tolerance against the processing load fluctuation. As shown in the figure 4, 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. Moreover, to cope with high load situation in which the processing load exceeds the natural tolerance, 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, as shown in figure 5 [4].

Unfortunately, the observation of the processing load and the run-time supply-voltage control take time due to the signal propagation in the observation and control circuit, the sampling interval of consumption current and the circuit's parasitic capacitance. On the other hand, the prediction of the individual occurrence timing of sensing events and communication requests is difficult while the typical amount of lifetime processing load can be predicted within a certain range. Therefore, instantaneously sudden increase of the sensing events and/or the communication requests may not be handled and thus may cause overload and eventually make the processor inoperative.

Consequently, the data-driven sensor networking processor should have tolerance to such instantaneously excessive load.

3. Autonomously variable input rate circular pipeline

To prevent processors from instantaneously excessive load situations, the fluctuation of internal processing load should be smoothed and the arrival input data should be accepted only when the sum of the increased load to process the input data and the internal processing load is under a given processing capability. That is, the load fluctuation should be predictable while the input of arrival data should be controlled according to the internal processing load in the processor.

In this section, the internal processing load predictability is discussed, and the input control mechanism stopping the acceptance of arriving data naturally is proposed for the operation execution circular pipeline in order to keep the processing load within the elastic capability of the self-timed pipeline.

3.1 Predictability of internal processing load

As illustrated in figure 6, the programs of the data-driven processor are defined by data-flow graph (DFG). The DFG consists of nodes and arcs, and each node corresponds to an operation while each arc represents the data-dependency between two successive operations. The data-dependencies between operations expose naturally the ILP (Instruction Level Parallelism) inherent in the programs, and thus describing target program by using the DFG results in extracting the whole ILP in the target programs. In the data-driven processor, each operand is executed independently from the other operands and the execution time of each operand is also independent from that of the other operands as a result of the real-time multiprocessing. Therefore, the maximum number of operations executed concurrently for single data is determined by the concurrent operations, and the maximum processing load which is the maximum number of data flowing in the operation execution circular pipeline can be calculated by the product of the maximum number of operations executed concurrently and the maximum number of data processed concurrently.

From the viewpoint of the overload avoidance, the maximum processing load should be as low as possible because the processing load observed though consumption current may differ from the actual processing load and it becomes difficult to keep the actual processing load within a certain value for the large fluctuation of processing load. Fortunately, the number of operations executed concurrently can be changed by postponing the execution timing of the operations on non-critical paths, as shown in figure 6. This program modification can smooth the processing load fluctuation without any overhead on the execution time of the operations on the critical path of target programs.

However, the number of the concurrently executed operations on the critical path should be preserved in order to keep

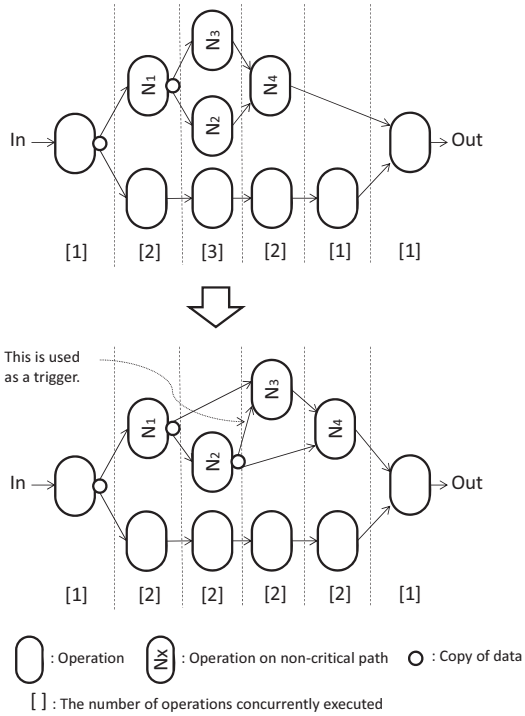


Fig. 6: Data-flow graph exposing the maximum processing load.

the throughput of the program, and thus the actual processing load may change during the observation. To prevent the processors from the overload, the acceptance of the arrival data should be postponed while the maximum processing load after accepting the data exceeds the elastic capability of the self-timed pipeline.

3.2 Elastic capability of self-timed pipeline

The operation execution circular pipeline has tolerance to temporal overload situation by virtue of the elastic capability of the self-timed pipeline. In the self-timed pipeline, each input data is packed into a packet-style set named token. The elastic capability can be defined by the propagation time of the signals to transfer the token. As shown in figure 3, the minimum time for handshake at a pipeline stage is (T_f+T_r), where T_f and T_r denotes the send signal propagation time and the ack signal propagation time respectively. When the temporal distance $D_{(t)}$ which means the signal propagation time on the critical path of the circuit between two adjacent tokens is equal to or greater than $(T_f + T_r)$, the token can be transferred at T_f because the ack signal arrives before send signal and T_r is overlapped. On the other hand, when $D_{(t)}$ is temporarily less than $(T_f + T_r)$, the transfer of the token is postponed until $D_{(t)}$ becomes equal to or greater than $(T_f + T_r)$. This postponed time is called transfer wait time in this paper.

The transfer wait time can be absorbed if it is equal to or less than the temporal margin $D_{(t)} - (T_f + T_r)$ of

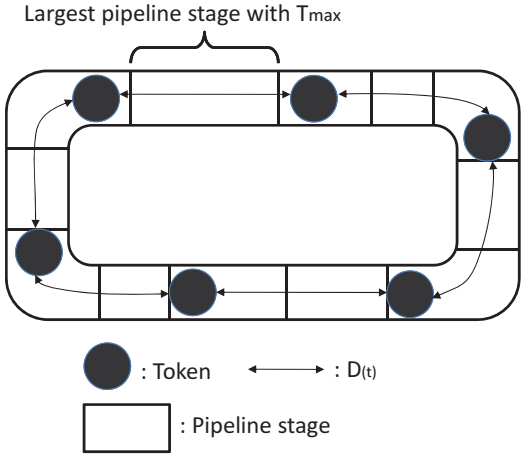


Fig. 7: Temporal distance between tokens.

the following token. That is, the sum of the $D_{(t)}$ in the circular elastic pipeline determines the elastic capability to absorb the transfer wait time. In the circular pipeline, $(T_f + T_r)$ of a pipeline stage is usually different from that of the other pipeline stages. For example, the arbitration at a merge stage may postpone the arrival of the ack signal and thus the T_r may be increased during the arbitration. Moreover, it is difficult to strictly retain the designed signal propagation time through circuit implementation phase in which unexpected delays are produced by design tools and fabrication environment.

As shown in figure 7, by virtue of the handshake of the STP, tokens autonomously attempt to keep $D_{(t)} \geq T_{max}$, where T_{max} denotes the largetst $(T_f + T_r)$ in the circular elastic pipeline. The tokens with $D_{(t)} \geq T_{max}$ are transferred between stages at T_f, and thus the sum of $D_{(t)}$ can be $\sum T_f$ at maximum. Based on these facts, the number of tokens, which is denoted by P_{total}, should satisfy equation (1) to assure $D_{(t)} \geq T_{max}$ for each data.

$$\sum T_f \geq T_{max} \times P_{total} \tag{1}$$

According to the equation (1), the design target, which is denoted by P_{DT}, is defined by equation (2).

$$P_{DT} = \lfloor \frac{\sum T_f}{T_{max}} \rfloor \tag{2}$$

Based on the above equations, the processing time increases due to the transfer wait time when the processing load, which is P_{total}, exceeds the P_{DT} temporally. However, the increased processing time decreases the throughput and may lead to the overload situation in which all of pipeline stage are filled with tokens and the pipeline becomes inoperative.

3.3 Autonomously variable input rate mechanism

To avoid the overload situation, the arriving data should be accepted only when the P_{total} can be kept within less than pl, where the pl indicates the total number of pipeline stages. As discussed in the section 3.1, the maximum processing load after accepting the arriving data, which is denoted by P_{max}, can be estimated, and it is calculated as $(P_{total}+1)\times P_{single}$, where the P_{single} denotes the maximum value of the P_{total} when processing single token. Based on these facts, to guarantee that the P_{max} is less than the pl, the arriving data should be accepted only when the P_{total} is less than $\frac{pl}{P_{single}}-1$.

To realize such input acceptance, the natural acceptance control of the self-timed pipeline is utilized. Each stage of self-timed pipeline waits to accept the tokens transferred from its preceding stage until it finishes processing and becomes empty. To inherit this control, the transfer control unit of the merge stage in the circular pipeline is modified to vary the input rate autonomously according to the number of the empty stages. Figure 8 illustrates the merge stage with the modified transfer control unit. As shown in the figure 3, either the send signal or the ack signal is asserted and becomes logically 0 during data processing. That is, whether a pipeline stage is empty or not can be detected by using only AND gate whose inputs are connected with the send and ack signals of the pipeline stage. In order to assure that the P_{total} is less than $\frac{pl}{P_{single}}-1$, the number of empty stages should be $pl-\lfloor\frac{pl}{P_{single}}-1\rfloor$. Based on these facts, the modified transfer control unit monitors the pipeline stages which should be empty by the AND gate whose inputs are connected all of the send and ack signals of the target pipeline stages and stops the handshake with the external pipeline stage while the target pipeline stages are empty.

To realize this handshake control, the arbitration circuit in the merge stage is modified as shown in figure 9. The original arbitration circuit realizes first-come-first-served basis as a result of the saturable reactor (SR) flipflops which sets one of two gate signals activating the cp signal at a time. In the modified arbitration circuit, the empty signal of the target pipeline stages is utilized to stop the handshake with the external pipeline stage during it is negated.

With this merge stage with the modified transfer control unit, the input rate of the circular pipeline can be varied autonomously according to the internal processing load, and the tolerance against instantaneously excessive load can be achieved.

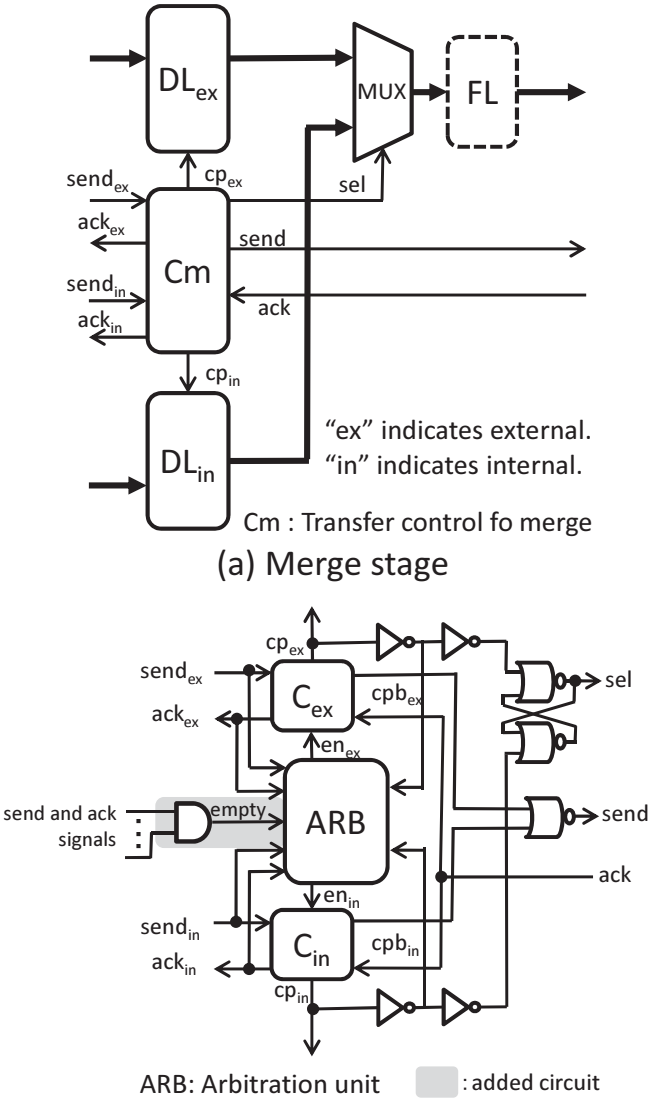


Fig. 8: Merge stage with autonomously variable input rate mechanism.

4. Evaluation on instantaneously excessive load tolerance

To show the feasibility and the effectiveness, the autonomously variable input rate mechanism is realized into the circular pipeline of the ULP-CUE and the tolerance against instantaneously excessive load is evaluated through circuit simulation.

The ULP-CUE is divided finely in order to eliminate the pipeline bottleneck. As a result of this pipeline division, the number of stages (pl) of the ULP-CUE is 13. The prototype LSI chip of the ULP-DDCMP integrating four ULP-CUE's has been designed and fabricated by 65nm CMOS 7-metal-layer process technology. The circuit of the ULP-CUE with the proposed mechanism is described at RTL (Register Transfer Level) by utilizing the ULP-DDCMP's RTL description and pipeline tact information extracted from the circuit layout of the prototype LSI chip.

As the application program, a UDP/IP program is used because the UDP/IP's connection-less communication results

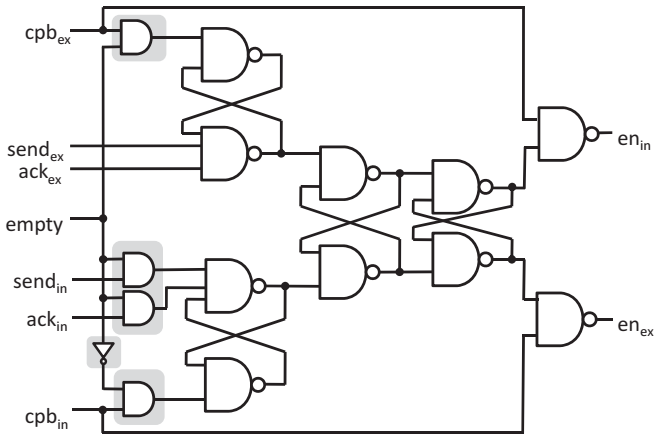


Fig. 9: Arbitration logic realizing variable input rate.

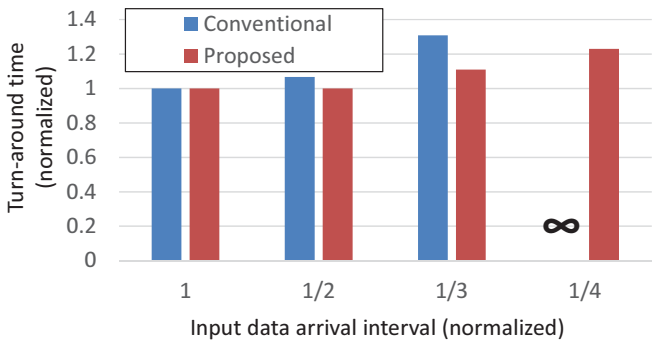


Fig. 10: Turn-around time.

in low-power consumption and it is one of the protocols suitable to realize sensor networking systems with low-power consumption. The ULP-CUE provides an instruction set enough to describe the UDP/IP handling program. The described UDP/IP handling program realizes the checksum calculation and the generation of the UDP/IP header, and the packets containing pseudo header and payload are input to the program and the program outputs IP datagrams. The number of the operations executed simultaneously in the originally described program varies from 1 to 5 for processing single token, and thus the P_{single} is 5. That is, the successive acceptance of only 3 arriving data may increase the P_{total} up to 15 instantaneously and make the ULP-CUE inoperative. As explained in the section 3.3, to avoid such instantaneous overload, 12 stages should be empty to accept arriving data because $pl - \lfloor \frac{pl}{P_{single}} - 1 \rfloor = 12$. Based on this calculation, the send and ack signals of the 12 stages are connected to the input of the AND gate of the proposed mechanism.

To check the tolerance, turn-around time from the acceptance to the output is measured by changing the interval time of input data arrival to the ULP-CUE. The turn-around time of the ULP-CUE with the proposed mechanism (proposed) is compared with that of the original ULP-CUE (conventional). The comparison result is shown in figure 10. The result shows that the ULP-CUE with the proposed mechanism can tolerate the instantaneously excessive load and operate continuously, while the original ULP-CUE becomes inoperative when the normalized input data arrival interval is 1/4. The turn-around time changes within approximately 30%. This is because the temporary overload is buffered by the elastic capability of the self-timed pipeline and the processing time is increased as a result.

5. Conclusion

In this paper, an autonomously variable input rate mechanism is proposed to realize data-driven sensor networking processor with tolerance against instantaneously excessive load. With the proposed mechanism, the operation execution circular pipeline never become inoperative regardless of the input data arrival timing. To show the feasibility and effectiveness of the proposed mechanism, the turn-around time of a UDP/IP program is measured through a circuit simulation while shortening the input data arrival interval to cause the instantaneously excessive load. The result shows that the circular pipeline with the proposed mechanism operates even while the conventional circular pipeline becomes inoperative.

Acknowledgement

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] Shuji Sannomiya, Hiroaki Nishikawa, "Highly-Dependable and Long-Lifetime Data-Driven Networking Processor with Energy Assurance Capability," in Proc. of PDPTA, pp.557-563, July 2015.
[2] 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.
[3] Shuji Sannomiya, Hiroaki Nishikawa,"Energy Efficient Data-Driven Networking Processor with Autonomous Load Distribution Capability," in Proc. of PDPTA, pp.514-520, July 2014.
[4] Shuji Sannomiya, Yukikuni Nishida, Makoto Iwata, and Hiroaki Nishikawa, "An Overload-Free Data-Driven Ultra-Low-Power Networking Platform Architecture," in Proc. of PDPTA, pp.604-610, July 2013.
[5] 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.
[6] 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.
[7] C. J. Myers, "Asynchronous circuit design," Univ. of Utah John Wiley & Sons, Inc., 2001.
[8] 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.
[9] Kazuhiro Aoki, Shuji Sannomiya, Hiroaki Nishikawa, "Data-Driven Sensor Networking System Simulator," in Proc. of PDPTA, pp.564-570, July 2015.

Self-Timed I/O Architecture of Data-Driven Sensor Hub

Hiroki SHIBUTA and Makoto IWATA
Graduate School of Engineering, Kochi University of Technology,
Kami, Kochi, 782-8502 Japan

Abstract— *With rapid increase of Internet of Things (IoT) devices, processing load on the cloud server would be tremendously increasing. In order to suppress such heavy load, edge-heavy computing has been studied at various communities. The typical edge device is generally composed of multiple sensors, a sensor hub to integrate sensed data, an application processor (AP), and a communication module. When equipped with a sensor hub, AP is free from sensor management process. Whenever the interrupt will occur in the hub, its processing resource and power will be wasted.*

This study focuses data-driven processor (DDP) that can operate without any interrupt process. This paper presents an I/O architecture of data-driven sensor hub (DDSH) processor as an extended version of DDP. DDSH is composed of the DDP core and I/O units which convert data/packet format in adapting to individual peripheral device interface. Furthermore, the proposed DDSH supports hub instructions to realize sensor fusion on edge devices and to accelerate their performance by application-specific coprocessors.

Keywords: sensor hub, data-driven processor, self-timed pipeline, internet of things

1. Introduction

With rapid increase of Internet of Things (IoT) devices [1], the amount of processing and traffic load on their cloud server will be tremendously increasing. Therefore, edge-heavy computing has been recently studied in various universities and companies to reduce the heavier load on the cloud server [2]. An edge IoT device is typically composed of multiple sensors, a micro controller unit (MCU) to operate multiple data streams from the sensors such as inertial measurement unit, Hall effect sensor, and so on. Also, it sometimes employs an internet communication module. If the IoT device is equipped with a sensor hub [3] to aggregate the sensed data from multiple sensors, as illustrated in Fig. 1, the application processor (AP) is free from any management process for various sensor interfaces so that the AP's processing resource can be dedicated to operate essential data with required minimum runtime power. Furthermore, the stand-by power of the IoT device can be reduced because the sensor hub can help the AP sleep longer.

The typical IoT device equipped with a sensor hub receives sequences of signals from multiple sensors, manipulates the data in sensor hub, performs an advanced processing in AP, and transmits the data to the cloud by using the communication equipment. For example, an edge-heavy computing device may extract a disparity image from two image sensors and transmit it to the AP or the cloud server. Such an integration process by combining the multiple sensor data is expected to be an edge-heavy computing function, which is called sensor-fusion [4].

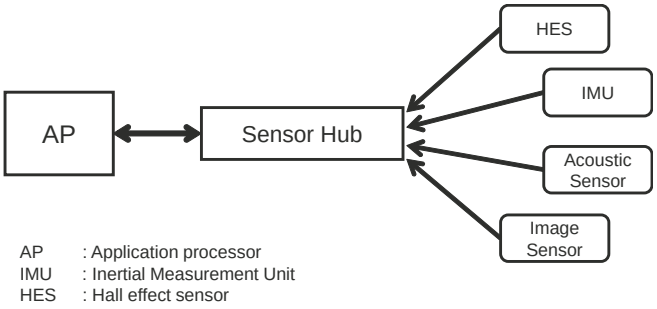


Fig. 1: IoT device equipped with sensor hub.

The typical sensor hub is generally based on the Von Neumann architecture. Therefore, it is necessary to preprocess the interrupt processing or the like when the input events from multiple sensors have occurred. As a result, the sensor hub is required to save current process context and to resume it from the interrupt point after the execution of interrupt process. Whenever the interrupt will occur in the hub, its processing resource and electric power will be wasted for the extra overhead of the interrupt handling.

We have studied on data-driven processors (DDP's) such as DDMP [5]. The DDP can perform parallel pipelined execution by a trigger for the arrival of data if the DDP is implemented based on the self-timed pipeline (STP) circuit with a handshake protocol. Therefore, the DDP doesn't require an interrupt process to receive multi-sensor data. For this reason, the DDP can be expected to realize high performance and lower power consumption as compared to the typical sensor hubs.

This paper proposes an architecture of data-driven sensor hub (DDSH) as an extension of the DDP. Specifically, we discuss an I/O architecture and its extended instruction set in order to realize essential mechanisms in the sensor hub. The performance evaluation results of the DDSH circuit which is designed by 65nm CMOS standard logic cells will be also discussed in the paper.

2. Data-Driven Sensor Hub

The basic operation of the sensor hub is to receive data from multiple sensors and to transmit data to an application processor. The sensor hub architecture is basically controlled by the program counter so that it is necessary to invoke a data receiving process by a polling or interrupt mechanism. The polling mechanism regularly monitors the validity of data coming from an external device. For example, in case of an IoT device to detect an irregular event during a few hours, millisecond-order polling process must waste the CPU resource and energy. In the case of interrupt processing, it is necessary to suspend its current process, execute the interrupt code, and resume the suspended process from the interrupted point. Therefore, frequent interruptions induce an increase in the performance degradation and power dissipation.

Therefore, an architecture suitable for the sensor hub must operate based on a passive operation mode. This means that it is not necessary to introduce any active receiving mechanisms such as the polling or interrupt mechanism. Thus, this study focuses on the data-driven processor (DDP) where the program execution is triggered by the arrival of the data. In this section, a sensor hub extension of the DDP is discussed after a brief introduction of the self-timed pipeline (STP) and the STP-based DDP.

2.1 Basic Architecture of Data-Driven Processor

In the DDP, data transfer between the stages of a pipeline is controlled by self-timed pipeline (STP) [5]. The STP is composed of Coincidence flip-flops (C-element's), as shown in Fig. 2. The STP operates in a handshake protocol between C_i and C_{i+1}. The C-element between adjacent pipeline stages exchanges the data transfer request signal (send) and the data transfer acknowledge signal (ack). The STP has an autonomous power saving feature, which means that the STP consumes power only when valid data in the stages is processed and transferred. This is because the STP makes it possible to minimize the wiring and power consumption [5].

By virtue of the STP behavior, the STP-based DDP works according to the passive operation mode. That is, only when an incoming packet arrives at the input portion of the DDP, the DDP operates the packet without any interrupt or polling operation. In the basic architecture of the DDP, the data-driven program is interpreted according to the data-dependency among the operation nodes. Each data on the data-driven program is implemented using a packet format shown in Fig. 3. The color identifier (*color*) denotes a context or state of a process instance in multi-process execution. It can be associated with a specific input source such as sensors. The generation identifier (*gen*) represents an index of a packet belonging to a packet stream. It can be used

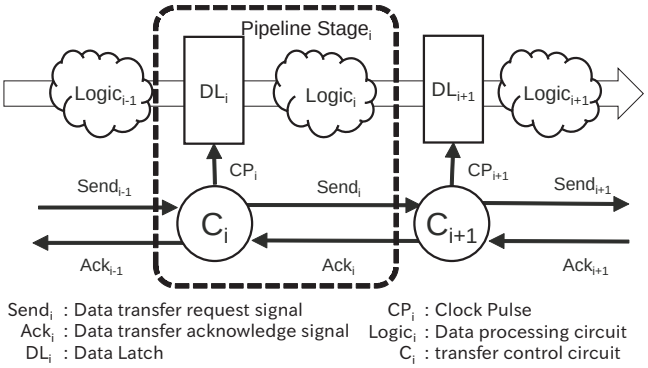


Fig. 2: Basic structure of self-timed pipeline.

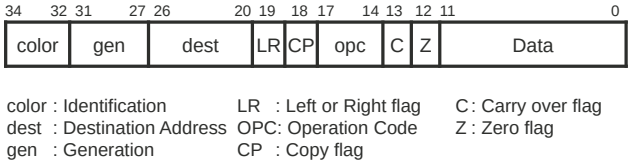


Fig. 3: Packet format for data-driven execution control.

to identify the order of the packets. The destination field (*dest*) denotes a destination node number in a data-driven program. The packet representing the data is manipulated in a circular self-timed pipeline to realize the data-driven processor shown in Fig. 4.

The data-driven execution control employed in the DDP is realized by the following self-timed pipeline stages:

- Merge Unit (M)
  The merge unit joins two packet streams from two independent self-timed pipelines into a packet flow through a single self-timed pipeline.
- Constant Memory (CST)
  The constant memory stores constant values to be used by immediate operations belonging to the data-driven program. If the operation code in the operand packet indicates an immediate operation, its corresponding constant value is fetched from the constant memory and it is then appended to the packet. If it is not a constant

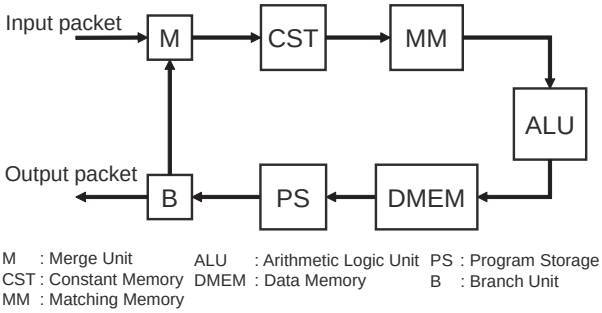


Fig. 4: Basic architecture of data-driven processor.

instruction, the packet passes through the CST to the MM without accessing to the constant memory.

- Matching Memory (MM)
  The matching memory provides a temporal storage to control the firing of binary operation nodes in the data-driven program. In case of binary operation, the packet is temporarily held in the internal memory of the MM. When its paired packet with the same destination and generation arrives, the MM outputs a packet containing two operands for the binary operation. At that time, the *LR* flag in the both operand packets is checked and whether the data packed in the packet is the first (left) or second (right) operand for its binary operation. In case of unary operation or immediate operation, the packet passes through the MM without any packet manipulation.
- Arithmetic Logic Unit (ALU)
  The ALU performs arithmetic and logic operations in accordance with the operation code (*opc*) in the operand-pair packet. After that, it appends the result data to the packet header and outputs the resultant packet to the data memory. At that time, if the result data is zero, the zero flag (*Z*) is set to 1. If the arithmetic operation generates a carry, the carry flag (*C*) is set to 1. The instruction set employed in the DDP supports basic arithmetic, logic, and shift operations, simple branch instructions based on the *Z* or *C* flags, and data memory access instructions (load or store).
- Data Memory (DMEM)
  The DMEM supports memory access for a load / store instruction. The memory address to load or store data is included in the operand packet. In case of the store instruction, the stored data is also included in the operand packet. The result packet of the DMEM includes its loaded data or a store completion flag.
- Program Storage (PS)
  The PS holds a data-driven program to be executed. The destination node in the result packet indicates the address for the next instruction. The PS concatenates the resultant data with the next operation code and destination node fetched from the program storage. If the *CP* flag in the packet is set, the result data will be duplicated (copied) and another operation code and destination node stored in the next PS address will be fetched. After that, the PS outputs it as next operand packet to the branch stage (B).
- Branch Unit (B)
  The branch unit splits a stream of operand packets to two output streams of packets flowing to two succeeding pipelines. The destination pipeline of every packet is determined according to its destination node or branch flag. One of the destination pipeline is the circular pipeline to continue to execute the remaining instructions of the program. The other one is the output pipeline to communicate with the external modules such as I/O devices, or co-processors.

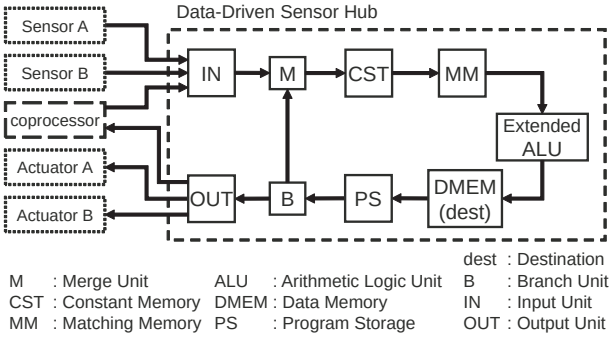


Fig. 5: Basic architecture of data-driven sensor hub.

Although the DDP works passively, some modifications or extensions are necessary to apply the DDP to a sensor hub processor, as discussed in the nest subsection.

2.2 Architecture of Data-Driven Sensor Hub

In order to apply the DDP to the sensor hub, I/O interface functions between the DDP and sensor devices will be required. In addition, a co-processor interface functions are required to realize an edge-heavy IoT device.

However, there are many kinds of interfaces for sensors such as I2C, SPI, etc. Furthermore, there are various co-processors such as the fast Fourier transform, floating-point arithmetic operations, and so on. It is impossible to adapt the DDP interface to individual specific interface of the device, so that a kind of generalized interface module should be introduced to realize a data-driven sensor hub (DDSH).

Therefore, our DDSH based on the basic DDP is equipped with a generalized input and output module, which interfaces multiple sensor devices and co-processors, as shown in Fig. 5. Hence, the different types of sensors or co-processors can be connected the DDSH. In the input (IN) module, raw data from the sensor are transformed into the DDP packet format. In the output (OUT) module, the DDP packet is transformed to the proper format depending on the external device or co-processor interface. Furthermore, additional instruction set is introduced into the extended ALU to interface them. Those are explained in detail in the following section.

3. I/O Architecture of DDSH

Generally, the sensor hub in the IoT device is connected to multiple sensors, such as passive infrared sensor and inertial measurement unit, and multiple actuation devices such as light emitting diode and speaker. Each device has its own specific data format and protocol. In this section, those data formats and protocols are classified in terms of the DDSH's passive operation mode and the I/O architecture of the DDSH will be proposed.

3.1 Requirement Specification

The packet format of the data-driven processor (DDP) includes the following information as shown in Fig. 3. The color identifier in the packet represents which external device generates or consumes the data in the packet. The generation identifier (*gen*) is used as an index order within a packet stream of its corresponding device identified by its color identifier. Although the destination field in the packet format is usually used as the destination node in the data-driven program, this field is also utilized as an output port number in the DDSH architecture. In order to represent flexible sensor hub functions coping with multiple sensors, those identifiers and destination have to be manipulated in its data-driven program. Therefore, the color modification instruction, the generation modification instruction, and the destination modification instruction are designed for the DDSH.

In the proposed DDSH architecture, diversity of input devices are abstracted by mapping specificity to those packet fields as summarized in Table 1. In the table, input devices are classified to synchronized or asynchronized device. Furthermore, a stream of data form the device is classified whether it is regularly ordered or not. If not, the data in the stream must be a form of packet, and thus it can be identified by its generation identifier. The non-ordered asynchronized I/O devices are excluded because they do not generally exist. The data stream from the synchronized I/O devices such as an A/D converter and micro control unit (MCU) is usually ordered, so that the data may be raw and thus it must be formed in the DDP packet format by adding the color and generation identifiers to the raw data. Then, the destination node, operation code, and flags necessary to process the input data are fetched at the I/O PS. The asynchronous event data from the sensor such as the Hall effect sensor (HES) can be dealt with in the same way in the DDSH if the data is digitized at the sensor device. As for the non-ordered asynchronous devices, we assume the self-timed data-driven processor or co-processor that copes with the similar packet format shown in Fig. 3.

As for the output interface of the DDSH, the destination field in the packet is used to identify an output external device. Some fields of the packet might not be necessary for the external device. Therefore, the output interface functions are specified as shown in Table 2.

Table 1: Specifications for DDSH interface (input).

	Synchronous Devices		Asynchronous Devices	
	ordered	non-ordered	ordered	non-ordered
color	Input ID		Input ID	✓
gen	(order)		(order)	✓
dest	I/O PS[color]	–	I/O PS[color]	✓
opc	I/O PS[color]		I/O PS[color]	✓
flags	I/O PS[color]		I/O PS[color]	✓
ex.	A/D, MCU	–	HES	DDP

flags: LR, CP, C, Z ✓ : transfer it with data. – : null

Table 2: Specifications for DDSH interface (output).

	Synchronous Devices		Asynchronous Devices	
	ordered	non-ordered	ordered	non-ordered
color	– / ✓	✓	–	✓
gen	(reorder)	✓	(reorder)	✓
dest	Output ID	Output ID	Output ID	Output ID
opc	–	–	–	✓
flags	–	–	–	✓
ex.	D/A, MCU	MCU	LED	DDP

flags: LR, CP, C, Z ✓ : transfer it with data. – : null

Since the DDSH can execute multiple operations in parallel based on the data availability in the MM, the order of execution might alter in some cases. In this case, the output packets from the DDSH must be re-ordered in order to communicate with a synchronous device such as D/A converter. Furthermore, in the case that the DDSH transfers the result packets to the application processor such as MCU, the color identifier associated with a specific sensor context must be transferred with the data.

If the synchronous device such as MCU can accept non-ordered data stream with the color and generation, the DDSH transfers the data with its identifiers, color, and generation. In case of the event-driven actuator such as toggle-switched LED, the output packets must be re-ordered if the orders of packets are altered in the DDSH. In case of the asynchronous device such as the DDP, the output packet can be transferred as it is.

3.2 Input module

According to the above discussion on the specification of the DDSH input interface, the input module (IN) is designed as shown in Fig. 6.

In order to accommodate the sensor devices in the IoT node, the DDSH must transform the raw sensor data into the DDSH packet. At first, the color identifier is attached to the raw data at individual color generator (CG) corresponding to the sensor. Next, through the merge unit, the generation

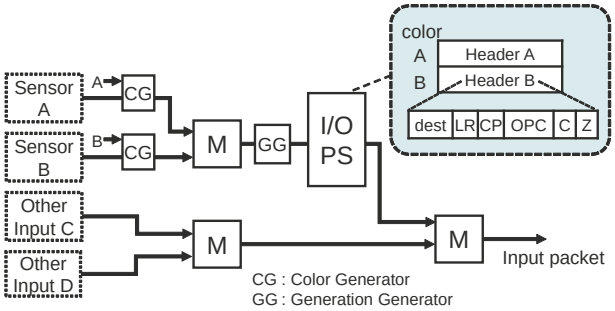


Fig. 6: A block diagram of input module.

identifier is attached to the packet at a generation generator (GG). After that, the header information for the packet is fetched at the I/O PS which holds the entry nodes of the data-driven programs for the sensor process such as filtering, sensor fusion, and so on. The address of the I/O PS is associated to the color identifier of the packet. In case of interface to the conventional synchronous processor such as MCU, the raw data can be accepted in the same way. In case of the input interface for the co-processor or DDP, the packet is accepted as it is. Additional modules for mutual conversion of data between the external device and DDP are input mechanism IN and the output mechanism OUT. Basically, the incoming packets to the DDSH are accepted based on the first-come-first-served policy when the normal merge unit is used for the input module. The priority-based service could be introduced if the prioritized merging would be implemented.

3.3 Output module

The output module (OUT) for the DDSH is designed as shown in Fig. 7. The output packet from the branch unit (B) has its destination as an identifier of the output external device (output ID). At each branch unit with packet formatter (BF) in the output module, the packet is routed to an output device specified by the output ID. If the output device accepts only an ordered data, the BF formats and outputs the data with its generation identifier. After that, if the orders of the output packets are altered in the DDSH, they are re-ordered at the small sorter module in accordance with their generation identifiers and transferred to the target output device. An efficient implementation of self-timed sorter has been already proposed in [6]. This can be utilized in our DDSH implementation.

The destination field of the DDSH packet is basically used as a destination node in the data-driven program. Since the data-driven program is stored in the PS, the output ID in destination field can be also stored in the PS. This implementation is feasible if the output ID can be defined statically. If the output device is decided in runtime, the output ID must be stored in the data memory addressable in the program. Therefore, a compound instruction to change output ID is introduced in the DDSH. The instruction of

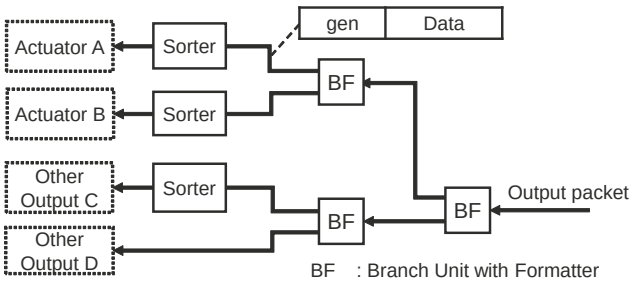


Fig. 7: A block diagram of output module.

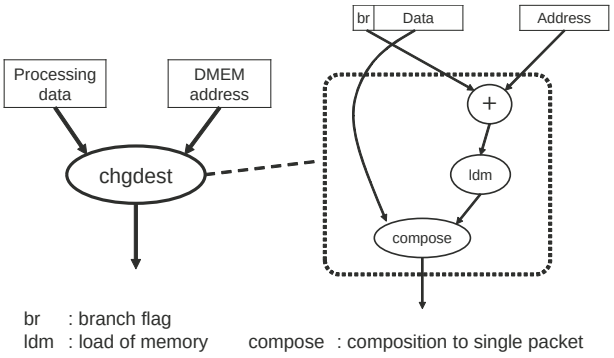


Fig. 8: Instruction for change destination (*chgdest*).

chgdest is executed as shown in Fig. 8. If branch instruction is running, the ALU calculates data memory address by operand of address and branch flag of data packet. After that, the DMEM reads output ID from the data memory by the calculation address. The output ID and data are composed into single packet at the DMEM stage.

As discussed in this section, the proposed architecture for self-timed data-driven sensor hub processor could be one of promising implementation to realize a sophisticated sensor hub that can operates I/O data without interface overhead in terms of the passive operation.

4. Evaluation

For the preliminary evaluation, DDSH is designed using a 65nm CMOS standard cell library. The designed circuit is described by Verilog-HDL and synthesized by Design Compiler, Synopsys Inc. The specifications of this circuit are shown in Table 3. The number of STP stages organizing the DDP core is 8, where the MM stage is divided into two stages, a stage for associative memory access and a stage for operand-pair packet composition, and where the PS stage is divided into two stage, a stage for program storage access and a stage for packet absorb and branch. As for other pipeline stages, M, CST, ALU, and DMEM , each of them is realized as single stage as shown in Fig. 5. The I/O devices connected to the DDSH is 4 respectively. Furthermore, each SRAM memory size necessary to implement the DDSH is as listed in the table.

Table 4 shows total cell area of the synthesized DDSH circuit. As shown in the table, the area overhead of the I/O modules is 67% in total cell area of the DDSH. This is because the size of the data is set to 12bit, which is a usual resolution in typical sensors. If the size of data is expanded to that of normal processors, e.g., 32bit or 64 bit, the area of I/O modules could be relatively reduced. It would be realistic condition, especially in case of the DDSH supporting heavy sensor fusion. Actually, total cell area of DDSH is reasonable as a general sensor hub, e.g., the core processor of the sensor hub SAM D20 (Atmel Co.) [7] is

Table 3: Specifications of DDSH circuit.

Process		SOTB 65nm CMOS
# stages of STP ring		8 stages
# I/O devices		4 inputs, 4 outputs
Constant memory	(CST)	15bit × 128 word
Matching memory	(MM)	37bit × 16 packets
Data memory	(DMEM)	12bit × 512 word
Program storage	(PS)	12bit × 128 word
I/O Program storage	(I/O PS)	14bit × 8 word

Table 4: Cell area of synthesized DDSH circuit.

	core	I/O	total
standard cells	1.2k	0.4k	1.6k
area [mm^2]	0.0122	0.0082	0.0204

equipped with ARM Coretex-M0 that area is $0.04mm^2$.

Table 5 is a comparison result of the simulated performance of DDSH and the emulated performance of a typical MCU in the case that both work as a sensor hub. In the comparison, RX210 (Renesas Electronics Corp.) is picked up as the typical MCU. The sensor hub performance of DDSH and MCU is defined as the effective throughput [operations/s] except for interrupt operations. In addition, the input data rate to the sensor hub is assumed to be 8 k sample/s.

Table 5: Performance comparison of DDSH and RX210.

	DDSH	RX210
Throughput $[operations/s]$	41.6M	33.6M

The table shows the DDSH achieves about 24 % performance improvement of the MCU. This is because it takes $(N+6)$ clock cycle (N denotes the number of used registers) to execute the interrupt process in RX210, i.e., its processing resource available for essential process are wasted.

5. Conclusion

In this paper, the I/O architecture of data-driven sensor hub (DDSH) with a single circular self-timed pipeline (STP) is proposed. The DDSH has less I/O overhead compared with program-counter-based sensor hub processors. The DDSH is composed of an input module (IN), an output module (OUT), and a data-driven processor core. In addition, I/O instructions are introduced into the DDSH for implementing a sensor fusion. The circuit performance of the proposed architecture is evaluated based on 65 nm CMOS standard cell library. The evaluation results show that the performance of the DDSH is 24 % superior to that of a typical MCU, RX210.

Since lower energy consumption is indispensable to most of battery-operated IoT devices, the designed circuit must be implemented with typical low power techniques, e.g., dynamic voltage scaling (DVS) [8] and power gating (PG) [9]. In this study, the computing performance of DDSH is only evaluated. Thus, power consumption of DDSH introducing those low power techniques should be evaluated as remaining works in our project. Furthermore, to tolerate instantaneous excessive load at the sensor hub must be supported as one of essential features for the IoT devices. Since the DDP realized by the STP has an elastic buffering capability, this can be extend to its I/O interface to detect and regulate such excessive traffic from multiple sensors, as proposed in [10].

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 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] "Special Issue on Advancing the Internet of Things," *IEEE Computing edge*, vol. 2, no. 4, pp. 9–45, Apr. 2016.

[2] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, Feb. 2014.

[3] P. Charith, J. Prem, and Z. Arkady, "Dynamic Configuration of Sensors Using Mobile Sensor Hub in Internet of Things Paradigm," *IEEE ISSNIP*, pp. 473–478, Apr. 2013.

[4] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, Jan 1997.

[5] H. Terada, S. Miyata, and M. Iwata, "DDMP's: Self-Timed Super-Pipelined Data-Driven Multimedia Processors," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 282–296, Nov. 1999.

[6] K. Komatsu, S. Sannomiya, and M. Iwata, "Interaction Self-Timed Pipelines and Elementary Couping Control Modules," *Transactions of IEICE*, vol. E92-A, no. 7, pp. 1642–1651, July 2009.

[7] "SAM D ARM Cortex M0 MCUs - Atmel," http://www.atmel.com/products/microcontrollers/arm/sam-d.aspx.

[8] K. Miyagi, S. Sannnomiya, M. Iwata, and H. Nishikawa, "Low-Powered Self-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling," in *International Conference on Parallel and Distributed Processing Techniques and Applications*, July 2013, pp. 618–624.

[9] K. Miyagi, M. Iwata, S. Sannnomiya, and H. Nishikawa, "Self-Timed Pipeline with Fine Grain Power Gating and Its Evaluation," *IEICE Transactions on Fundamentals*, vol. J97-A, no. 8, pp. 554–564, Aug. 2014.

[10] S. Shuji, N. Yukikuni, I. Makoto, and N. Hiroaki, "Data-Driven Sensor Networking Processor Tolerating Instantaneously Excessive Load," in *International Conference on Parallel and Distributed Processing Techniques and Applications*, July 2016 (to be presented).

Self-Timed Pipeline Register Operating at Near-Threshold Voltage

Tomoki OGAWA and Makoto IWATA
Graduate School of Engineering, Kochi University of Technology,
Kami, Kochi, 782-8502 Japan

Abstract— *In recent years, steady improvement on both performance and energy efficiency of LSI systems by virtue of the miniaturization of CMOS process has been harder. However, lower power devices are still demanded, especially in growing IoT market. Although near-/sub-threshold voltage operation is one of promising ultra-low-power techniques, there are critical issues on exponential performance degradation, huge performance variations, and high probability of functional failures. In order to solve those issues, this study focuses on the self-timed pipeline circuit which has inherent robustness against circuit delay variation.*

In this study, a self-timed data-transfer control circuit and pipeline register operable under sub-threshold voltage is proposed. Through SPICE simulation of self-timed control circuit, called C-element, an optimal CMOS transistor sizing method is discussed to help C-element and pipeline register more robust under sub-threshold voltage region. SPICE simulation results of 65nm CMOS circuit reveal that the circuit can correctly operate in 0.1 V steps under nominal voltage, variations of process, voltage, and temperature.

Keywords: Near-/Sub-threshold voltage, C-element, self-timed pipeline

1. Introduction

Miniaturization of semiconductor process along with Moore's Law has brought great benefit to modern computing devices, but the increase of power consumption of CMOS circuits has become a serious problem with denser integration of semiconductor transistors. However, lower power devices are still demanded, especially in growing IoT market.

In order to reduce the power consumption significantly, near-threshold computing (NTC) in which the circuit operates at near or less threshold voltage has been recently studied [1], [2]. This is because power consumed in CMOS circuit is proportional to the square of the voltage. However, the problem is often caused by lowering power supply voltage (VDD). Major critical problems on NTC are as follows.

(1) Performance Degradation.
The performance of CMOS circuit decreases in proportion to the supply voltage when it is over the threshold voltage (Vth) of the CMOS transistor. However, the performance is degraded exponentially when near- or sub-threshold voltage.

(2) Increase performance Variation.
Drivability of CMOS transistor usually depends on Vth, VDD, and temperature, especially in the near-threshold region. As a result, NTC will bring considerable variations in circuit performance.

(3)Function Failure.
Similar to the performance variation, variations in process, voltage, and temperature have a significant impact on the occurrence probability of function failures in the circuit, especially at the near-threshold voltage region. In order to eliminate the function failures, circuit design margin have to be considered to guarantee plenty of space at the sacrifice of its performance. As a result, it may induce the increase of leakage energy per operation.

In case of the modern synchronous circuit, clock distribution to the whole chip area tends to lengthen the wires. The longer wiring delay might induce more malfunction due to the clock skew. In contrast, the self-timed pipeline (STP) circuit [3] controls data-transfer within the pipeline by using hand-shake signals between adjacent pipeline stages. STP is hence designed to wire only adjacent pipeline stages, so that the influence of the wire delay variation induced by various process and environmental variations could be localized. The design margin could be also minimized for sub-threshold voltage operation. It can be said that the STP itself has an inherent robust feature against various variability of the CMOS transistor. This smart feature implies that the STP has a potentiality to operate without malfunction under the dynamic voltage scale control, from super- to near-/sub-threshold voltage. This is because the voltage scaling could be regarded as a sort of voltage variation, as long as the supply voltage is gently scaled [4].

In order to achieve significant power reduction, this study aims to investigate an optimal circuit design of the STP operating at near-threshold voltage. At first, the behaviors of typical STP circuit is characterized under various voltage and temperature conditions. After that, optimal transistor sizing method for self-timed data-transfer control circuit, called C-element, is proposed. Finally, widely operating conditions of the self-timed pipeline register including the C-element which is designed by 65nm CMOS transistors, will be revealed through SPICE simulation results.

ISBN: 1-60132-444-8, CSREA Press ©

2. Low-Voltage Characteristics of STP

The influences of PVT variations appear to be worse relatively when lowering the supply voltage of general CMOS circuit to sub-threshold region, in comparison with recommended supply voltage operation. The self-timed pipeline circuit (STP) has a potentiality to moderate those influences because of its localized wiring between pipeline stages.

Figure 1 shows a basic structure of self-timed pipeline where every pipeline stage is composed of a data latch, function logic, and data-transfer control circuit called C-element shown in figure 2.

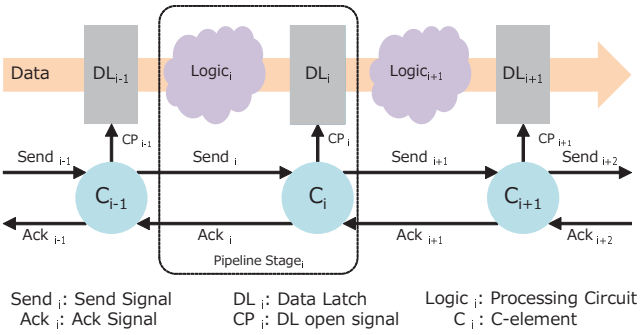


Fig. 1: Basic structure of STP.

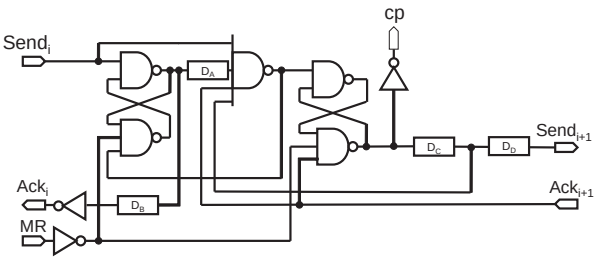


Fig. 2: Basic logic circuit of C-element.

Figure 3 illustrates a typical timing sequence of the STP. The STP circuit usually operates as follows.

(1) Master reset: At first, a master-reset signal initializes all states of C-elements. At that time, every handshake (send and ack) signal is set to 1 (high).

(2) Ready to transfer data: When a data packet hold at the stage$_{i-1}$ is ready to be transferred, its data-latch open signal CP$_{i-1}$ is asserted and send$_i$ signal is set to 0 (low) to begin to transfer the data. Then, the stage$_i$ negates ack$_i$ as an acknowledge signal.

(3) Completion to transfer data: Then, the stage$_{i-1}$ asserts send$_i$ for completion of the data transfer. After that, the stage$_i$ asserts the CP$_i$ to receive the data and the stage then asserts ack$_i$. It implies that the stage$_i$ is ready to transfer the data to the next stage$_{i+1}$ as well.

Every pipeline stage in the STP operates with the above hand-shake protocol. Therefore, even if an adjacent pipeline

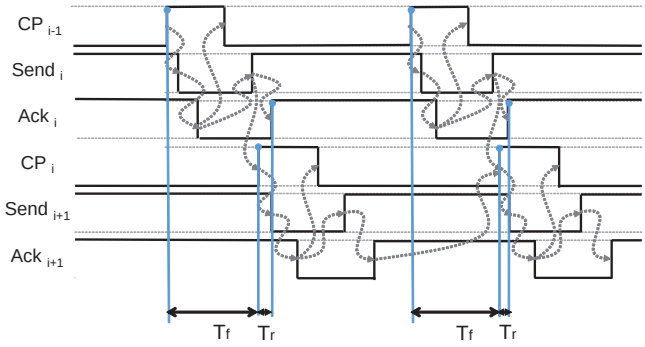


Fig. 3: Typical timing chart of C-element.

stage might be delayed due to some PVT variation, the data can be adaptively transferred based on the hand-shake protocol, where the time from the positive edge of the CP$_i$ to that of the CP$_{i+1}$ represents the data-transfer time between the two stages and where the time between a positive edge of the CP$_i$ to the next positive edge of that represents the period of consecutive data transfer at the stage$_i$. Furthermore, it is noted that the STP stage realized by the CMOS transistors consumes active power only when the stage is transferring and processing the data. This means that the STP does not require additional clock gating technique to save its dynamic power of clock distribution.

The article [1] reported critical issues under near-threshold voltage, where the nominal voltage is 1.1 V and the near-threshold voltage is 400 mV. In this case, performance of the fanout-of-four inverter in industrial 40 nm CMOS is degraded in 1/10th. Performance variations of transistor switching speed increases in 20x, in which process variations increase in 5x, the sensitivity due to temperature and supply ripple increase in 2x respectively. Function failure rate increases in 5 orders of magnitude.

Those issues are related to characteristics of individual CMOS gate composing the CMOS circuit. In terms of circuit design for the pipeline operating at near-threshold voltage region, there are two counter methods, synchronous and asynchronous. As for the synchronous pipeline, its clock cycle must be adjusted to the longest delay in the pipeline, i.e., it might be occurred under the worst case in PVT variations. The longest path such as clock distribution tree or forwarding mechanism might exist among several pipeline stages. In such case, the worst case design requires larger margin to reduce its function failure rate in sacrifice of the performance.

As for the self-timed pipeline, a kind of asynchronous pipeline, the influence of performance variations and the function failure rate can be limited to a single pipeline stage and suppressed in comparison of that of the synchronous pipeline. Thanks to the hand-shake protocol of the STP, the operating speed of the C-element alters autonomously even if the PVT variations affect the delay time of the CMOS cells

composing the C-element. As a result, the average pipeline throughput is restricted by the slowest pipeline stage in the STP. As with the synchronous pipeline, setup and hold time of data latches must be guaranteed in the STP so that each delay time of send and ack signals have to be adjusted to the longest latency of the critical path in its corresponding stage. Furthermore, the SR latch in the C-element might oscillate or fall into a meta-stable state when both input signals of the RS latch change within a short time. A systematic design to avoid those malfunction under near-threshold voltage will be required with consideration of PVT variations.

Furthermore, in order to realize the dynamic voltage scaling including sub-threshold voltage region, lower limit of supply voltage must be settled. This is because the energy minimum voltage around the sub-threshold region must exist due to the exponential increase of leakage power beyond the reduction of switching power.

3. NTC-Oriented C-Element

In order to ensure correct operation of the C-element under near-threshold voltage, there are two major issues to be solved. The first one is that the setup and hold time violation of the data latch must be avoided even if the transistor delay increases exponentially. The second one is that the electric potential on cyclic paths in the C-element must be kept steady at high potential, VDD, or low potential, VSS.

The first problem is solved by adjusting the delay buffers, D_a, D_b, D_c, and D_d, of the C-element.

In general, forwarding latency required to transfer valid data from one set of data-latches to a set of data-latches in the succeeding stage is calculated by the sum of response time of the data latch τ_q, delay time of a critical path in the logic τ_{cp}, and setup time of the data latch τ_{setup}. Thus, handshake time of STP must be adjusted to the forwarding latency. As explained in Section 2, the original STP circuit operates based on the 4-phase handshake protocol so that latency time T_f required from the first to the third phase of the protocol has to exceed the forwarding latency. After completing the data-transfer, the data latch has to receive the next data correctly. Therefore, backward latency time T_r required for the fourth phase must exceed hold time of the data latch τ_{hold}.

$$T_f \geq \tau_q + \tau_{cp} + \tau_{setup} \quad (1)$$

$$T_f \geq \tau_{hold} \quad (2)$$

Using the two parameters and, it is then possible to define pipeline throughput as $1/(T_f + T_r)$ and pipeline efficiency as $T_f/(T_f + T_r)$. Pipeline throughput is a measure of packet flow rate through the pipeline. Pipeline efficiency is the proportion of net processing time spent on packet processing in terms of pipeline throughput.

The lower the supply voltage is set, the weaker the driving power of the output port of CMOS logic cell becomes. Thus, it is hard for output electric potential of the CMOS cell to keep steady VDD or VSS under the near-threshold voltage. The following techniques can be introduced as a candidate solution to tune the driving power of the related CMOS cells.

1) Replacement: to replace the target cell to a stronger cell which generates more driving current.
2) Multiplication: To multiplicate nMOS/pMOS transistors constructing the target cell.
3) Widening: To widen gate-width of nMOS/pMOS transistors of the target cell.

In terms of the technological difficulty of those tuning techniques, the first one is tried to be applied to C-element circuit design by using 65nm SOI-CMOS process. At first, the basic logic of the C-element is synthesized with zero capacitive load for every cell. After that, the critical cells are replaced to suitable stronger cells.

Figure 4 is a diagram showing the drivability of CMOS cells in the C-element which is synthesized with zero capacitive load for every cell. In the figure, each number described on individual cell denotes the relative drivability of the cell.

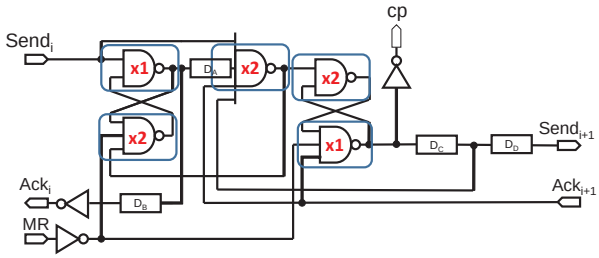


Fig. 4: C-element synthesized with zero capacitive load.

Figure 5 shows a C-element CMOS circuit modified with large cell replacement.

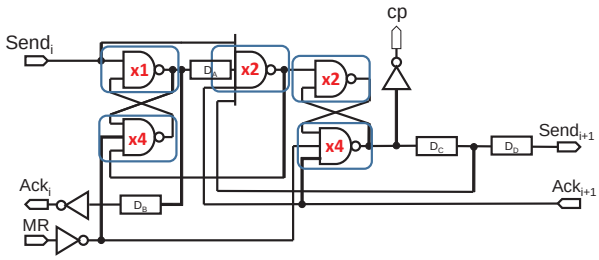
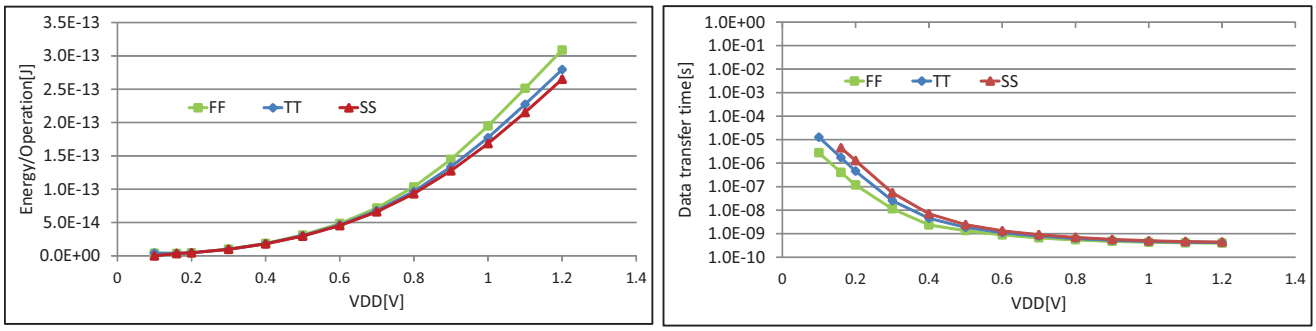


Fig. 5: C-element modified with large cell replacement.

4. Evaluation

The robust operation of the STP register with the modified C-element is verified by using SPICE simulation with the 65nm SOI-CMOS process libraries.

As a preliminary verification, the two kinds of single C-element CMOS circuit designed by using standard threshold voltage transistor (SVT) and low threshold one (LVT) are



(a) energy/operation. (b) data transfer time.

Fig. 6: Performance characteristics of SVT-based C-element ($-20°$C).

simulated respectively. The operating conditions are assumed as listed in table 1. The supply voltage range is set from 0.1 V to 1.2 V in 0.1 V steps, the temperature is $-20°$C, $25°$C, and $75°$C, the process corner of nMOS and pMOS transistors is slow-slow (SS), typical-typical (TT), and fast-fast (FF).

Table 1: Transistors and PVT conditions in SPICE.

Transistors	SVT, LVT
Process Corner	SS, TT, FF
Voltage	0.1V to 1.2V
Temperature	$-20°$C, $25°$C, $75°$C

In the SPICE simulation, the data transfer time (T_t) from receiving a data packet to sending the packet is measured as well as the average power during the data-transfer time (P_t). Precisely describing that in the figure 3, it is the time from when send_{i-1} is negated to when ack_i is asserted. That is, T_t indicates about $T_f \times 2$. Therefore, the energy (E_t) consumed in the C-element to operate a data packet can be calculated by $E_t = P_t \times T_t$. In this measurement, the effective signal level is set to over 90 % of VDD as the high level signal and to under 10 % of VDD as the low level signal. The used SPICE simulator is Hspice, Synopsys Inc.

At first, the measurement results of the SVT-based C-element circuit at -20°C, 25°C, and 75°C is shown in figure 6, 7, and 8 respectively. The SVT-based C-element can operate well without function failure, except for case of 0.1 V and the SS process condition. As discussed in section 2, the energy/operation decrease in proportion to the square of VDD and the data-transfer time in case of 0.1 V is degraded about 5 order of magnitude in comparison with the 1.2 V.

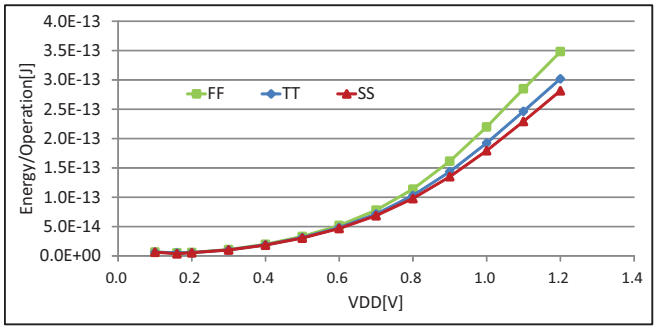
At first, the measurement results of the LVT-based C-element circuit at $-20°$C, $25°$C, and $75°$C is shown in figure 9, 10, and 11 respectively. Compared with the SVT circuit, the LVT-based C-element can hardly operate in some cases of 0.1 V.

As discussed in section 2, there is the energy minimum point around the near-threshold voltage due to the exponential increase of leakage power. However, the minimum point cannot be observed in the preliminary measurements. Therefore, the supply voltage range around the sub-threshold is finely set from 0.1 V to 0.2 in 10 mV steps and the energy minimum voltage is searched in case of 25°C. Figure 12 shows the energy/operation around the energy minimum voltage, 0.16 V, in both SVT and LVT circuits. This means that the NTC operating at the supply voltage lower than 0.16 V can bring no advantageous effect to the C-element.

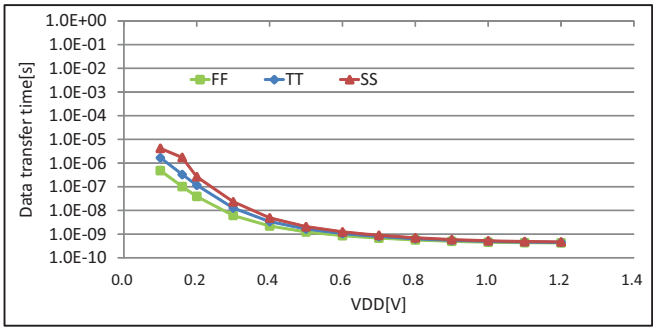
5. Conclusion

Near-threshold computing (NTC) technique for operating the circuit below the threshold or less called is expected to be a promising scheme to reduce power consumption significantly. There is challenging issues on the near-/sub-threshold voltage operation, in which function failure must be eliminated and performance degradation and variations must be suppressed.

This paper focuses on the self-timed pipeline (STP) circuit which has an inherent robust feature against the PVT variations and discusses the low voltage characteristics of the CMOS STP circuit. After that, an optimal transistor sizing method for the STP operating at near-threshold voltage is proposed and verified by using SPICE simulation of 65 nm CMOS process libraries (SVT and LVT) in various conditions, where VDD is 0.1 V to 1.2 V in 0.1 V steps, temperature is $-20°$C, $25°$C, and $75°$C, and the process corner is SS, TT, and FF. The lower limit voltage for minimizing the energy/operation was 160 mV for the C-element realized by either SVT or LVT transistor. The simulation results reported in the paper are limited to the single C-element CMOS circuit, so that the practical STP circuit such as DDMP should be designed by applying the proposed method in the paper and evaluated under near-threshold voltage region. Those works are still remained as further investigations in our research project.

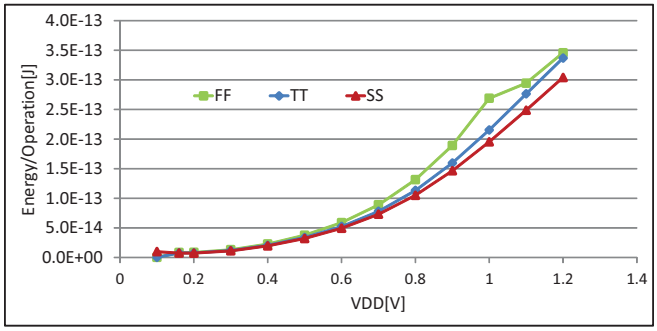


(a) energy/operation.

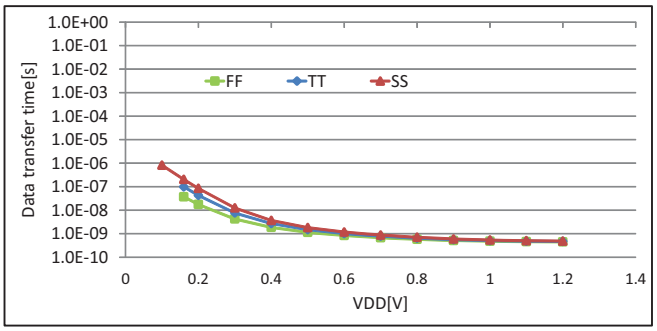


(b) data transfer time.

Fig. 7: Performance characteristics of SVT-based C-element (25°C).

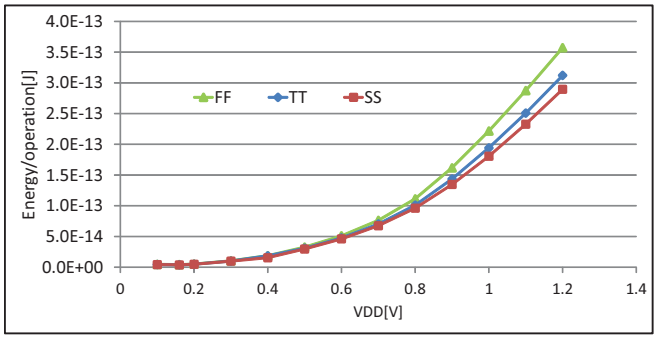


(a) energy/operation.

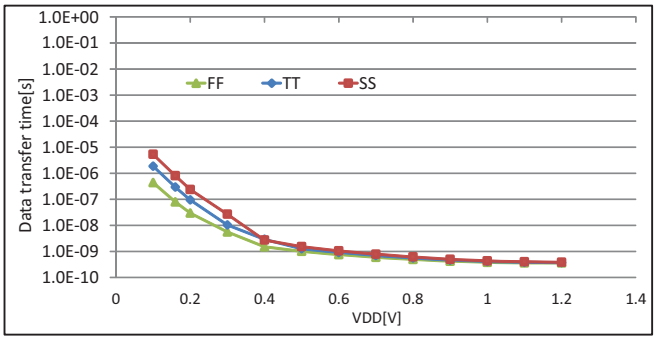


(b) data transfer time.

Fig. 8: Performance characteristics of SVT-based C-element (75°C).

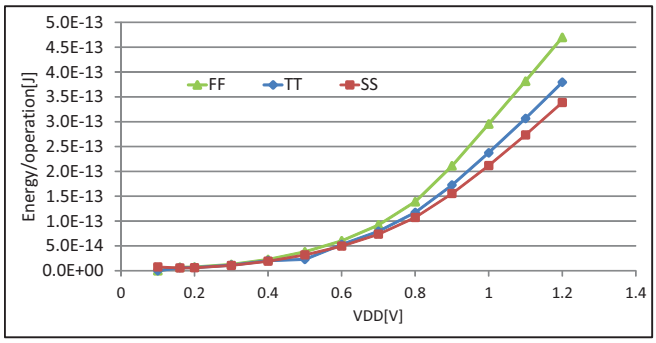


(a) energy/operation.

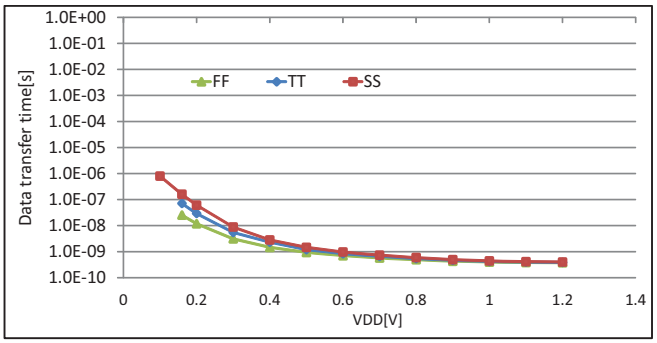


(b) data transfer time.

Fig. 9: Performance characteristics of LVT-based C-element (−20°C).

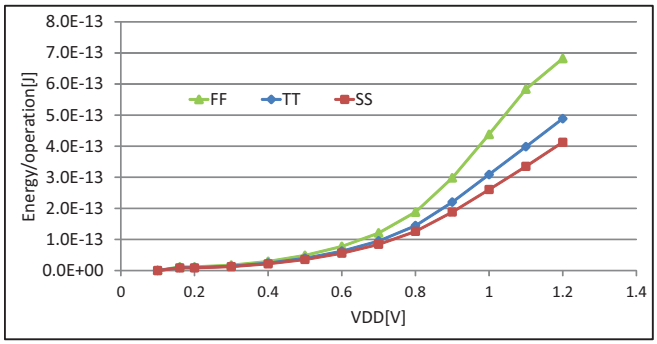


(a) energy/operation.

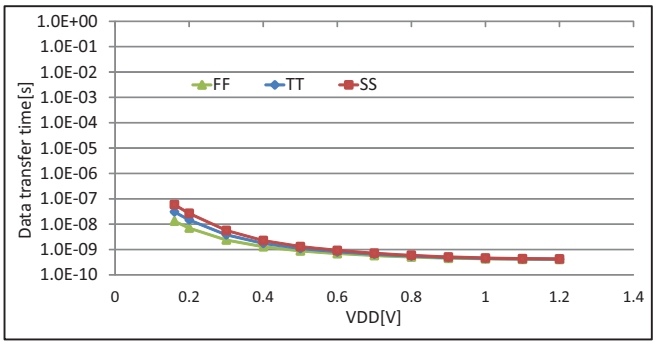


(b) data transfer time.

Fig. 10: Performance characteristics of LVT-based C-element (25°C).



(a) energy/operation.



(b) data transfer time.

Fig. 11: Performance characteristics of LVT-based C-element (75°C).

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.

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.

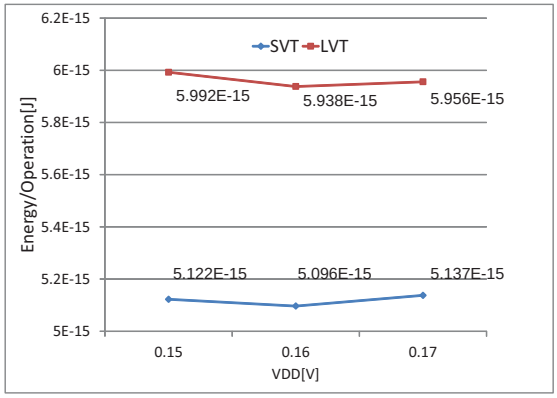


Fig. 12: Energy minimum voltage of C-element.

References

[1] Ronald G. Dreslinski, Michael Wieckowski, David Blaauw, Dennis Sylvester, Trevor Mudge, "Near-Threshold Computing: "Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," Proceedings of the IEEE, Vol. 98, No. 2, pp. 253–266, Feb. 2010.

[2] I-Chyn Wey, Po-Jen Lin, Bing-Chen Wu, Chien-Chang Peng, and Pin-Hsi Lin, "Near-threshold-voltage circuit design: The design challenges and chances," International SoC Design Conference (ISOCC), pp. 138–141, Nov. 2014.

[3] Hiroaki Terada, Soichi Miyata, and Makoto Iwata, "DDMP's: Self-Timed Super-Pipelined DataDrivenProcessors," Proceedings of the IEEE, Vol. 87, No. 2, pp. 282–296, Feb. 1999.

[4] Kei Miyagi, Shuji Sannomiya, Makoto Iwata, and Hiroaki Nishikawa, "Low-PoweredSelf-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling," International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA'13), pp. 618–624, July 2013.

SESSION

WORKSHOP: MATHEMATICAL MODELING AND PROBLEM SOLVING, MPS

Chair(s)

Prof. Hayaru Shouno

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

Parallel Processing for Density-based Spatial Clustering Algorithm using Complex Grid Partitioning and Its Performance Evaluation

Tatsuhiro Sakai[1,2], **Keiichi Tamura**[1], **Kohei Misaki**[1], **and Hajime Kitakami**[1]
[1]Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan
[2]JSPS Research Fellow, Japan

Abstract— *Density-based spatial clustering algorithms, which have been well studied in database domains, are based on densities of geospatial data. Recently, the sizes and volumes of spatial databases have been increasing not only because of the popularity of geographical data, but also because of the popularity of geosocial media. Therefore the speedup for the processing of density-based spatial clustering algorithms is one of the most important challenges in many different application domains. In this paper, we propose a new parallelization model on a multi-core CPU using the spatial partition method for DBSCAN, which is one of the most fundamental algorithms for density-based spatial clustering. The new parallelization model utilizes a data replication technique and complex grids for the parallel processing of DBSCAN, in order to improve the speedup performance of parallel processing. The experimental results show that our new model outperforms a conventional data parallelization model.*

Keywords: density-based spatial clustering, spatial database, parallel processing, multi-core CPU, complex grid

1. Introduction

With the increasing interest in big data, the use of geospatial databases for ICT (information and communications technology) has received much attention in recent years. The clustering technique for geospatial data is one of the most well studied techniques because it allows us to reveal spatial relevance of geospatial data. To extract clusters for geospatial data, a huge number of spatial clustering techniques have been proposed. Clustering techniques for geospatial data differ from traditional clustering techniques (e.g., k-means method) only in that clusters for geospatial data do not always form circles. For example, contaminated land sites form arbitrary shapes from a satellite observation.

A density-based spatial clustering algorithm is one of the simplest but most robust clustering techniques for geospatial data. The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm was first introduced by Ester et al. [1][2], and it applies a density-based concept of spatial clusters. Spatial clusters are recognized by analyzing the density of data points. Areas with a high density of data points are spatial clusters, whereas areas with a low density are not. DBSCAN can discover spatial clusters with arbitrary shapes. Therefore, many methods apply this algorithm to geospatial databases because spatial clusters in geospatial databases are not circular. The key concept of the DBSCAN algorithm is that for each data point in a spatial cluster, the neighborhood with a user-defined radius has to contain at least a minimum number of points; i.e., the density in the neighborhood must exceed some predefined threshold.

In this paper, we focus on the speedup of the DBSCAN algorithm. The goal of this study is to develop a novel parallel-processing parallelization model for DBSCAN on a multi-core CPU. Currently, PCs and workstations have one or more multi-core CPUs. A multi-core CPU is a single microprocessor with two or more independent CPU cores on a die, which are the units that read and execute program instructions. It is necessary to develop an efficient parallelization model for spatial clustering techniques on a multi-core CPU.

The main contributions of this study are as follows:

- To parallelize the DBSCAN algorithm, the proposed parallelization model is based on the master-worker model using data parallelism. The DBSCAN algorithm has spatial independence at the data level, because a spatial cluster can be extracted independently of the extraction of other spatial clusters. In data parallelism, an entire geospatial database is divided into two or more sub-databases called partitions using grid partitioning. A partition is assigned to a worker thread on a CPU core, and it is executed on a worker thread.
- To extract a spatial cluster that is spread over several grids, we have to calculate the density of geospatial data near the boundary of the grids correctly. Each gird contains a replication of geospatial data beyond the borders of the grid. This replication allows us to calculate the density of geospatial data near the boundary of the grids. Moreover, several spatial clusters extracted from adjacent grids are merged if they are connected.
- To reduce the number of replications, the proposed parallelization model utilizes complex grid partitioning. One of the disadvantages of grid partitioning is the increase in the number of replications due to merging. In complex grid partitioning, a complex grid is composed of highly dense adjacent grids. Composing a complex grid reduces the number of grids; therefore, the number

of replications decreases compared with simple grid partitioning. This improves the overall performance of the parallel processing.

The rest of this paper is organized as follows. In Section 2, related work is reviewed. In Section 3, a density-based spatial clustering algorithm and its algorithm are presented. In Section 4, we propose a novel parallelization model for the parallel processing of DBSCAN. In Section 5, we report the experimental results. In Section 6, we conclude the paper.

2. Related work

Recently, the parallelization model of DBSCAN for speedup of its algorithm has been proposed [3][4] as the sizes and volumes of spatial databases have been increasing because of the popularity of geographical data [5]. Xu et al. [3] proposed the parallelization model of DBSCAN on a cluster computer. The method divides an entire geospatial dataset using grid division of the space index, and each computer performs clustering for the divided geospatial data. It is possible to perform parallel processing of clustering by using multiple computers. Moreover, research on parallel processing of DBSCAN has also been conducted on the new computing platform, example, the parallelization model using graphics processing unit (GPU) [6][7] and MapReduce [8].

Misaki et al. [9] proposed a parallelization model for the parallel processing of DBSCAN on a multi-core CPU. In previous model, a geospatial database is divided into two or more sub-databases called partitions using grid partitioning on the basis of data parallelism. Each CPU core performs the same processing on different partitions. In the experimental results, the previous model showed the effectiveness of parallel processing in terms of speedup; however, the process for each grid partitioning is time consuming because each grid is increased in the number of replications. The proposed new model reduce the processing time because decreasing the number of replications by using complex grid partitioning.

3. DBSCAN

In this section, the definitions of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) are briefly reviewed.

3.1 Definitions

In DBSCAN, the ϵ-neighborhood of a geospatial data is defined as geospatial data in the neighborhood of a user-defined given radius ϵ.

Definition 1 (ϵ-neighborhood $GSN_\epsilon(gsd)$) The ϵ-neighborhood of a geospatial data gsd_p denoted by $GSN_\epsilon(gsd_p)$, is defined as

$$GSN_\epsilon(gsd_p) = \{gsd_q \in GSD | dist(gsd_p, gsd_q) \leq \epsilon\}, \quad (1)$$

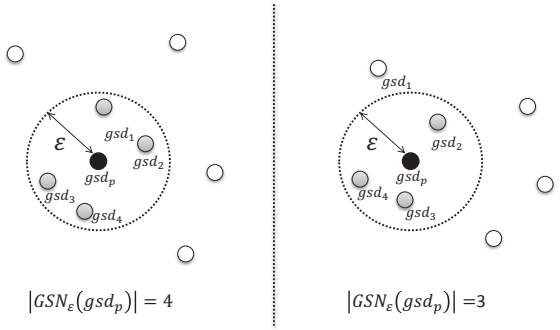


Fig. 1: Example of definition 1, 2, and 3

where the function *dist* returns the distance between geospatial data gsd_p and geospatial data gsd_q.

An example of the ϵ-neighborhood of gsd_p is shown on Fig. 1. On the left side of the figure, there are four geospatial data in the ϵ-neighborhood of gsd_p. Moreover, in the right side of Fig. 1, there are three geospatial data in the ϵ-neighborhood of gsd_p.

Definition 2 (Core geospatial data, Border geospatial data) A geospatial data gsd_p is called a core geospatial data if there is at least the minimum number of geospatial data, $MinGSD$, in the ϵ-neighborhood $GSN_\epsilon(gsd_p)$ ($|GSN_\epsilon(gsd_p)| \geq MinGSD$). Otherwise, ($|GSN_\epsilon(gsd_p)| < MinGSD$), gsd_p is called a border geospatial data.

Suppose that $MinGSD$ is set to four. A geospatial data gsd_p on the left side of Fig. 1 is a core geospatial data, because there are four geospatial data in $GSN_\epsilon(gsd_p)$. A geospatial data gsd_p on the right side of Fig. 1 is a border geospatial data because the number of geospatial data in $GSN_\epsilon(gsd_p)$ is less than $MinGSD$.

Definition 3 (Density-based directly reachable) Suppose that a geospatial data gsd_q is in the ϵ-neighborhood of gsd_p. If the number of geospatial data in the ϵ-neighborhood of gsd_p is greater than or equal to $MinGSD$, i.e., if $|GSN_\epsilon(gsd_p)| \geq MinGSD$, gsd_q is density-based directly reachable from gsd_p.

On the left side of Fig. 1, geospatial data gsd_p is a core geospatial data, because $|GSN_\epsilon(gsd_p)| \geq MinGSD$. Geospatial data gsd_1, gsd_2, gsd_3, and gsd_4 are in the ϵ-neighborhood of gsd_p. These four geospatial data are density-based directly reachable from gsd_p. On the other hand, on the right side of Fig. 1, geospatial data gsd_p is a border geospatial data; i.e., it is not $|GSN_\epsilon(gsd_p)| \geq MinGSD$. These three geospatial data are not density-based

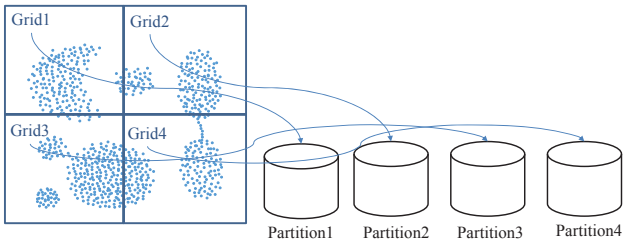


Fig. 2: Grid partitioning

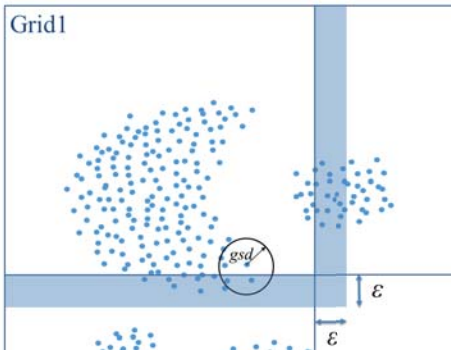


Fig. 3: Geospatial data around the border of a grid

directly reachable from gsd_p although geospatial data gsd_2, dgs_3, and gsd_4 are in the ϵ-neighborhood of gsd_p.

Definition 4 (Density-based reachable) Suppose that there is a geospatial data sequence $(gsd_1, gsd_2, \cdots, gsd_n)$, and the $(i+1)$-th geospatial data gsd_{i+1} is density-based directly reachable from the i-th geospatial data gsd_i. The geospatial data gsd_n is then density-based reachable from gsd_1.

Definition 5 (Density-based connected) Suppose that geospatial data gsd_p and gsd_q are density-based reachable from geospatial data gsd_o. If $|GSN_\epsilon(gsd_o)| \geq MinGSD$, we denote that gsd_p is density-based connected to gsd_q.

A density-based spatial cluster consists of two types of data: core geospatial data, which are mutually density-based reachable; and border geospatial data, which are density-based directly reachable from the core geospatial data. A density-based spatial cluster is defined as follows.

Definition 6 (Density-based spatial cluster) A density-based spatial cluster (GSC) in a geospatial data set GSD that satisfies the following restrictions:

(1) $\forall gsd_p,\ gsd_q \in GSD$, if and only if $gsd_p \in GSC$ and gsd_q is density-based reachable from gsd_p, and gsd_q is also in GSC.

(2) $\forall gsd_p,\ gsd_q \in GSC$, gsd_p is density-based connected to gsd_q.

3.2 Algorithm

To extract density-based spatial clusters, approximate core geospatial data are appended recursively. A density-based spatial cluster is created using a core geospatial data first, and neighbors of the core geospatial data are then added to the cluster. For each geospatial data gsd_i in GSD, the following steps are executed. If gsd_i is a core geospatial data according to Definition 2, it is assigned to a new spatial cluster GSC, and all the neighbors are queued to a candidate queue Q for further processing. The processing and assignment of geospatial data to the current spatial cluster continue until Q is empty. The next geospatial data is then dequeued from Q. If the dequeued geospatial data is not already assigned to the current spatial cluster, it is assigned to the current spatial cluster. The ϵ-neighborhood of the dequeued geospatial data is then queued to Q, which puts input geospatial data into Q if they are not already in Q.

4. Proposed Method

In this section, we propose a new parallelization model for the parallel processing of DBSCAN on a multi-core CPU.

4.1 Data Parallelism using Grid Partitioning

In this study, we focus only on the data-parallelism-based master-worker model on a multi-core CPU. In data parallelism, a geospatial database is divided into two or more sub-databases called partitions. The extraction of spatial cluster can be performed in parallel using these partitions. In a multi-core CPU environment, each CPU core performs the same processing on different partitions. The proposed parallelization model utilizes grid partitioning to divide the whole database. The whole space is divided into boundary boxes called grids. Each data in the geospatial database is assigned to a grid that includes the data. Fig. 2 shows an example of grid partitioning. In this example, the whole area is divided into four subareas called grids. A set of geospatial data is assigned to a partition.

4.2 Data Replication

In the grid partitioning framework, we cannot determine whether a geospatial data near the border of a grid is core geospatial data or not using only the data set of the grid that contains geospatial data. In Fig. 3, even though a geospatial data gsd near the border of Grid 1 is a core geospatial data the geospatial data gsd is not identified as a core geospatial data, because some its neighbors are located in Grid 3. To determine whether a geospatial data near the border of a grid is core geospatial data or not, all grids extend only ϵ. Therefore, adjoining grids overlap. In Fig. 3, Grid 1 contains

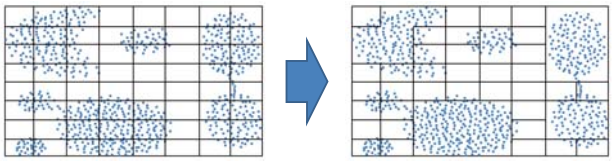


Fig. 4: Example of complex grid partitioning

not only a set of geospatial data located in its area but also a set of geospatial data located in the area shown with a transmission color. The set of geospatial data located in the area is a replication.

4.3 Complex Grid

To reduce the number of replications, the proposed parallelization model utilizes complex grid partitioning. Fig. 4 shows an example of complex grid partitioning. The left side of Fig. 4 shows simple grid partitioning. One of the disadvantages of simple grid partitioning is the increase in the number of replications due to merging. In complex grid partitioning on the right side of Fig. 4, a complex grid is composed of highly dense adjacent grids. Composing a complex grid reduces the number of grids; therefore, the number of replications decreases compared with simple grid partitioning. This improve the overall performance of the parallel processing.

Moreover, if data there is concentrated in one of the grids, the loads are not distributed. Then, if the number of geospatial data in a grid is larger than the number of the entire geospatial data divided by the number of workers, the grid is further divided.

The steps are the processing steps of creating complex grids.

(1) For each grid, the number of geospatial data in the grid is counted.
(2) For each grid, if the number of geospatial data is larger than the number of all geospatial data divided by the number of workers, the grid is further divided.
(3) For each grid, if the number of geospatial data is larger than twice the average of the number of geospatial data, the grid is labeled a dense grid. Otherwise the grid is labeled a non-dense grid.
(4) Each dense grid combines with the adjoining dense grids up to the number of all geospatial data divided by the number of workers. A set of dense grids then forms a complex grid.
(5) Each non-dense grid forms a complex grid.

4.4 Dynamic Load Balancing

The proposed model utilizes the task pool to distribute the loads. A processing of spatial clustering for a partition associated with a complex grid is referred to as a task. The master thread manages tasks using the task pool. Each worker perform clustering after obtaining a task from the task pool. Finally, if the task pool is empty and each worker finishes task processing, the entire process is completed.

4.5 Merging Clusters

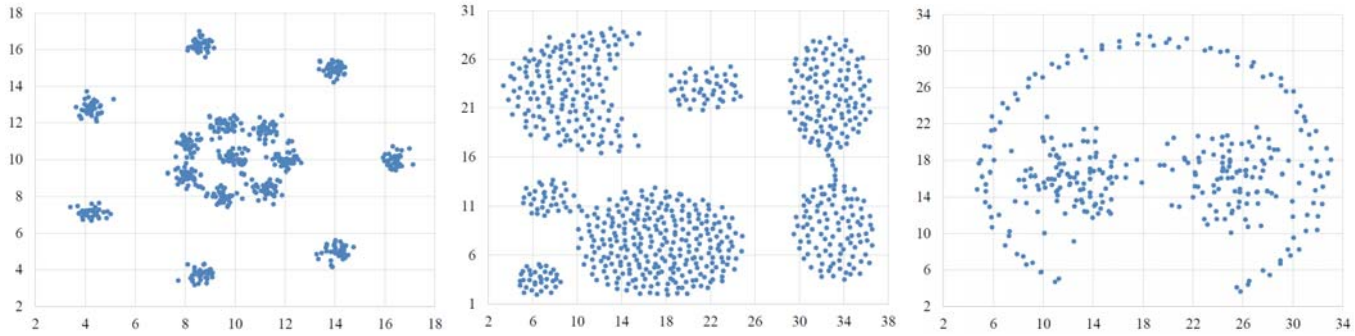
To extract a spatial cluster that spread over several grids, the proposed model merges extracted spatial clusters from each partition. First, the proposed model obtains adjacent grids information from the area number of each partition and the division points for each dimension. On the basis of the information from the adjacent grids, the proposed model extracts overlapping clusters from spatial clusters in a grid and spatial clusters in grids adjacent to its grid. It is possible to extract overlapping spatial clusters because of data replication. The extracted overlapping spatial clusters are merged, and those spatial clusters become one spatial cluster. The proposed model can obtain the same as clustering results using no parallel method by the merging clusters.

4.6 Algorithm

The processing steps of the master thread and the worker threads are as follows.

A) Master Thread

(1) The master thread received a geospatial database GSD, and parameters p, ϵ, and $MinGSD$.
(2) The whole space is divided into p subspaces for each dimension. A separated space is a grid. For each geospatial data $gsd \in GSD$, the master thread assigns gsd to a grid. For each grid, the number of geospatial data is calculated.
(3) If the number of geospatial data in a grid is larger than the number of the entire geospatial data divided by the number of workers, the grid is further divided.
(4) The master thread calculates $GSN_\epsilon(gsd)$ for geospatial data.
(5) The master thread generates complex grids from a set of grids.
(6) The master thread creates a tack pool.
(7) The complex grid is referred to as a partition. The master puts a partition in the task pool.
(8) The master thread creates t worker threads.
(9) The master thread receives a request for task assignment from a worker thread.
(10) If the task pool is not empty, the master thread pops a task from the task pool and sends it the task to the worker thread. Otherwise, the master worker sends a wait message to the worker thread.
(11) If the master thread has sent wait messages to all the worker threads, the processing step returns to (12). Otherwise, the processing step returns to (9).



R15 *Aggregation* *Pathbased*

Fig. 5: Distribution for each dataset

(12) The master thread destroys the task pool.
(13) The master thread sends an end message to each worker thread.
(14) The master thread receives clustering results from all the worker threads.
(15) The master thread merges clusters that spread over several grids.
(16) The master thread returns a set of spatial clusters.

B) Worker Thread

(1) The worker thread sends a task assignment request to the master thread.
(2) If the worker thread receives a wait message, the processing step goes to (4). Otherwise, the worker thread receives a task from the master thread.
(3) The worker thread extracts spatial clusters from a partition associated with the task using the DBSCAN algorithm. The worker thread puts the clustering results of the assigned task in a result pool. The processing step returns to (1).
(4) The worker thread waits for an end message from the master thread.
(5) The worker thread sends a set of spatial clusters to the result pool.

5. Performance Evaluation

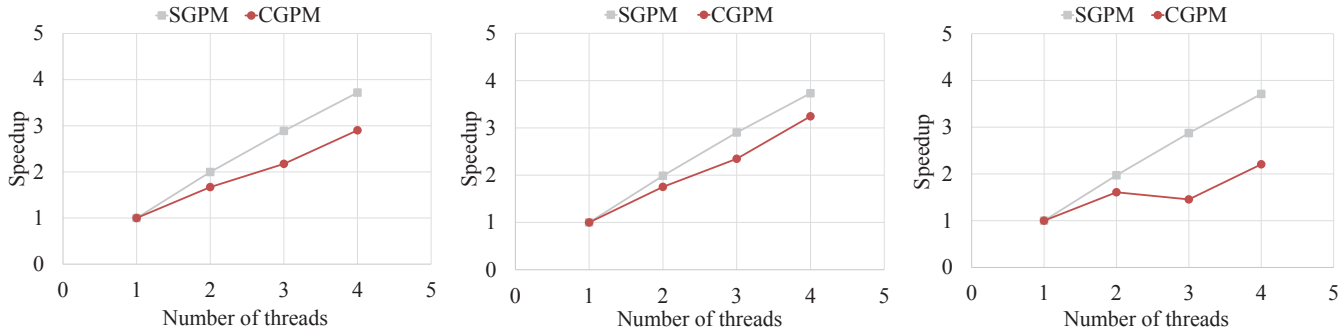
To evaluate the proposed model for parallel processing for DBSCAN on a multi-core CPU, we employed actual data sets. We implemented the proposed model for parallel processing using multi-thread. We conducted an experiment with a PC with the following specifications: CPU INTEL XEON E5-1270 V2 (number of core = 4) @3.5 GHz, memory:32GB. In the experiment, we used three types of datasets: *R15*, *Aggregation* and *Pathbased*. Fig. 5 shows the datasets. For, each dataset, we expanded the number of geospatial data to approximately 100,000 by increasing geospatial data around each geospatial data artificially.

The parameters were set to the number of initial grid divisions of each dimension $p = 8$. Moreover, the parameters were set to $\epsilon = 0.5$ and $MinGSD = 3000$ with *R15*, $\epsilon = 1.8$ and $MinGSD = 1550$ with *Aggregation*, and $\epsilon = 1.6$ and $MinGSD = 2300$ with *Pathbased* so as to be correct clustering results. We compared the results of changing the number of worker threads t from 1 to 4.

In the experiments, we measured the processing time of DBSCAN using the proposed model, which utilizes complex grid partitioning (denoted by CGPM), and the previous model, which utilizes simple grid partitioning (denoted by SGPM). Fig. 6 shows the speedup for each dataset. The vertical axis represents the speedup ratios, while the horizontal axis shows the number of worker threads. In SGPM, the speedup ratio reaches up to approximately 3.7 times for each datasets. In CGPM, the speedup ratio reaches up to approximately 3 times using *R15* and *Aggregation*. However, the speedup ratio reaches up to approximately 2 times using *Pathbased*. The previous model obtained a higher speedup compared to the proposed model.

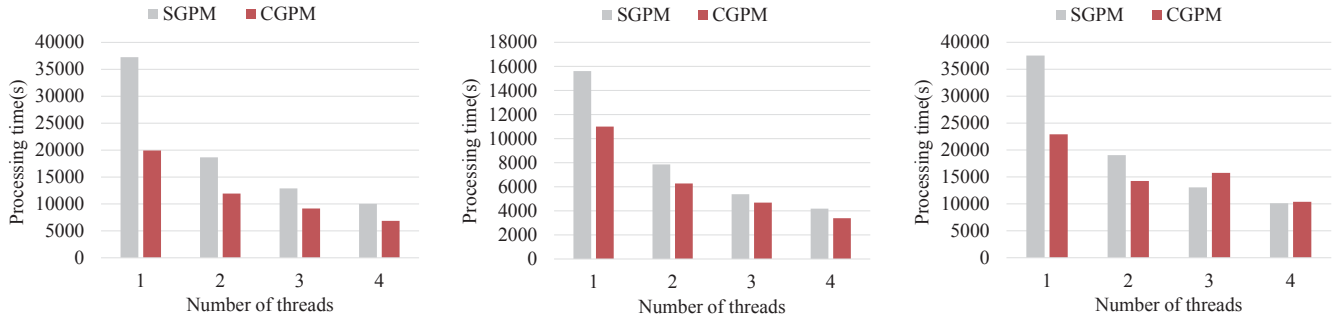
In addition, Fig. 7 shows the processing time for each dataset. The processing time of CGPM are faster than that of SGPM using *R15* and *Aggregation*, as shown in Fig. 7. This is because the number of replication data is reduced by the complex grid partition. The processing time with $t = 3$ and 4 of CGPM is worse then that of SGPM using *Pathbased*. It is assumed that deviation of the loads occurred by conbining of dense grid.

The previous model obtained a higher speedup compared to the proposed model, as shown in Fig 6. It is assumed that the number of replication data with worker threads $t = 4$ is more than the number of replication data with worker threads $t = 1$, because the less the number of threads, the more combining of dense grids increase. We then conducted an experiment with chenging the condition of combining of the dense grids. The condition is combining up to the number of all geospatial data divided by four. That is, we set the same condition for each the number of worker thread. Fig 8 and Fig 9 show the speedup and processing time, respectively.



Speedup of *R15* Speedup of *Aggregation* Speedup of *Pathbased*

Fig. 6: Speedup for each dataset



Processing time of *R15* Processing time of *Aggregation* Processing time of *Pathbased*

Fig. 7: Processing time for each dataset

The speedup ratio of CGPM is much the same as the speedup ratio of SGPM using *R15* and *Aggregation*. The processing time of CGPM is faster than the processing time of SGPM, as shown in Fig 9. However, the processing time of CGPM in Fig 9 is slower than the processing time of CGPM in Fig 7. We are necessary to develop a method for automatic setting of the combining condition for each the number of worker thread.

Datasets whose data distribution exhibits a small deviation on the coordinate space. We then created datasets whose data distributions exhibited large deviations. We artificially expanded the number of geospatial data of each dataset to about 100,000. *A_R15*, *A_Aggregation* and *A_Pathbased* are half of the geospatial data distributed in the lower left. The parameters were set as $p = 8$ and $t = 4$. The parameters were set as $\epsilon = 0.2$ and $MinGSD = 3000$ with *A_R15*, $\epsilon = 1.0$ and $MinGSD = 4650$ with *A_Aggregation*, and $\epsilon = 1.4$ and $MinGSD = 6900$ with *A_Pathbased*.

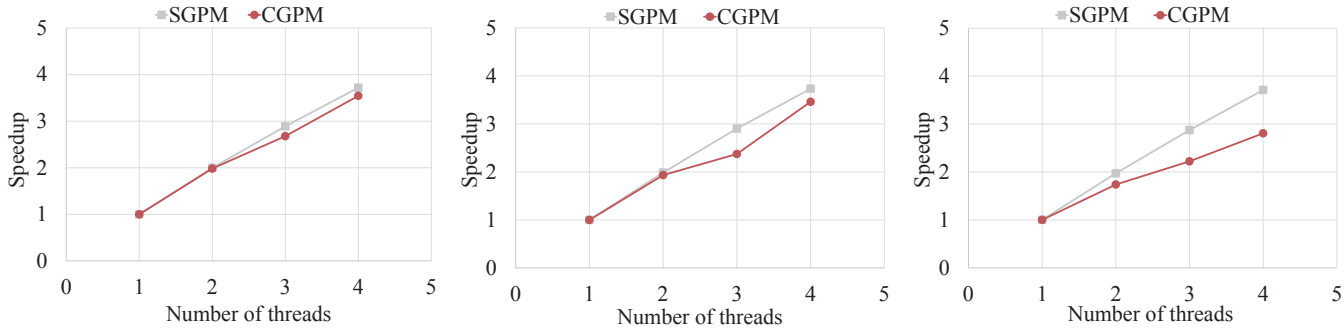
Table 1 shows the processing time for each process using *A_R15*, *A_Aggregation*, and *A_Pathbased*. The processing time of CGPM is faster than that of SGPM using *A_R15*, *A_Aggregation* and *A_Pathbased*, as shown in Table 1. These results indicate that CGPM is effective in terms of the processing time, using a data distribution with a large deviation on the coordinate space.

6. Conclusion

This paper proposed a new parallelization model on a multi-core CPU for the parallel processing of DBSCAN. The proposed parallelization model utilizes the data replication technique and complex grids in order to improve the speedup performance of parallel processing. The data replication technique is utilized to determine whether a geospatial data near the border of a grid is core geospatial data or not. Moreover, the proposed model reduces the number of replications owing to the complex partition grid partition. The experimental results showed that the proposed parallelization model outperforms the conventional parallelization model, which utilizes the simple grid partition. In our future work, we intend to discuss combining condition of the dense grids. Moreover, we intend to conduct experiments by increasing the number of workers.

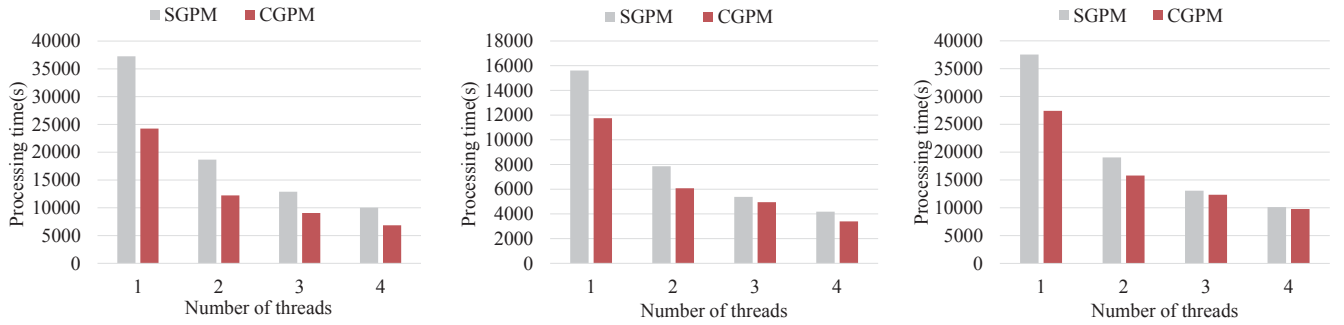
Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 16J05403 and 26330139, and Hiroshima City University Grant for Special Academic Research (General Studies).



Speedup of *R15* Speedup of *Aggregation* Speedup of *Pathbased*

Fig. 8: Speedup for each dataset with chenging the condition of combining of the dense grids



Processing time of *R15* Processing time of *Aggregation* Processing time of *Pathbased*

Fig. 9: Processing time for each dataset with chenging the condition of combining of the dense grids

Table 1: Processing time for each process using data distribution with a large deviation

Datasets	Model	Calculating neighbors(s)	Creating tasks(s)	Tasks process(s)	Merging clusters(s)	Total time(s)
A_R15	SGPM	26.05	0.08	21296.92	0.02	21323.07
	CGPM	30.15	16.42	12767.14	0.06	12813.78
A_Aggregation	SGPM	38.48	0.09	27733.29	0.11	27771.98
	CGPM	49.12	31.56	22728.25	0.14	22809.17
A_Pathbased	SGPM	71.69	0.12	31777.16	0.14	31849.11
	CGPM	97.29	88.39	30876.74	0.20	31062.62

References

[1] M. Ester, H. peter Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD-96, 1996, pp. 226–231.

[2] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[3] X. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Mining and Knowledge Discovery*, vol. 3, no. 3, pp. 263–290, 1999.

[4] D. Arlia and M. Coppola, "Experiments in parallel clustering with dbscan," in *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, ser. Euro-Par '01, 2001, pp. 326–331.

[5] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith, "The sequoia 2000 storage benchmark," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93, 1993, pp. 2–11.

[6] C. Böhm, R. Noll, C. Plant, and B. Wackersreuther, "Density-based clustering using graphics processors," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ser. CIKM '09, 2009, pp. 661–670.

[7] B. Welton, E. Samanas, and B. P. Miller, "Mr. scan: Extreme scale density-based clustering using a tree-based network of gpgpu nodes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13, 2013, pp. 84:1–84:11.

[8] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, "Mr-dbscan: An efficient parallel density-based clustering algorithm using mapreduce," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, 2011, pp. 473–480.

[9] K. Misaki, K. Tamura, and H. Kitakami, "Parallel processing for density-based clustering algorithm on a multi-core cpu," in *Proceedings 2014 IEEE SMC Hiroshima Chapter Young Researchers' Workshop*, 2014, pp. 33–36.

Proposal on a Linear Regression being hardly affected by outliers and its application to the Estimation of Michaelis Constant

T. Matsuda[1], **Y. Kawaguchi**[1], **and K. Ohsugi**[1]
[1]Department of Computer Science, Shizuoka Institute of Science and Technology, Fukuroi, Shizuoka, Japan

Abstract— *The Michaelis constant is utilized for the investigation on the relation between an enzyme and a substrate concentration, and is estimated by Michaelis-Menten equation. The conventional estimation method is the least squares method, but it is problem that the data of the Michaelis includes some outlier. In this study, we proposed a estimation method of a linear regression which is robust for outliers, and applied to the estimation of Michaelis constant in a simple case. Moreover, we compared our proposed method with the conventional method, and showed the effectiveness of our proposed method.*

Keywords: Michaelis constant, Linear regression, least squares fitting, Outliers

1. Introduction

Michaelis-Menten equation is very important theory in the enzyme kinetics and had supported development of the former enzyme chemistry [1]. Michaelis-Menten equation is defined in the following way:

$$\frac{1}{v} = \frac{K_m}{V_{max}}\frac{1}{[S]} + \frac{1}{V_{max}}. \tag{1}$$

Here, v is a speed of reaction of the enzyme E, $[S]$ is the concentration of the substrate S, V_{max} indicates the maximum rate achieved at maximum substrate concentrations and K_m is the Michaelis constant. Michaelis-Menten equation expresses that the speed of reaction increases when the substrate concentration $[S]$ increases. The Michaelis constant K_m is the substrate concentration at $v = \frac{V_{max}}{2}$. Michaelis-Menten equation is well-known model of enzyme kinetics in biochemistry, but it may not give the complete data fitting. In particular, we have to pay attention to the case including some outliers over the data set $\left\{\frac{1}{[S_i]}, \frac{1}{v_i}\right\}_{i=1}^{I}$. In the theory of the enzyme kinetics, it is very important to use the concentration of the substrate in the condition of a zero-order reaction, because a chemical reaction proceeds regardless in the density of the substance. However, the Michaelis constant K_m is estimated by using the data except a zero-order reaction, therefore the error of the measurement of K_m are caused by some outliers. So, it is to be desired that outliers are removed when we estimate K_m. Many methods on removing outliers had been studied before now [2] [**?**], it is not easy to judge whether the given data is an outlier or not correctly.

In this study, we proposed an algorithm of a linear regression that is robust for outliers, and estimated the Michaelis constant K_m by using our proposed algorithm. Our proposed algorithm is based on some linear classifier, called Exact Soft Confidence-Weighted Learning (SCW) [3]. SCW is an algorithm on an online supervised learning, but the estimation of K_m is not a supervised learning. Therefore, we included some device in our proposed algorithm to to apply algorithm of supervised learning. The data of this study is the one that used for investigating the enzymatic activity of Alkaline phosphatase (ALP). In the data of this study, outliers appear in the range of $0 < [S] < 1$. To investigate the effectiveness of our proposed algorithm, we generated some outliers artificially, and compared the estimation result of our proposed algorithm and the least squares method as a conventional method. As a result, we confirmed that our proposed algorithm is less subject to outliers. The rest of this paper is organized as follows. Section 2 explains on the Michaelis-Menten Equation simply. Section 3 proposes an algorithm of a linear regression. Section 4 shows the experiment of our proposed algorithm, and Section 5 concludes this study.

2. Michaelis-Menten Equation

Enzyme E plays an important role in a chemical reaction of animal and plant bodies. A material which is catalyzed by an enzyme is called a substrate S. An enzyme and a substrate reacts and generates a complex ES.

$$E + S \rightarrow ES \tag{2}$$

We assume that the rate constant of Eq. (2) is k_1. A complex ES is decomposed into E and S (Eq. (3)), or becomes a reaction product P (Eq. (4)).

$$ES \rightarrow E + S \tag{3}$$

$$ES \rightarrow E + P \tag{4}$$

We assume that the rate constant of Eq. (3) and Eq. (4) is k_2 and k_3, respectively.

The speed of reaction of the enzyme is determined by $[E]$ and $[S]$. Michaelis-Menten equation is derived using steady-state approximation [**?**] in the following way.

$$v = \frac{V_{max}[S]}{K_m + [S]}, \tag{5}$$

where

$$K_m = \frac{k_2 + k_3}{k_1}.$$

If $[S] > K_m$, then $v = V_{max}$ and v does not depend on $[S]$ (zero-order reaction). If $[S] < K_m$, then

$$v = \frac{V_{max}}{K_m}[S].$$

From Eq. (5), we have

$$\begin{aligned} \frac{1}{v} &= \frac{K_m + [S]}{V_{max}[S]} \\ &= \frac{K_m}{V_{max}}\frac{1}{[S]} + \frac{1}{V_{max}}. \end{aligned}$$

This equation is called Lineweaver-Burk plot. The set of data $\left\{\frac{1}{[S_i]}, \frac{1}{v_i}\right\}_{i=1}^{I}$ does not lie over a straight line. Therefore, the Michaelis constant K_m is estimated using the linear least squares fitting. Let

$$\begin{aligned} y &= \frac{1}{v}, \\ x &= \frac{1}{[S]}, \\ a &= \frac{K_m}{V_{max}}, \\ b &= \frac{1}{V_{max}} \end{aligned}$$

and $\{(x_i, y_i)\}_{i=1}^{I} = \left\{\frac{1}{[S_i]}, \frac{1}{v_i}\right\}_{i=1}^{I}$. Our goal is to compute the linear regression $y = ax + b$ from the data $\{(x_i, y_i)\}_{i=1}^{I}$. The parameters a and b are computed as follows.

$$\begin{aligned} a &= \frac{\left(\sum_{i=1}^{I} y_i\right)\left(\sum_{i=1}^{I} x_i^2\right) - \left(\sum_{i=1}^{I} x_i\right)\left(\sum_{i=1}^{I} x_i y_i\right)}{I\left(\sum_{i=1}^{I} x_i^2\right) - \left(\sum_{i=1}^{I} x_i\right)^2} \\ b &= \frac{I\left(\sum_{i=1}^{I} x_i y_i\right) - \left(\sum_{i=1}^{I} x_i\right)\left(\sum_{i=1}^{I} y_i\right)}{I\left(\sum_{i=1}^{I} x_i^2\right) - \left(\sum_{i=1}^{I} x_i\right)^2} \end{aligned}$$

The estimation results of a and b may be influenced by outliers including in data set. In this study, we proposed another estimation method of the computation on the Michaelis constant K_m. We will introduce our proposed algorithm in the next section.

3. Algorithm

In this section, we propose an algorithm of a linear regression estimator based on the online supervised learning which is called SCW [3]. Therefore, firstly, we will overview the algorithm of SCW.

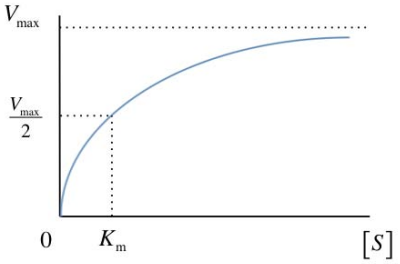


Fig. 1: Michaelis constant K_m

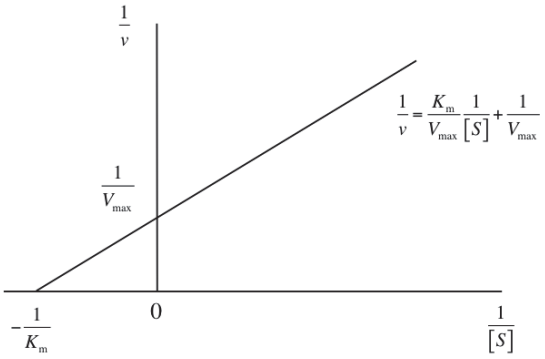


Fig. 2: Lineweaver-Burk plot

3.1 SCW

SCW is an online learning algorithm considered a soft margin learning, and is constructed by extending the confidence-weighted learning (CW) [4]. The algorithms of CW and SCW estimate the parameter $\mathbf{w} \in \mathbf{R^N}$ of the linear classifier

$$< \mathbf{w}, \mathbf{s} >= \sum_{\mathbf{n=1}}^{\mathbf{N}} \mathbf{w_i s_i},$$

where $\mathbf{s} \in \mathbf{R^N}$ is an input data. In the supervised learning, we consider some label data t_i associated to the input data $\mathbf{s_i}$. We assume $t_i \in \{1, -1\}$ in this study. The algorithm of CW utilizes a normal distribution $\mathbf{N}(\mu, \mathbf{\Sigma})$ with mean vector $\mu \in \mathbf{R^d}$ and covariance matrix Σ to determine the parameter $\mathbf{w}$. It is assumed the parameter $\mathbf{w}$ has a normal distribution $\mathbf{N}(\mu, \mathbf{\Sigma})$, that is

$$\mathbf{w} \sim \mathbf{N}(\mu, \mathbf{\Sigma}).$$

The algprithm of CW estimates the parameter $\mathbf{w}$ by using the information of the frequency of the data. For the high frequency data, the update of $\mathbf{w}$ becomes slower. On the other hand, the update of $\mathbf{w}$ becomes faster for the low frequency data. The optimization problem of CW is written as follows.

$$(\mu_\mathbf{n}, \mathbf{\Sigma_n}) = \arg\min_{\mu, \mathbf{\Sigma}} \mathrm{KL}\left(\mathbf{N}(\mu, \mathbf{\Sigma}), \mathbf{N}(\mu_\mathbf{n}, \mathbf{\Sigma_n})\right)$$

subject to $\Pr[t_n < \mathbf{w}, \mathbf{s_n} >\geq \mathbf{0}] \geq \eta$. Here, $0 \leq \eta \leq 1$ and $\mathrm{KL}(\cdot,\cdot)$ denotes the Kullback Leibler information.

$$\mathrm{KL}(A(x), B(x)) = \int A(x) \log \frac{A(x)}{B(x)} dx.$$

The restrection condition $\Pr[t_n < \mathbf{w}, \mathbf{s_n} >\geq \mathbf{0}] \geq \eta$ can be rewirtten as

$$t_n < \mu, \mathbf{s_n} >\geq \mathbf{\Phi^{-1}}(\eta)\sqrt{< \mathbf{s_n}, \mathbf{\Sigma s_n} >},$$

where Φ is the cumulative function of the normal distribution. The algorithm of SCW is obtained from the following optimization problem.

$$(\mu_\mathbf{n}, \mathbf{\Sigma_n}) = \arg\ \min_{\mu,\mathbf{\Sigma}}\ \mathrm{KL}\left(\mathbf{N}(\mu, \mathbf{\Sigma}), \mathbf{N}(\mu_\mathbf{n}, \mathbf{\Sigma_n})\right)$$

subject to

$$\max\left(0, \Phi^{-1}(\eta)\sqrt{< \mathbf{s_n}, \mathbf{\Sigma s_n} >} - t_n < \mu, \mathbf{s_n} >\right) = 0.$$

SCW allows some error of probability $(1-\eta)$ and stops a rapidly change of the parameter $\mathbf{w}$. The update rules of the parameter μ and Σ of SCW are obtained from

$$\begin{aligned} \mu_{n+1} &= \mu_n + \alpha_n t_n \Sigma_n \mathbf{s_n} \\ \Sigma_{n+1} &= \Sigma_n - \beta_n \Sigma_n \mathbf{s_n^{\mathrm{T}} s_n \Sigma_n}, \end{aligned}$$

where

$$\begin{aligned} \alpha_n &= \min\left\{C, \max\left\{0, \frac{1}{p_n\zeta}\left(-m_n\phi + A\right)\right\}\right\} \\ A &= \sqrt{m_n^2\frac{\Phi^{-1}(\eta)}{4} + p_n(\Phi^{-1}(\eta))^2\zeta} \\ \beta_n &= \frac{\alpha_n\Phi^{-1}(\eta)}{\sqrt{u_n} + p_n\alpha_n\Phi^{-1}(\eta)} \\ u_n &= \frac{-\alpha_n p_n\Phi^{-1}(\eta) + \sqrt{\alpha_n^2 p_n^2(\Phi^{-1}(\eta))^2 + 4p_n}}{4} \\ p_n &= < \mathbf{s_n}, \mathbf{\Sigma s_n} > \\ m_n &= t_n < \mu_n, \mathbf{s_n} > \\ \phi &= 1 + \frac{(\Phi^{-1}(\eta))^2}{2} \end{aligned}$$

The data for estimating the Michaelis constant K_m has not label data $t_n \in \{-1, 1\}$. Therefore, we need some idea to apply the algorithm of SCW to our proposed algorithm.

3.2 Proposed Algorithm

Here, we propose an algorithm of linear regression estimator in the following way.
(Step 1)
Let $\{(x_n, y_n)\}_{n=1}^N$ be a sample.
(Step 2)
Compute the average coordinate of $\{(x_n, y_n)\}_{n=1}^N$

$$\begin{aligned} x_r &= \frac{1}{N}\sum_{n=1}^N x_n, \\ y_r &= \frac{1}{N}\sum_{n=1}^N y_n, \end{aligned}$$

and transform (x_n, y_n) into

$$\begin{aligned} X_n &= x_n - x_r, \\ Y_n &= y_n - y_r \end{aligned}$$

(Step 3)
Compute the eigenvector $\mathbf{p} = (p_1, p_2)$ corresponding to the first principal component of the variance covariance matrix

$$\frac{1}{N}\begin{pmatrix} \sum_{n=1}^N X_n^2 & \sum_{n=1}^N X_n Y_n \\ \sum_{n=1}^N Y_n X_n & \sum_{n=1}^N Y_n^2 \end{pmatrix} \tag{6}$$

and the line

$$y = \frac{p_2}{p_1}x$$

(Step 4)
If $Y_n > \frac{p_2}{p_1}X_n$ (resp. $Y_n \leq \frac{p_2}{p_1}X_n$), then we give the label $(X_n, Y_n, z_n = +1)$ (resp. $(X_n, y_n, z_n = -1)$).
(Step 5)
Compute the parameter w_1 and w_2 of

$$w_1 X + w_2 Y = 0$$

by using the algorithm of SCW.
(Step 6)
By substituting the transforms of (Step 2), we have

$$\begin{aligned} y &= -\frac{w_1}{w_2}x + \frac{w_1 x_r + w_2 y_r}{w_2} \\ &= ax + b. \end{aligned}$$

In the process of (Step 4), we give the label data $\{+1, -1\}$ by using the eigenvector (the first principal component) of the variance - covariance matrix. The line through the average coordinate (x_r, y_r) with the slope $-\frac{w_1}{w_2}$ may fit to the data $\{(X_n, Y_n)\}_{n=1}^N$ if the data does not have outliers. In this study, we will give some outliers artificially and investigate the behavior of our proposed algorithm and conventional method.

4. Experiment

In this section, we will do an experiment using real data.

4.1 Experiment Data

Firstly, we explain the real data of this study. Our data of this study is composed of

E : Alkaline phosphatase (ALP)
S : p-Nitrophenyl phosphate (pNPP)
P : p-Nitrophenol (pNP)

This chemical experiment investigate the reaction mass of pNP and the enzymatic activity of ALP using the substrate pNPP under the condition without the factor of hindrance. ALP is an enzyme distributed over a liver, a bone and a small intestine and using for the study of the bone metabolism. Table 1 is the specific data of $[S]$ and $\frac{1}{v}$.

Table 1: Data of S and v

$[S]$ (mM)	v (μ mol/min)	$\frac{1}{[S]}$	$\frac{1}{v}$
0.05	0.004478	20	223.3
0.1	0.007518	10	133.02
0.2	0.011751	5	85.1
0.5	0.017284	2	57.86
1	0.020835	1	47.99
2	0.02391	0.5	41,82
5	0.0259	0.2	38.61
10	0.02605	0.1	38.39

We used 8 data set $\{(x_{n_j}, y_{n_j})\}_{n_j=1}^{N_j}, (j = 1, 2, \cdots, 8)$ to estimate the constants of the line

$$\frac{1}{v} = \frac{K_m}{V_{max}} \frac{1}{[S]} + \frac{1}{V_{max}}.$$

4.2 Experiment Results

Here, we will show the result of our proposed method and the conventional method (linear least squares fitting). The red line and green line of the Fig 3 and Fig 4 indicate the result of our proposed method and the conventional method, respectively. Fig 3 and 4 are obtained from the data set No. 1 and No. 2, respectively.

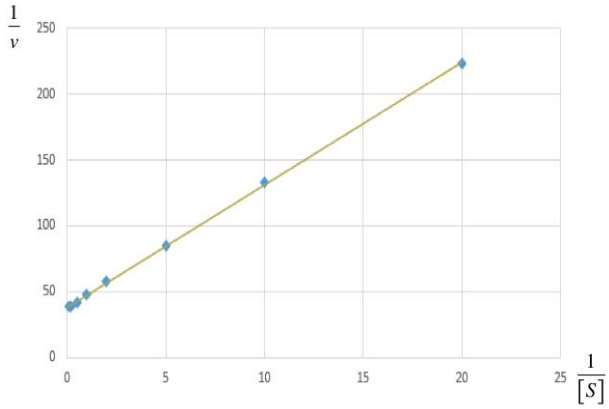


Fig. 3: Experiment result 1

It can be seen that almost of the data of Fig 3 are lying on some line. Therefore, it may be considered that the result of our proposed method and the conventional method coincide almost. On the other hand, the data of Fig 4 are scattered from some line. Therefore, the result of our proposed method is different from the one of the conventional method. To compare our proposed method and the conventional method, we will compute the distance from the data to the fitting line. The residual sum of squares are minimized in the linear least squares fitting. In this study, we compute the sum of the length of a perpendicular lowered to the fitting line from the

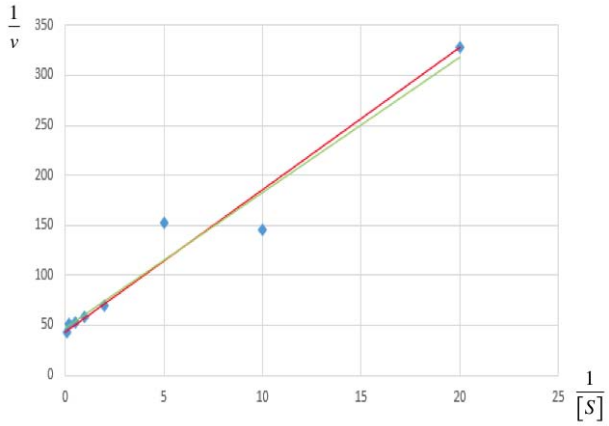


Fig. 4: Experiment result 2

data coordinate. We summarized the computation result of the sum of the length of a perpendicular in Table 2.

Table 2: The distance between a data and a line

Number of data set	Proposed Method	Conventional Method
1	0.846	0.848
2	6.258	7.182
3	3.383	3.609
4	1.902	1.962
5	2.402	2.41
6	0.690	0.691
7	1.019	1.031
8	0.689	0.691

We can see that results of our proposed method and the conventional method are almost the same with the exception of the data set No. 2 and No. 3. In all cases, the distance between a data and a line of our proposed method is smaller than the one of the conventional method. The purpose of this study is to propose the estimation method that is less subject to outliers. Therefore, we investigate the behavior of our proposed method and the conventional method by giving an outlier.

4.3 Experiment for Outliers

To investigate the Influence by outliers, We changed the part of the data set No. 1 (the data of Table 1) to an outlier. Here, we call the data set No. 1 the original data. The red line and the green line of Fig 5, 6, 7 and 8 indicate the estimation result of our proposed method and the conventional method, respectively. Moreover, the dotted line is the estimation result of the conventional method using the original data. Fig 5 was obtained by changing the data $(\frac{1}{[S]}, \frac{1}{v}) = (10, 133.02)$ of Table 1 into $(10, 100)$. From Fig 5, we can see that the influence by an outlier concerning the parameter estimation of our proposed method is smaller than the one of the conventional method. We summarized the changed point in

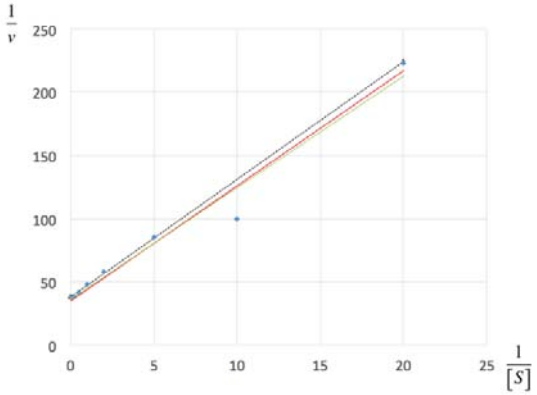


Fig. 5: Experiment result 1 on outliers

Table 3.

Table 3: Outliers data

Number of figure	Outlier
Fig 5	$(10, 133.02) \rightarrow (10, 100)$
Fig 6	$(10, 133.02) \rightarrow (10, 70)$
Fig 7	$(10, 133.02) \rightarrow (10, 180)$
Fig 8	$(10, 133.02) \rightarrow (10, 210)$

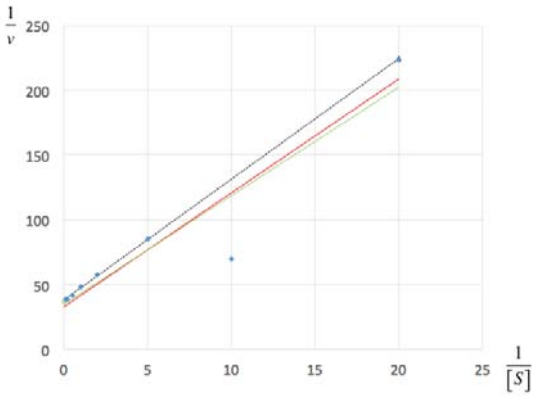


Fig. 6: Experiment result 2 on outliers

In any cases of Fig 5, 6, 7 and 8, we can see that our proposed method is robust by comparing with the conventional method because the estimated line of our proposed method is closely to the dotted line. Therefore, it may be said that our proposed method achieved our purpose. Namely, our proposed method may be robust slightly for outliers in this experiment of the estimation on the Michaelis constant by comparing to the conventional method.

5. Conclusions

In this study, we proposed an algorithm of a linear regression that is robust for outliers. The result of Section 4 shows the effectiveness of our proposed algorithm. The data of this study has the condition without the factor of

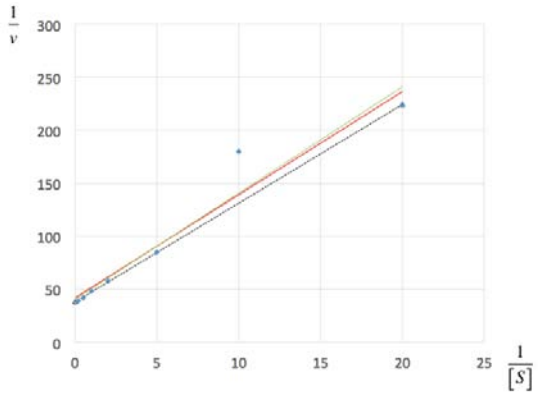


Fig. 7: Experiment result 3 on outliers

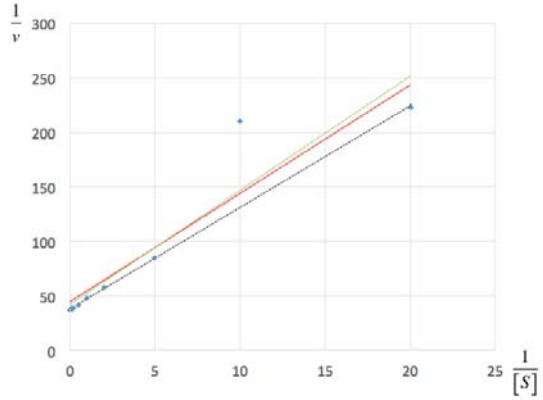


Fig. 8: Experiment result 4 on outliers

an inhibitor in enzyme reactions. Our proposed algorithm can apply to a non-linear problem. To apply our proposed algorithm to complex problem and provide free software of our proposed algorithm are our future problems.

Acknowledgement

The experiment data of this study was provided from the class "Experiments in Materials and Life Science" of Department of Materials and Life Science at Shizuoka Institute of Science and Technology. The authors wish to acknowledge Dr. Akihiro Saito for providing the data.

References

[1] Michaelis, L. and Menten, M. L. *Biochem*, Z, 49, p.333, 1913.
[2] V Hodge, J Austin. *A survey of outlier detection methodologies*, Artificial Intelligence Review 22 (2), pp.85-126, 2004.
[3] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, JÃűrg Sander. *LOF: identifying density-based local outliers*, Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pp.93-104, 2000.
[4] Jialei Wang et al. *Exact Soft Confidence-Weighted Learning*, International Conference on Machine Learning, pp.121-128, 2012.
[5] A. D. McNaught (Author), A. Wilkinson *Compendium of Chemical Terminology*, Wiley; 2 edition, 1997.
[6] Dredze, Mark, Crammer, Koby, and Pereira, Fernando. *Confidence-weighted linear classification*, International Confer- ence on Machine Learning, pp. 264âĂŞ271, 2008.

Implementation of Computing Singular Pairs for Large Scale Matrices using ARPACK

Masami Takata[1], **Sho Araki**[2], **Kinji Kimura**[2], **Yuki Fujii**[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— *In this paper, we propose a new implementation for computing singular pairs for large-scale matrices, which is introduced from the viewpoint of the computational order and the caches of shared-memory multi-core processors. In the case when only the partial eigenvalues and eigenvectors from the absolute maximum or the absolute minimum eigenvalue of a target matrix are needed, we use an effective software package, called ARPACK(ARnoldi PACKage), and transform singular value decompositions into eigenvalue problems. To get the singular value decompositions using ARPACK, the transformed eigenvalue problems can be generally computed through the use of two matrix-vector operations at each iteration. If the size of the target matrix is large, the large number of the elements in the target matrix cause the caches of the shared-memory multi-core processors to overflow. On the other hand, the proposed implementation can achieve a high cache hit ratio because each row in the target matrix can be reused. The proposed implementation was evaluated by experimentation. The experimental results show that the computation time of the proposed implementation is about* 80% *of that of the conventional implementation.*

Keywords: Krylov subspace method, IRA algorithm, IRL algorithm, sparse matrix, dense matrix, matrix-vector multiplication

1. Introduction

In statistical analysis, feature quantity of target matrices is obtained by using principal component analysis. In principal component analysis, SVD (singular value decomposition) of a large matrix, which can be dense or sparse, is needed. In SVD, we compute singular pairs, which consists of a singular value and the corresponding singular vectors. However, the singular pairs, which have larger or smaller singular values, are more useful. Thus, a partial SVD, in which we compute only these singular pairs, is more suitable. An SVD can be transformed into an eigenvalue problem through multiplication of a target matrix by the transpose matrix. In this case, all eigenvalues are non-negative numbers.

In the case where only the partial eigenpairs, which are combined with a eigenvalue and the corresponding eigenvector, from the absolute maximum or the absolute minimum eigenvalue of the target matrix are needed, these eigenpairs are computed using the IRA(implicitly restarted Arnoldi) algorithm [7] and the IRL(implicitly restarted Lanczos) algorithm [8] using the ARPACK (ARnoldi PACKage) [5] software, which is a solver for large-scale matrices. The IRA and IRL algorithms, which are two of the Krylov subspace methods, are effective for solving partial eigenvalue problems. The idea of the IRA and IRL algorithms is to reduce the computational cost by limiting the number of bases in Krylov subspace.

Since ARPACK adopts reverse communication interface [5], users simply compute matrix-vector operations. In general, a partial eigenvalue problem can be computed by using matrix-vector multiplication, of which the number should be set at about 10 times the number of required eigenvalues in ARPACK. In the case where ARPACK is used for SVD, two matrix-vector operations are generally needed at each iteration. In an example file (dsvd.f) [5] for partial SVD in ARPACK, the computational order and the caches of shared-memory multi-core processors are not considered. We therefore propose a new implementation using ARPACK. The new implementation is more effective in terms of parallel computation on shared-memory multi-core processors with large caches. By using OpenMP directives, the implementation can result in much higher performance in parallel computing.

In Section 2, we introduce the Krylov subspace method. In Section 3, we introduce the IRA and the IRL algorithms. In Section 4, we propose an implementation of SVD using ARPACK software. In Section 5, we evaluate the performances of the proposed implementation on the multi-core processor with large caches and discuss the results.

2. Krylov subspace method

2.1 Arnoldi algorithm

The Arnoldi algorithm [1] transforms the target matrix A to the approximate matrix $H_k \in \mathbb{R}^{k\times k}$ of A, whose size is rather smaller than A, by using the Krylov subspace method.

The Krylov subspace is a linear subspace based on the power method and is composed of $A, \mathbf{q}_1$, and k. $\mathbf{q}_1$ is an initial vector and $k\ (k < n)$ is an iteration number:

$$\mathcal{K}(A, \mathbf{q}_1, k) = \mathrm{span}\{\mathbf{q}_1, A\mathbf{q}_1, \ldots, A^{k-1}\mathbf{q}_1\}. \qquad (1)$$

Algorithm 1 Arnoldi algorithm

1: Set initial vector $\mathbf{q}_1$;
2: $\mathbf{v}_1 := \mathbf{q}_1/\|\mathbf{q}_1\|$;
3: **for** $j := 1$ **to** k **do**
4: $\quad \mathbf{r}_j := A\mathbf{v}_j$;
5: $\quad$ **for** $i := 1$ **to** j **do**
6: $\quad\quad h_{ij} := \mathbf{v}_i^\top \mathbf{r}_j$;
7: $\quad\quad \mathbf{r}_j := \mathbf{r}_j - h_{ij}\mathbf{v}_i$;
8: $\quad$ **end for**
9: $\quad h_{j+1,j} := \|\mathbf{r}_j\|$; $\mathbf{v}_{j+1} := \mathbf{r}_j/h_{j+1,j}$;
10: **end for**

The iteration number k depends on the algorithms. In the Arnoldi algorithm explained in this section and the Lanczos algorithm introduced in Section 2.2, the iteration number k is determined when the eigenvalues of matrix H_k is well approximated to that of A. On the other hand, in the implicitly restarted Arnoldi algorithm in Section 3, k is determined by users as the limit of the number of bases in the Krylov subspace.

Let $A \in \mathbb{R}^{n\times n}$ be non-symmetric. We set an initial vector $\mathbf{q}_1 \in \mathbb{R}^n$ ($\mathbf{q}_1 \neq \mathbf{0}$), and generate new base vectors $\mathbf{q}_k \in \mathbb{R}^n$ from the vectors which we have already computed. The $\mathbf{q}_k$ is a base vector of the Krylov subspace $\mathcal{K}(A, \mathbf{q}_1, k)$. Actually, in the Arnoldi algorithm, for each iteration, a new base $\mathbf{q}_k$ is obtained using $\mathbf{q}_k = A^{k-1}\mathbf{q}_1$ and orthogonalization. The new base vector is orthogonalized against existing base vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{k-1}$, and the new base vector $\mathbf{q}_k$ is normalized to $\mathbf{v}_k \in \mathbb{R}^n$. Then, an orthonormal basis results:

$$\mathcal{K}(A, \mathbf{q}_1, k) = \mathcal{K}(A, \mathbf{v}_1, k) = \mathrm{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}, \quad (2)$$

where $\mathbf{v}_j$ ($j = 1, \ldots, k$) are orthonormal vectors obtained by the Arnoldi algorithm.

Algorithm 1 shows the pseudocode of the Arnoldi algorithm. Lines 5 to 8 of Algorithm 1 denote the orthogonalization part. The orthogonalization part of Algorithm 1 is written by the modified Gram-Schmidt algorithm [4].

After the k-th iteration for $k = 2, 3, \ldots$ in Algorithm 1, the following equation holds:

$$AV_k = V_k H_k + \mathbf{r}_k \mathbf{e}_k^\top, \quad (3)$$

where $V_k := [\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_k] \in \mathbb{R}^{n\times k}$, $\mathbf{e}_k \in \mathbb{R}^k$ is the k-th column vector of the $k \times k$ identity matrix, and

$$H_k := \begin{bmatrix} h_{11} & h_{12} & \cdots & \cdots & h_{1k} \\ h_{21} & h_{22} & \cdots & \cdots & h_{2k} \\ 0 & h_{32} & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{k,k-1} & h_{kk} \end{bmatrix}. \quad (4)$$

Moreover, $\mathbf{r}_k \in \mathbb{R}^n$ is a residual vector $\mathbf{r}_k := h_{k+1,k}\mathbf{v}_{k+1}$. $H_k \in \mathbb{R}^{k\times k}$ is an upper Hessenberg matrix, and it is an approximate matrix of A. The components of H_k are represented by using the equation $h_{ij} = \mathbf{v}_i^\top A\mathbf{v}_j$, which is expressed in lines 5 to 8 of Algorithm 1.

We introduce the stopping criterion of the Arnoldi algorithm as follows. Let $\lambda_j^{(k)} \in \mathbb{R}$ and $\mathbf{y}_j^{(k)} \in \mathbb{R}^n$ be the eigenvalues of H_k and the unit eigenvectors corresponding to $\lambda_j^{(k)}$, respectively. Then, the following equation is satisfied:

$$H_k \mathbf{y}_j^{(k)} = \lambda_j^{(k)} \mathbf{y}_j^{(k)}. \quad (5)$$

When we set $\mathbf{x}_j^{(k)} := V_k \mathbf{y}_j^{(k)} \in \mathbb{R}^n$, the following equation is formulated from Eq. (3):

$$\begin{aligned} A\mathbf{x}_j^{(k)} - \lambda_j^{(k)}\mathbf{x}_j^{(k)} &= AV_k\mathbf{y}_j^{(k)} - \lambda_j^{(k)} V_k \mathbf{y}_j^{(k)} && (6)\\ &= AV_k\mathbf{y}_j^{(k)} - V_k\lambda_j^{(k)}\mathbf{y}_j^{(k)} && (7)\\ &= AV_k\mathbf{y}_j^{(k)} - V_k H_k \mathbf{y}_j^{(k)} && (8)\\ &= (AV_k - V_k H_k)\mathbf{y}_j^{(k)} && (9)\\ &= \mathbf{r}_k \mathbf{e}_k^\top \mathbf{y}_j^{(k)} && (10)\\ &= (\mathbf{e}_k^\top \mathbf{y}_j^{(k)})\mathbf{r}_k. && (11) \end{aligned}$$

If we set $E := -(\mathbf{e}_k^\top \mathbf{y}_j^{(k)})\mathbf{r}_k {\mathbf{x}_j^{(k)}}^\top \in \mathbb{R}^{n\times n}$, Eq. (11) is transformed into

$$(A + E)\mathbf{x}_j^{(k)} = \lambda_j^{(k)}\mathbf{x}_j^{(k)}. \quad (12)$$

Equation (12) is regarded as an eigenvalue problem which has the added perturbation E to A. Thus, if the norm $\|E\|_2$ is small, $\lambda_j^{(k)}$ approximates an eigenvalue of A.

Algorithm 2 Lanczos algorithm

1: Set initial vector $\mathbf{q}_1$;
2: $\mathbf{v}_1 := \mathbf{q}_1/\|\mathbf{q}_1\|$; $\beta_0 := 0$; $\mathbf{q}_0 := \mathbf{0}$;
3: **for** $j := 1$ **to** k **do**
4: $\quad \mathbf{r}_j := A\mathbf{v}_j - \beta_{j-1}\mathbf{v}_{j-1}$;
5: $\quad \alpha_j := \mathbf{v}_j^\top \mathbf{r}_j$;
6: $\quad \mathbf{r}_j := \mathbf{r}_j - \alpha_j\mathbf{v}_j - \beta_{j-1}\mathbf{v}_{j-1}$;
7: $\quad \beta_j := \|\mathbf{r}_j\|$; $\mathbf{v}_{j+1} := \mathbf{r}_j/\beta_j$;
8: **end for**

2.2 Lanczos algorithm

As well as the Arnoldi algorithm, the Lanczos algorithm [6] generates orthonormal bases $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ according to the increasing iteration number k.

Algorithm 2 shows the pseudocode of the Lanczos algorithm. In contrast to the A in the Arnoldi algorithm, here $A \in \mathbb{R}^{n\times n}$ is assumed to be symmetric. Hence,

$$\begin{aligned} h_{ij} &= \mathbf{v}_i^\top A\mathbf{v}_j = \mathbf{v}_i^\top A^\top \mathbf{v}_j \\ &= (A\mathbf{v}_i)^\top \mathbf{v}_j = \left(\mathbf{v}_j^\top (A\mathbf{v}_i)\right)^\top = h_{ji} \end{aligned} \quad (13)$$

is satisfied. Then, for the approximate matrix $T_k \in \mathbb{R}^{k\times k}$ at the end of the k-th iteration in the Lanczos algorithm, the following equation holds:

$$AV_k = V_k T_k + \mathbf{r}_k \mathbf{e}_k^\top, \quad (14)$$

where $V_k := [\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_k] \in \mathbb{R}^{n\times k}$, $\mathbf{e}_k \in \mathbb{R}^k$ is the k-th column vector of the $k \times k$ identity matrix, and

$$T_k := \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ddots & \vdots \\ 0 & \beta_2 & \alpha_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{k-1} \\ 0 & \cdots & 0 & \beta_{k-1} & \alpha_k \end{bmatrix}. \quad (15)$$

T_k is a symmetric tridiagonal matrix whose eigenvalues approximate those of A.

However, in the Lanczos algorithm, the orthogonality of vectors becomes worse as the iteration number increases because the algorithm is more susceptible to rounding errors than the Arnoldi algorithm. Thus, lines 5 to 8 of Algorithm 1 are usually used even if A is symmetric. Moreover, the stopping criterion of the Lanczos algorithm is the same as that of the Arnoldi algorithm.

3. Implicitly restarted Arnoldi algorithm and Implicitly restarted Lanczos algorithm

In this section, following [7], [8], we introduce the IRA and IRL algorithms. The number of desired eigenpairs is set to be ℓ. In the Arnoldi and Lanczos algorithms, for each iteration, a new base vector is added with the expansion of the Krylov subspace until we obtain an approximate matrix. The cost of the re-orthogonalization keeps on increasing, so these algorithms need a lot of memory and computational time. The IRA and IRL algorithms reduce these re-orthogonalization costs by limiting the number of bases in Krylov subspace to m ($\ell < m \ll n$). The IRA and IRL algorithms are implemented in ARPACK [5].

3.1 Implicitly shifted QR steps

In the IRA and IRL algorithms, the implicit QR steps are used. The implicit QR steps are derived from the explicit QR steps. The QR steps are the algorithm to renew $\tilde{H}_m^{(i)} \in \mathbb{R}^{m\times m}$ based on the following recurrence formula:

$$\tilde{H}_m^{(i)} = \tilde{Q}_i\tilde{R}_i \quad (16)$$
$$\tilde{H}_m^{(i+1)} = \tilde{R}_i\tilde{Q}_i. \quad (17)$$

Starting from the initial matrix $H_m^{(1)} \in \mathbb{R}^{m\times m}$, $\tilde{H}_m^{(i)}$ is the matrix at the end of the ith iteration. The following equation is obtained:

$$\tilde{H}_m^{(i)} = \tilde{Q}_{i-1}^\top \cdots \tilde{Q}_2^\top \tilde{Q}_1^\top H_m^{(1)} \tilde{Q}_1\tilde{Q}_2\cdots\tilde{Q}_{i-1} \quad (18)$$
$$= \tilde{Q}^\top \tilde{H}_m^{(1)} \tilde{Q} \quad (\tilde{Q} := \tilde{Q}_1\tilde{Q}_2\cdots\tilde{Q}_{i-1}). \quad (19)$$

On the other hand, in the implicitly shifted QR steps, the shift values $\mu_i \in \mathbb{R}$ are introduced:

$$\tilde{\tilde{H}}_m^{(i)} - \mu_i I = \tilde{\tilde{Q}}_i\tilde{\tilde{R}}_i \quad (20)$$
$$\tilde{\tilde{H}}_m^{(i+1)} = \tilde{\tilde{R}}_i\tilde{\tilde{Q}}_i + \mu_i I. \quad (21)$$

Starting from the initial matrix $H_m^{(1)}$, at the end of the ith iteration we obtain $\tilde{\tilde{H}}_m^{(i)} \in \mathbb{R}^{m\times m}$. Then, the following equation is satisfied:

$$\tilde{\tilde{H}}_m^{(i)} = \tilde{\tilde{Q}}_{i-1}^\top \cdots \tilde{\tilde{Q}}_2^\top \tilde{\tilde{Q}}_1^\top H_m^{(1)} \tilde{\tilde{Q}}_1\tilde{\tilde{Q}}_2\cdots\tilde{\tilde{Q}}_{i-1} \quad (22)$$
$$= \tilde{\tilde{Q}}^\top H_m^{(1)} \tilde{\tilde{Q}} \quad (\tilde{\tilde{Q}} := \tilde{\tilde{Q}}_1\tilde{\tilde{Q}}_2\cdots\tilde{\tilde{Q}}_{i-1}) \quad (23)$$

The implicitly shifted QR steps are as follows. The starting matrix $\tilde{\tilde{H}}_m^{(1)}$ is the upper Hessenberg matrix and, if μ_i is the eigenvalue of $\tilde{\tilde{H}}_m^{(1)}$, $\tilde{\tilde{R}}_1 \in \mathbb{R}^{m\times m}$ is an upper triangle matrix with $\{\tilde{\tilde{R}}_1\}_{n,n} = 0$:

$$\tilde{\tilde{H}}_m^{(1)} = \begin{bmatrix} * & * & \cdots & \cdots & * \\ * & * & \cdots & \cdots & * \\ 0 & * & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * & * \end{bmatrix}, \quad (24)$$

$$\tilde{\tilde{R}}_1 = \begin{bmatrix} * & * & \cdots & \cdots & * \\ 0 & * & \cdots & \cdots & * \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}. \quad (25)$$

Thus, $\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1$ becomes an upper Hessenberg matrix with $\{\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1\}_{n,n-1} = 0$ and $\{\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1\}_{n,n} = 0$:

$$\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1 = \begin{bmatrix} * & * & \cdots & \cdots & \cdots & * \\ * & * & \cdots & \cdots & \cdots & * \\ 0 & * & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * & * \\ 0 & \cdots & \cdots & 0 & 0 & 0 \end{bmatrix}. \quad (26)$$

Then, we obtain

$$\tilde{\tilde{H}}_m^{(2)} = \begin{bmatrix} * & * & \cdots & \cdots & \cdots & * \\ * & * & \cdots & \cdots & \cdots & * \\ 0 & * & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * & * \\ 0 & \cdots & \cdots & 0 & 0 & \mu_1 \end{bmatrix}. \quad (27)$$

Repeating this process m times, the diagonal components of $\tilde{\tilde{H}}_m^{(m)}$ are composed of the eigenvalues of $H_m^{(1)}$.

3.2 Implicit restarting

We compute m steps of Arnoldi iteration and then restart the iteration with a new initial vector $\mathbf{v}^+ \in \mathbb{R}^n$ chosen by performing the implicitly shifted QR algorithm from the Arnoldi vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$. In this section, we introduce the method to compute an ideal vector $\mathbf{v}^+$.

After the m steps of Arnoldi iteration, we obtain the following relation:

$$AV_m = V_m H_m + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top. \tag{28}$$

Then, we compute all the eigenvalues $\lambda_1, \ldots, \lambda_m$ of H_m, and divide them into $\lambda_1, \ldots, \lambda_\ell$, which approximate the desired eigenvalues of A, and $\lambda_{\ell+1}, \lambda_{\ell+2}, \ldots, \lambda_m$. Next, we apply the $m-\ell$ implicitly shifted QR steps to H_m with $\lambda_{\ell+1}, \lambda_{\ell+2}, \ldots, \lambda_m$ as the shift values to obtain H_m^+. By using $\lambda_{\ell+1}, \lambda_{\ell+2}, \ldots, \lambda_m$ as the shift values,

$$\mu_{m-\ell} = \lambda_{\ell+1}, \mu_{m-\ell-1} = \lambda_{\ell+2}, \ldots, \mu_1 = \lambda_m, \tag{29}$$

and we are able to extract the unnecessary components of vectors in the direction of the corresponding eigenvectors.

The relationship between H_m and H_m^+ is as follows;

$$Q^+ := Q_1 Q_2 \cdots Q_{m-\ell}, \tag{30}$$

$$V_m^+ := V_m Q^+, \tag{31}$$

$$H_m^+ := (Q^+)^\top H_m Q^+, \tag{32}$$

$$H_m^+ = \begin{bmatrix} * & \cdots & & & & & & \cdots & * \\ * & & & & & & & & \vdots \\ 0 & \ddots & & & & & & & \\ \vdots & \ddots & \ddots & \ddots & & & & & \\ & & & * & * & & & & \\ & & & & 0 & \mu_{m-\ell} & & & \\ & & & & & \ddots & \ddots & & \vdots \\ \vdots & & & & & \ddots & \ddots & \mu_2 & * \\ 0 & \cdots & & & & \cdots & 0 & 0 & \mu_1 \end{bmatrix}. \tag{33}$$

Then, from Eq. (28), we get

$$AV_m Q^+ = V_m H_m Q^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+ \tag{34}$$

$$= V_m Q^+ (Q^+)^\top H_m Q^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+ \tag{35}$$

$$= V_m Q^+ H_m^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+. \tag{36}$$

Therefore, we obtain the following equation;

$$AV_m^+ = V_m Q^+ H_m^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+. \tag{37}$$

From Eqs. (31), (32), and (37), the following relationship from the 1st to the ℓth columns of Eq. (37) is formulated:

$$AV_m^+(:,1:\ell) = V_m^+ H_m^+(:,1:\ell) + \mathbf{v}_m\mathbf{e}_m^\top Q^+(:,1:\ell) \tag{38}$$

$$= V_m^+(:,1:\ell)H_m^+(1:\ell,1:\ell) + h_{\ell+1,\ell}^+\mathbf{v}_{\ell+1}^+\mathbf{e}_\ell^\top + q_{m,\ell}\mathbf{v}_m\mathbf{e}_\ell^\top \tag{39}$$

$$= V_m^+(:,1:\ell)H_m^+(1:\ell,1:\ell) + \mathbf{v}_\ell^+\mathbf{e}_\ell^\top, \tag{40}$$

where $\mathbf{v}_\ell^+ := h_{\ell+1,\ell}^+\mathbf{v}_{\ell+1}^+ + q_{m,\ell}\mathbf{v}_m$. Thus, we are able to restart the Arnoldi decomposition with the initial vector $\mathbf{v}^+$ and Eq. (40).

Algorithm 3 shows the pseudocode of the IRA algorithm.

Moreover, we note that the IRL algorithm is more suitable than the IRA algorithm, when a target matrix is symmetric.

Algorithm 3 IRA algorithm

1: Set m: an upper limit and set ℓ: the number of the desired eigenpairs
2: Input: Arnoldi decomposition $AV_m = V_m H_m + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top$;
3: **for** $i := 1, 2, \cdots$ **do**
4: Compute all the eigenvalues of H_m: $\lambda_1, \ldots, \lambda_m$;
5: Divide eigenvalues: $\lambda_1, \ldots, \lambda_\ell$ and $\lambda_{\ell+1}, \ldots, \lambda_m$;
6: Implicitly shifted QR steps for H_m $m-\ell$ times ($\lambda_{\ell+1}, \lambda_{\ell+2}, \cdots, \lambda_m$ are shift values);
7: $Q^+ = Q_1 Q_2 \cdots Q_{m-\ell}$;
8: $V_m^+ = V_m Q^+$, $H_m^+ = (Q^+)^\top H_m Q^+$;
9: $\mathbf{v}_{m+1}^+ := \mathbf{v}_{m+1}^+ / h_{m+1,m}$;
10: $V_\ell^+ := V_m^+(:,1:\ell), H_\ell^+ := H_m^+(1:\ell,1:\ell)$;
11: $m-\ell$ step Arnoldi algorithm starting with $AV_\ell^+ = V_\ell^+ H_\ell^+ + h_{\ell+1,\ell}\mathbf{v}_{\ell+1}^+\mathbf{e}_\ell^\top$;
12: **end for**

4. Singular Value Decomposition using ARPACK

4.1 Transformation into eigenvalue problem

To apply the IRL algorithm in ARPACK, SVD should be transformed into an eigenvalue problem.

A $w \times n$ ($w \geq n$) rectangular matrix $A^{(r)}$, in which data is stored in row-major order, is decomposed into $A^{(r)} = U^{(r)}\Sigma^{(r)}{V^{(r)}}^\top$. Here, $\Sigma^{(r)}$ is a diagonal matrix whose elements are singular values ${\sigma^{(r)}}_j \geq 0$ ($j : 1 \leq i \leq n$) $\in \mathbb{R}$ of $A^{(r)}$, $U^{(r)} = (\mathbf{u}_1^{(r)}, \mathbf{u}_2^{(r)}, \cdots, \mathbf{u}_w^{(r)}) \in \mathbb{R}^{w\times w}$ is a left orthogonal matrix, in which $\mathbf{u}_j^{(r)} \in \mathbb{R}^w$ corresponding to $\sigma_j^{(r)}$ is aligned, and $V^{(r)} = (\mathbf{v}_1^{(r)}, \mathbf{v}_2^{(r)}, \cdots, \mathbf{v}_n^{(r)}) \in \mathbb{R}^{n\times n}$ is a right orthogonal matrix, in which $\mathbf{v}_j^{(r)} \in \mathbb{R}^n$ corresponding to $\sigma_j^{(r)}$ is aligned. Moreover, the pairs of $(\sigma_j^{(r)}, \mathbf{u}_j^{(r)}, \mathbf{v}_j^{(r)})$ are called singular pairs. Each pair satisfies ${A^{(r)}}^\top \mathbf{u}_j^{(r)} = \sigma_j^{(r)}\mathbf{v}_j^{(r)}$ and $A^{(r)}\mathbf{v}_j^{(r)} = \sigma_j^{(r)}\mathbf{u}_j^{(r)}$.

Among singular pairs $(\sigma_j^{(r)}, \mathbf{u}_j^{(r)}, \mathbf{v}_j^{(r)})$, the following relation holds:

$$\begin{aligned} A^{(r)\top}A^{(r)}\mathbf{v}_j^{(r)} &= A^{(r)\top}\left(\sigma_j^{(r)}\mathbf{u}_j^{(r)}\right) \\ &= \sigma_j^{(r)}\left(A^{(r)\top}\mathbf{u}_j^{(r)}\right) = {\sigma_j^{(r)}}^2\mathbf{v}_j^{(r)}, \end{aligned} \tag{41}$$

$$\mathbf{u}_j^{(r)} = \frac{A^{(r)}\mathbf{v}_j^{(r)}}{||A^{(r)}\mathbf{v}_j^{(r)}||_2}. \tag{42}$$

Eqation (41) shows that the singular value problem of $A^{(r)}$ can be changed to the eigenvalue problem of $A^{(r)\top}A^{(r)}$ algebraically. Namely, the squares of singular values ${\sigma_j^{(r)}}^2$ of $A^{(r)}$ are equal to eigenvalues of $A^{(r)\top}A^{(r)}$. Therefore, when $A^{(r)\top}A^{(r)}$ is the input matrix in the IRL algorithm, the singular values of $A^{(r)}$ and the right singular vectors corresponding to the singular values can also be computed. Moreover, the left singular vectors can be obtained from Eq.(42).

In the case of $n > w$, data in a rectangular matrix $A^{(c)}$ should be stored in column-major order. Note that, the data format in a matrix $A^{(c)\top}$ is the same as that in a matrix $A^{(r)}$. When the values $\sigma_j^{(c)} \in \mathbb{R}$ and the vectors $\mathbf{u}_j^{(c)} \in \mathbb{R}^w$ and $\mathbf{v}_j^{(c)} \in \mathbb{R}^n$ that satisfy $A^{(c)\top}\mathbf{u}_j^{(c)} = \sigma_j^{(c)}\mathbf{v}_j^{(c)}, A^{(c)}\mathbf{v}_j^{(c)} = \sigma_j^{(c)}\mathbf{u}_j^{(c)} (j = 1, \ldots, r, r < w)$ are found, $\sigma_j^{(c)}$ are called the singular values of $A^{(c)}$, and $\mathbf{u}_j^{(c)}$ and $\mathbf{v}_j^{(c)}$ are called, respectively, the left and the right singular vectors corresponding to $\sigma_j^{(c)}$. For a singular pair $(\sigma_j^{(c)}, \mathbf{u}_j^{(c)}, \mathbf{v}_j^{(c)})$, the following relation holds:

$$\begin{aligned} A^{(c)}A^{(c)\top}\mathbf{u}_j^{(c)} &= A^{(c)}\left(\sigma_j^{(c)}\mathbf{v}_j^{(c)}\right) \\ &= \left(A^{(c)}\mathbf{v}_j^{(c)}\right)\sigma_j^{(c)} = \mathbf{u}_j^{(c)}{\sigma_j^{(c)}}^2, \end{aligned} \tag{43}$$

$$\mathbf{v}_j^{(c)} = \frac{\mathbf{u}_j^{(c)}A^{(c)}}{\|\mathbf{u}_j^{(c)}A^{(c)}\|_2}. \tag{44}$$

Eqation (43) shows that the singular value problem of $A^{(c)}$ can be changed to the eigenvalue problem of $A^{(c)}A^{(c)\top}$ algebraically.

4.2 Pseudocode

The discussion of the data format in $A^{(r)\top}A^{(r)}$ is the same as that of $A^{(c)}A^{(c)\top}$. Hence, we explain only the case of $A^{(r)}$.

To employ the IRL algorithm, $A^{(r)\top}A^{(r)}\mathbf{x}$ can be generally computed by using two matrix-vector operations at each iteration. Thus, once $\tilde{\mathbf{x}} = A^{(r)}\mathbf{x}$, $\mathbf{r} = A^{(r)\top}\tilde{\mathbf{x}}$ is computed, where $\tilde{\mathbf{x}} \in \mathbb{R}^w$ and $\mathbf{r} \in \mathbb{R}^n$. In this paper, this implementation is called as the conventional implementation. Algorithm 4 shows the pseudocode of the conventional implementation.

Algorithm 4 Conventional implementation

```
1: x̃ = Ax;
2: r = A^⊤x̃;
```

Algorithm 5 Proposed implementation

```
1: r = 0;
2: #omp parallel for private(t) reduction(+:r)
3: for i = 1 to n do
4:     t = ⟨a_i, x⟩ (a_i = A^(r)(i,:));
5:     r = r + t a_i;
6: end for
7: #omp end parallel for
```

Considering the computational order and the caches of shared-memory multi-core processors, we propose that $A^{(r)\top}A^{(r)}\mathbf{x}$ is computed using the following iteration.

1) $t = A^{(r)}(i,:)\mathbf{x}$
2) $\mathbf{r} = \mathbf{r} + tA^{(r)\top}(i,:)$

Here $A^{(r)}(i,:)$ $(i : 1 \le i \le n) \in \mathbb{R}^n$ is the i-th row vector of $A^{(r)}$. The iteration can be performed efficiently because the data of $A^{(r)}(i,:)$ and $A^{(r)\top}(:,i)$ are the same and have been stored in caches. [1] Consequently, $A^{(r)\top}A^{(r)}\mathbf{x}$ can be computed efficiently on shared-memory multi-core processors with large cashes. In this paper, this implementation is called the proposed implementation. Algorithm 5 shows the pseudocode of the proposed implementation. In the proposed implementation, since the size of an element in $A^{(r)}(i,:)$ is 8bytes, the size of caches needs, theoretically, to be more than $n \times 8$bytes.

4.3 Sparse matrix

The proposed implementation can be made efficient for use in the case of not only dense but also sparse matrices.

In the case of a $w \times n$ ($w \ge n$ and $w < n$) rectangular matrix $A^{(r)}$, these elements should be stored in CRS (compressed row storage) and CCS (compressed column storage) formats [3], respectively. The CRS and CCS formats are the most general [2]. These formats require no assumptions about sparse matrices and any unnecessary elements are not contained in these format.

The CRS format stores only non-zero elements of the matrix rows sequentially. If a non-symmetric sparse matrix $A^{(S)}$ is given, we write the matrix as three vectors $\mathbf{val}_R$, $\mathbf{col\_ind}$ and $\mathbf{row\_ptr}$. $\mathbf{val}_R$ is a vector of floating-point numbers, and stores the value of the non-zero elements of the given matrix $A^{(S)}$ traversal in row-wise order. $\mathbf{col\_ind}$ is a vector of integers indicates the column indices of the

[1] The data in $A^{(r)}(i,:)$ is stored in serial order. Therefore, when n is smaller than the size of caches, all data in $A^{(r)}(i,:)$ is stored in caches at the same time. Since $A^{(c)}(:,j)$ is stored in serial order, the case of $A^{(c)}(:,j)^\top$ is performed in the same way.

elements in $\mathbf{val}_R$. $\mathbf{row\_ptr}$ is also a vector of integers indicates the locations in $\mathbf{val}_R$ that start a row. The last entry of $\mathbf{row\_ptr}$ indicates the number of non-zero elements in the matrix $A^{(S)}$.

The CCS format is equivalent to the CRS format except that the CCS format traverse the non-zero elements of $A^{(S)}$ in column-wise order. In other words, the CCS format can be interpreted as the CRS format for ${A^{(S)}}^\top$. The CCS format is composed of the three vectors $\mathbf{val}_R$, $\mathbf{col\_ind}$ and $\mathbf{row\_ptr}$. $\mathbf{val}_R$, vector of floating-point numbers, stores the value of the non-zero elements of the given matrix $A^{(S)}$ traversal in column-wise order. $\mathbf{row\_ind}$, a vector of integers, indicates the row indices of the elements in $\mathbf{val}_R$. $\mathbf{col\_ptr}$, a vector of integers, indicates the locations in $\mathbf{val}_R$ that start a column. The last entry of $\mathbf{col\_ptr}$ indicates the number of non-zero elements in the matrix $A^{(S)}$.

As an example, consider the non-symmetric matrix $A^{(S')}$ defined by

$$A^{(S')} = \begin{bmatrix} 1 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 0 & -1 & 0 & 3 \\ 2 & 7 & 3 & 2 & 6 & 0 \\ 0 & 3 & 8 & 4 & 0 & 0 \\ 3 & 5 & 0 & 9 & 5 & 9 \\ 0 & 0 & 0 & 0 & 2 & 6 \end{bmatrix}. \tag{45}$$

The CRS format for $A^{(S')}$ is

$$\mathbf{val}_R = \{1, 3, 2, -1, 3, 2, \cdots, 5, 9, 2, 6\}, \tag{46}$$
$$\mathbf{col\_ind} = \{1, 5, 2, 4, 6, 1, \cdots, 5, 6, 5, 6\}, \tag{47}$$
$$\mathbf{row\_ptr} = \{1, 3, 6, 11, 14, 19, 20\}. \tag{48}$$

The CCS format for $A^{(S')}$ is

$$\mathbf{val}_C = \{1, 2, 3, 2, 7, 3, \cdots, 2, 3, 9, 6\}, \tag{49}$$
$$\mathbf{row\_ind} = \{1, 3, 5, 2, 3, 4, \cdots, 6, 2, 5, 6\}, \tag{50}$$
$$\mathbf{row\_ptr} = \{1, 4, 8, 10, 14, 18, 20\}. \tag{51}$$

5. Experiment

We have performed experiments to evaluate the performance of the proposed implementation. The dimension of the sample sparse matrix is 200000×100000 and this matrix is composed of the 1000 non-zero elements in every row. The number of required singular pairs ℓ are 50, 100, 200 and 400 from the maximum singular value of $A^{(r)}$. In ARPACK, the dimension of the Krylov subspace m is determined by the users; in the numerical experiments, m is set to $m = 2\ell$.

Table 1 lists the specifications of the computer used in the experiments.

The size of an element in the matrix $A^{(r)}$ is 64 bits in the case of double precision. Therefore, the size of a row in the matrix $A^{(r)}$ is much smaller than the size of the L3 cache. Consequently, all data in $A^{(r)}(i,:)$ are stored to the caches at the same time. Hence, the iteration of the proposed implementation can be performed efficiently because the data of $A^{(r)}(i,:)$ and ${A^{(r)}}^\top(:,i)$ are the same and have been stored in the caches.

Table 1: Specifications of the experimental environment

	Environment (Appro 2548X), Kyoto University
CPU	Intel Xeon E5-4650L @2.6GHz, 32cores (8 cores ×4) L3 cache: 20MB × 4
RAM	DDR3-1066 1.5TB, 136.4GB/sec
Compiler	Intel C++/Fortran Compiler 14.0.2
Options	-O3 -xHOST -ipo -no-prec-div -mcmodel=medium -shared -intel
Software	Intel Math Kernel Library 11.1.2

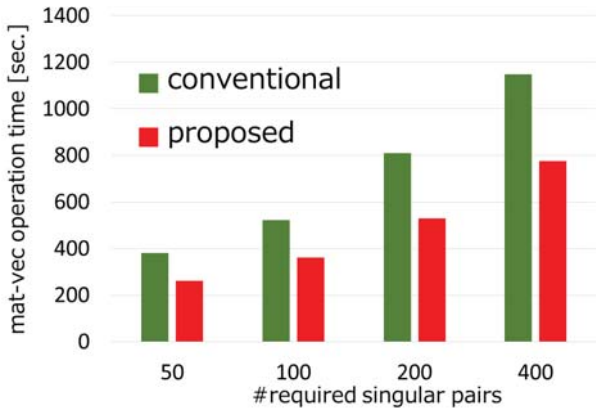


Fig. 1: Computation time of ${A^{(r)}}^\top A^{(r)}\mathbf{x}$ ($A^{(r)}$ is a $200,000 \times 100,000$ real matrix), comparison the conventional and the proposed

Figure 1 shows the results of the experiments. The number of required singular pairs is listed on the horizontal axis, and the vertical axis indicates the computation time for ${A^{(r)}}^\top A^{(r)}\mathbf{x}$. The results show that the computation time for ${A^{(r)}}^\top A^{(r)}\mathbf{x}$ of the proposed implementation is about 80% of that of the conventional implementation.

6. Conclusions

To obtain only the partial singular values and singular vectors of the target matrix, it is effective to use ARPACK, which is known as a solver of eigenvalue problems for large-scale matrices. Therefore, we have transformed SVD problems into eigenvalue problems.

In ARPACK, transformed eigenvalue problems are generally computed by using two matrix-vector operations at each iteration. In the case of large-scale matrices, not all of the elements in the eigenvalue problems can be stored in the caches at the same time. Hence, we have proposed a new implementation, which is introduced from the viewpoint of the computational order and the caches of shared-memory multi-core processors. In the proposed implementation, if only one row in a target matrix can be stored in the caches, high cache hit ratios can be archived.

We performed experiments to evaluate the proposed implementation. In the experiments, we used a machine with 20MB L3 caches. Therefore, the size of a row in a target matrix with dimension size 100000 is much smaller than the size of the L3 cache. The experimental results showed that the computation time of the proposed implementation is about 80% of that of the conventional implementation.

References

[1] W. E. Arnoldi, "The principle of minimized iterations in the solution of the matrix eigenvalue problem," *Quart. Appl. Math.*, vol.9, pp.17-29, 1951.

[2] J. Dongarra, (1995) Templates for the Solution of Linear Systema: Building Blocks for Iterative Methods, [Online]. Available: http://netlib.org/linalg/html_templates/node89.html

[3] I. Duff, R. Grimes, AND J. Lewis, "Sparse matrix test problems," *ACM Trans. Math. Soft.*, vol.15, pp.1-14, 1989.

[4] G. H. Golub and C. F. van Loan, *Matrix Computations*, Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[5] R. B. Lehoucq, D. C. Sorensen, and C. Yang.. (1998) ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods. [Online]. Available: http://www.caam.rice.edu/software/ARPACK

[6] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *J. Res. Nat. Bureau Standards, Sec*, vol.B, no.45, pp.255-282, 1950.

[7] D. C. Sorensen, "Implicit application of polynomial filters in a k-step Arnoldi method," *SIAM J. Matrix Anal. Appl.*, vol.13, pp.357-385, 1992.

[8] D. C. Sorensen D. Calvetti, and L. Reichel, "An Implicitly Restarted Lanczos Method for Large Symmetric Eigenvalue Problems," *Elect. Trans. Numer. Anal.*, vol.2, pp.1-21, 1994.

Generating All Solutions of Minesweeper Problem Using Degree Constrained Subgraph Model

Hirofumi Suzuki, Sun Hao, and Shin-ichi Minato
Graduate School of Information Science and Technology, Hokkaido University

Abstract— *Minesweeper is one of the most popular puzzle game. Several kinds of decision or counting problems on Minesweeper have been studied. In this paper, we consider the problem to generate all possible solutions for a given Minesweeper board, and propose a new formulation of the problem using a graph structure, called* degree constrained subgraph model. *We show experimental results of our efficient graph enumeration techniques for various sizes of Minesweeper boards.*

Keywords: minesweeper, generating, graph model, degree constrained subgraph

1. Introduction

Minesweeper is one of the most popular puzzle game, which is frequently bundled with operating systems and GUIs, including Windows, X11, KDE, GNOME, etc. The objective of this game is to find all hidden mines in covered cells with the some helps of hints.

There are several problems related to Minesweeper, the *Minesweeper consistency problem* [1], the *Minesweeper counting problem* [2], and the *Minesweeper constrained counting problem* [3]. Minesweeper on graph structures are also studied in [3]. These problems ask us whether or not the input Minesweeper board has any solutions, valid assignments of mines.

Those problems was studied as one of decision problems or counting problems. However, the objective of Minesweeper is considered as to really assign some mines to uncovered cells with some constraints. From this viewpoint, we consider a new problem which contains the above problems. This problem requires all solutions of the input Minesweeper board. Solving this problem is useful for finding the best solution with some costed mines, revealing that there is no mine, and calculating the probability of mine placement at each cell.

For finding one solution of the minesweeper, we may use some simple backtracking.search algorithms, however, it is hard to generate all the solutions because of the combinatorial explosion in terms of computation time and space. Recently, Zero-suppressed Binary Decision Diagram (ZDD) [4] is known as a compact representation for manipulating a set of combinations. ZDDs are useful for generating all the solutions for a Minesweeper board. In this method, it is a naive way to use a combinatorial model that one logic variable (combinatorial item) is assigned to each cell, to represent whether a mine exists at the cell or not.

In this paper, we propose yet another formulation using a graph structure, called *degree constrained subgraph model*, and show an efficient method using ZDD-based graph enumeration technique [5]. We experimentally compared performance of the methods based on our graph model and the naive combinatorial model. The result showed that our formulation is effective for the problem.

In section 2, we explain the rules of Minesweeper in detail, and introduce some problems related to Minesweeper. In section 3, we explain the naive combinatorial model using ZDD for finding all valid assignments of mines. In section 4, we show the proposal formulation using degree constrained subgraph model, and explain the method based on graph enumeration technique. In section 5, we show the experimental results. In section 6, we explain the calculation method for mine probability.

2. Problems on Minesweeper

Minesweeper consists of a grid of cells. All cells at the initial board is covered (see Fig.1). At each move, the player may uncovers a cell. There are three types of uncovered cells, *mine cells*, *hint cells*, and *free cells*. A mine cell contains a mine, a hint cell has information about number of mine cells surrounding it (called *count*), and a free cell contains nothing. In this paper, we consider the free cells as the hint cells whose counts are 0, and draw mine as a black circle (•). The goal of the game is to uncover all free cells (see Fig.2). If mine cell was uncovered, the game becomes over and the player loses (see Fig.3).

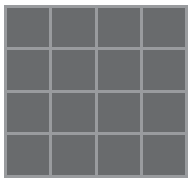


Fig. 1: The initial board.

The *Minesweeper consistency problem* (or simply the *Minesweeper problem*) is a decision problem, whether or not a given Minesweeper grid has a valid assignment of mines (see Fig.4 and Fig.5). The consistency problem was proved

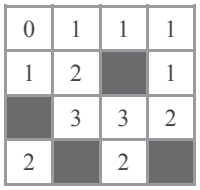


Fig. 2: A winning state.

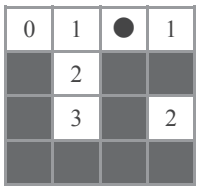


Fig. 3: A lost state.

to be NP-complete [1], even if each cell in the grid has only one mine surrounding it [6]. Counting number of valid assignments to the given Minesweeper grid is also defined as a problem, the *Minesweeper counting problem* (called #Minesweeper in [2]). In setting of Fig.4, the solution for the counting problem is 66. The counting problem was proved to be #P-complete [2]. In [3], *Minesweeper constrained counting problem* is defined. The input of the constrained counting problem also includes the total number of mines.

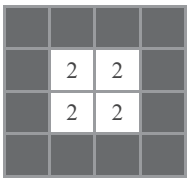


Fig. 4: This setting has valid assignment.

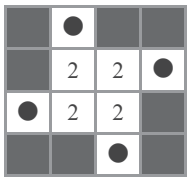


Fig. 5: An example of assignment for Fig.4.

Althought the grid is used for the board in general Minesweeper, we can use a graph structure instead of the board. Each vertex of graph corresponds to a hint cell or a free cell, or a covered cell (see Fig.6). If vertex is a hint cell, it has a count of the number of mines in adjacent vertices, otherwise may have a mine. Then, the Minesweeper on grid boards is considered as the Minesweeper on grid graphs (see Fig.7). Minesweeper on graph structure was studied in [3], and polynomial algorithm is provided for the consistency, counting problem for Minesweeper on trees and on graphs of bounded treewidth.

We consider a new problem for Minesweeper, the required output is all valid assignments of mines for given Minesweeper board. We refer to the problem as *Minesweeper generation problem*. The Minesweeper generation problem is including both the consistency problem and the counting problem.

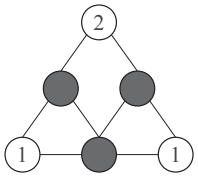


Fig. 6: Minesweeper on graph.

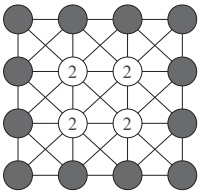


Fig. 7: Graph based on grid board of Fig.4.

In the Minesweeper generation problem, the input is a Minesweeper board $MB(m, n, C, A)$. There is m covered cells numbered from 1 to m, and n hint cells numbered from 1 to n. Note that the covered cell is ignored if hint cell of surrounding it is nothing. C has n integers $c_1, c_2, \ldots, c_n$, and c_i is count written in i-th hint cell ($c_i \geq 0$). A has n sets of integers $A_1, A_2, \ldots, A_n$, and A_i has all numbers of covered cells surrounding i-th hint cell ($A_i \subseteq \{1, 2, \ldots, m\}$).

3. Naive Combinatorial Model Using ZDD

Combination of covered cells correspond to assignment of mines, namely, mines are assigned to all covered cells included in the combination. Hence set of all valid combinations of cells represents solution to Minesweeper generation problem.

Zero-suppressed Binary Decision Diagram (ZDD) [4] is compact data structure for manipulating sets of combinations, and has many application to combinatorial problems [4] [7]. We explain a method for solving the Minesweeper generation problem based on naive combinatorial model using ZDD.

3.1 Zero-suppressed Binary Decision Diagram

ZDD is a compact representation of binary decision tree (see Fig.8 and Fig.9). The tree has a root node and two terminal nodes, a 0-terminal and a 1-terminal. A path which connects the root to the 1-terminal node corresponds to a combination in the set. Each internal node of the tree has a label of an item and two edges, one is 0-edge, the other is 1-edge. The 0-edge represents that the item is not included in the combination, the 1-edge is opposite. In general, the order of appearances of items is fixed.

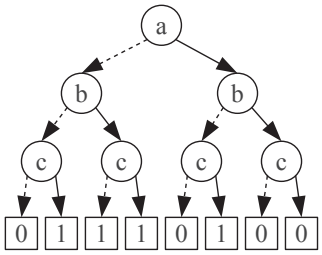


Fig. 8: An example of binary decision tree that represents set of combinations $\{\{a,c\},\{b,c\},\{b\},\{c\}\}$. 0-edge is dotted, and 1-edge is solid.

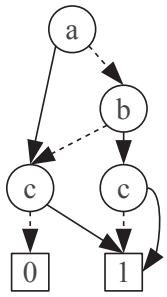


Fig. 9: ZDD for Fig.8.

ZDDs are based on the following reduction rules.

- Deletion rule: delete all redundant nodes whose 1-edge point to the 0-terminal (see Fig.10).
- Sharing rule: share all equivalent subgraphs (see Fig.11).

To make ZDDs compact and canonical for representing sets of combinations, we should apply these reduction rules as much as possible without minding order.

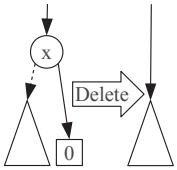


Fig. 10: Deletion rule.

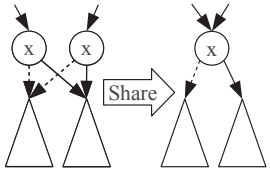


Fig. 11: Sharing rule.

A conventional ZDD package supports various operations for manipulating sets of combinations. The ZDD package maintains a compact data structure in handling those operations. Here we show some operations, which used for generating the solutions of the Minesweeper generation problem. In the following, P and Q indicate an instance of sets of combinations represented by a ZDD, respectively.

$P \cup Q$	the union set of P and Q
	result is $\{c \mid c \in P \text{ or } c \in Q\}$
$P \cap Q$	the intersection set of P and Q
	result is $\{c \mid c \in P \text{ and } c \in Q\}$
$P * Q$	the direct product set of P and Q
	result is $\{p \cup q \mid p \in P \text{ and } q \in Q\}$

3.2 Using ZDD Operation

For representing solutions of Minesweeper using ZDDs, we consider each covered cell as an item. We define U as the set of all items, $x_i \in U$ denote the item of i-th covered cell. A subset $X \subseteq U$, combination of covered cells, corresponds to an assignment of mines. We also defile $M_{valid} \subseteq 2^U$ as the set of combinations that each of them represents valid assignment for a given Minesweeper board. Then, our objective is to generate M_{valid} as output.

We define $M_i \subseteq 2^U$ as the set of all the assignments satisfying the i-th hint. Then, a valid assignment must be commonly included in all M_i $(i = 1, 2, \ldots, n)$. Hence the set of all the valid assignments M_{valid} is represented by:

$$M_{valid} = \cap_{i=1}^{n} M_i.$$

Thus, our method constructs the n ZDDs each of which represent M_i for all i, and calculate the intersection of those ZDDs.

For calculating M_i, we define the set $S(X,k)$ as all combinations of any k items in the item set $X \subseteq U$. For example, $X = \{a,b,c\}$ and $k = 2$, then $S(X,k) = \{\{a,b\},\{a,c\},\{b,c\}\}$. Using notation of set $S(X,k)$, M_i is represented by following, where $X_i = \{x_j \mid j \in A_i\}$ is the set of all covered cells surrounding i-th hint cell.

$$M_i = S(X_i, c_i) * 2^{(U \setminus X_i)}$$

We can construct a ZDD which represents M_i effectively, using memorised recursion technique (see algorithm 1). In algorithm 1, $recursive(i, X, c, m)$ generates a ZDD ret which represents

$$S(X,c) * 2^{(\{x_i,\ldots,x_m\} \setminus X)}.$$

If $x_i \in X$, we can divide the set of combinations represented by ret into two disjoint subsets, one has x_i in each combination:

$$(\{\{x_i\}\} * S(X \setminus \{x_i\}, c-1)) * 2^{(\{x_i,\ldots,x_m\} \setminus X)},$$

and the other has no x_i in each combination:

$$S(X \setminus \{x_i\}, c) * 2^{(\{x_i,\ldots,x_m\} \setminus X)},$$

otherwise ret equals ZDD which represents

$$((\{\{x_i\}\} * S(X,c)) \cup S(X,c)) * 2^{(\{x_i,\ldots,x_m\} \setminus X)}.$$

As a result, we get the algorithm for calculating M_{valid} (see Algorithm 2). In the following, we define $Z(x)$ as ZDD which represents $\{\{x\}\}$, set of combinations consisting of only $\{x\}$.

Algorithm 1 $recursive(i, X, c, m)$

```
if i = m then
  if c = 0 then
    return ZDD consisting of only 1-terminal
  end if
  return ZDD consisting of only 0-terminal
end if
if memo table has a ZDD at (i, c) then
  return ZDD memorised at (i, c)
end if
ret ← null
if x_i ∈ X then
  z1 ← recursive(i + 1, X \ {x_i}, c − 1, m)
  z0 ← recursive(i + 1, X \ {x_i}, c, m)
  ret ← (Z(x_i) ∗ z1) ∪ z0
else
  z ← recursive(i + 1, X, c, m)
  ret ← (Z(x_i) ∗ z) ∪ z
end if
memorise ret at (i, c)
return ret
```

Algorithm 2 The method for Minesweeper generation problem based on naive combinatorial model using ZDD

Input: $MB(m, n, C, A)$, and U
Output: ZDD which represents all valid assignments of MB

```
for i = 1 to n do
  X_i ← {x_j ∈ U | j ∈ A_i}
  initialize memo table for algorithm 1
  Z_i ← recursive(1, X_i, c_i, m)
end for
Z_valid ← ∩_{i=1}^{n} Z_i
return Z_valid
```

This algorithm is based on the naive combinatorial model. An advantage of this model is the simple notation and compact processing with ZDD. However, the algorithm repeats algebraic operations as many as the number of hints, and thus the computation time may become large. As an improvement, we propose yet another formulation in the following section.

4. Graph Model for Minesweeper

We propose a formulation for the problem to find a valid assignment of mines for the input Minesweeper board. In this formulaion, we use graph structure, called degree constrained subgraph model. Then, we show that generating all solutions of the formulated problem is equivalent to generating all valid assignments for the Minesweeper board. For solving the formulated problem, we can use ZDD-based graph enumeration technique.

4.1 Notations and Definitions for Degree Constrained Subgraph Model

In undirected graph $G = (V, E)$, we define d_v as degree of vertex v, number of edges connected with v. We also define d'_v as degree of $v \in V$ in subgraph $G' = (V, E' \subseteq E)$.

We define the degree constraint for vertex v as follows.

$$dc_v \subseteq \mathbb{N} \cup \{0\}$$

dc_v denotes the set of valid degrees of v in subgraph. For given graph G and degree constraints dc_v for all $v \in V$, degree constrained subgraph is a subgraph G' satisfying the following constraints (see Fig.12).

$$d'_v \in dc_v \text{ for all } v$$

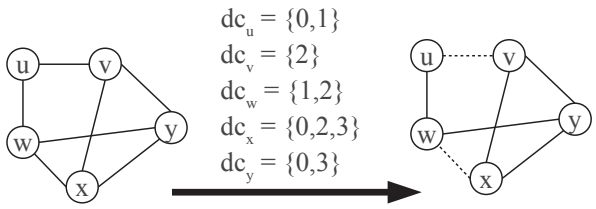


Fig. 12: An example of degree constrained subgraph.

4.2 Formulation

For a given Minesweeper board $MB(m, n, C, A)$, we construct a graph (named MG). We define B as a set of the vertices for covered cells, and $b_i \in B$ means the vertex for the i-th covered cell. We also define H as set of vertices for hint cells. $h_j \in H$ means the vertex of the j-th hint cell. In the following, we call the vertex of covered cell *covered vertex*, and call the vertex of hint cell *hint vertex*. The graph has the set of edges E which connect vertices of adjacent cells, namely, $e = \{b_i, h_j\} \in E$ if $i \in A_j$. Then, $MG = (B \cup H, E)$ is a bipartite graph consisting of the covered vertices and the hint vertices.

To make the correspondence between a mine assignment and a subgraph of MG, we set degree constraints on MG. If we assign a mine to the i-th covered vertices, b_i should connect with all the adjacent hint vertices in the subgraph of MG, otherwise be independent. Then, we get the following degree constraints.

$$dc_{b_i} = \{0, d_{b_i}\} \quad \forall i \in \{1, 2, \ldots, m\} \tag{1}$$

For converting from a subgraph G' of MG under the degree constraints (1) to an assignment of mines, we assign a mine to the i-th covered cell if $d'_{b_i} \neq 0$.

In addition, since our objective is to find a valid assignment of mines, hint vertex h_i should connect with c_i adjacent covered vertices in the subgraph of MG. Then, we get the following degree constraints.

$$dc_{h_j} = \{c_j\} \quad \forall j \in \{1, 2, \ldots, n\} \tag{2}$$

As a result, we also get the following theorem.

Theorem 1 *There is a one-to-one correspondence between the subgraphs of MG under degree constraints* (1) (2) *and the valid assignments for MB.*

Proof. MG has an edge $e = \{b_i, h_j\}$ if and only if the i-th covered cell adjacent to the j-th hint cell. We define MG_{dc} as the set of all the degree constrained subgraphs of MG. We also define MB_{dc} as the set of assignments converted from $\forall G' \in MG_{dc}$, and also define MB_{valid} as the set of all the valid assignments. We should show $MB_{dc} = MB_{valid}$.

First, we show $MB_{dc} \subseteq MB_{valid}$. In $G' \in MG_{dc}$, any covered vertex with a non-zero degree connects to all the adjacent hint vertices. Since each hint vertices must satisfy the degree constraint, for all j, the j-th hint cell has c_j mine cells surrounding it. Thus, all assignments in MB_{dc} is valid, and we get $MB_{dc} \subseteq MB_{valid}$.

Next, we show $MB_{dc} \supseteq MB_{valid}$. We consider an induced subgraph of MG based on a valid assignment in MB_{valid}, and we call the subgraph MG_{ind}. MG_{ind} has the subset of covered vertices $B' \subseteq B$ and all the hint vertices H, and B' consists of the covered vertices that there is a mine in corresponding cell, then MG_{ind} has the set of edges $E' = \{e | e \in E$ and $b \in e$ and $b \in B'\}$. Thus, in G_{ind}, the degree of $b \in B'$ is d_b and the degree of $b \notin B'$ is 0, then all covered vertices satisfy the degree constraints. In addition, since the assignment is valid and each hint vertex connects all adjacent covered vertexes whose degree is not zero, the degree of $h_j \in H$ is equivalent to c_j. Thus, all the hint vertices also satisfy the degree constraints. Hence $MG_{ind} \in MG_{dc}$, and we get $MB_{dc} \supseteq MB_{valid}$.

Thus, the theorem follows. □

4.3 Using Graph Enumeration Technique

By the theorem 1, the Minesweeper generation problem is solved by generating all solutions of the formulated problem. Here we can use an efficient ZDD-based graph enumeration technique shown in [5], *frontier-based search method*. The frontier-based search generates a ZDD which represents all the subgraphs as the combinations of edges, satisfying various topological constraints, for example paths, cycles, trees, forests, and the degree constraints (see Fig.13).

Frontier-based search begin construction of ZDD with only the root node, and advance the search by top-down manner which depends on the order of edges; after deciding order of the edges, in i-th step, all the bottom nodes branch off, and make two child nodes, one correspond to the case of using i-th edge, and the other correspond to the case of not using i-th edge. In addition, frontier-based search prune some unnecessary nodes, and merge some equal nodes. These processes are realized by the supporting infomation of the search which is called $mate$ (we leave the details to reference [5]).

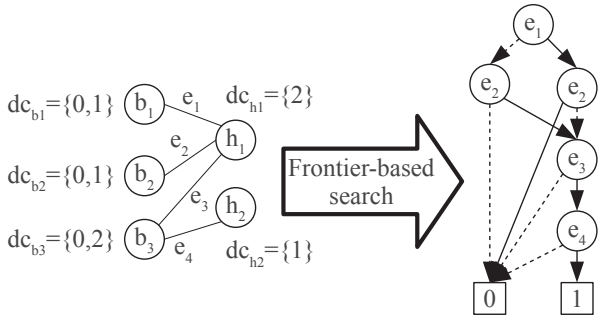


Fig. 13: An example of frontier-based search.

The coventional enumeration algorithms output all solutions one by one, and thus it is hard to generate all the solutions because of the combinatorial explosion in terms of computation time and space. In contrast, the frontier-based search method output a large number of solutions as a compact ZDD to avoid combinatorial explosion in many cases, and the computation time and space depends on the size of ZDD.

5. Computational Experiments

We experimentally compared the performance of the method using degree constrained model and the method using the naive combinatorial model (which based on ZDD operations). Especially, we use not only grid based Minesweeper board but also graph based Minesweeper board, to compare them under various situations. The program was coded in C++, and compiled using g++. The experiments were done on the PC with Intel Core i7-3930K 3.2GHz CPU and 64GB memory.

The instance boards were randomly generated boards. Some of them are based on 30×30 grid, and the others are based on randomly generated graph which has 300 vertices and 900 edges (relatively sparse). We set three types of ratio of mine cells, 10%, 20%, and 30%, and set nine types of ratio of visible hints, 10%, 20%, ..., and 90%. Tables 1 and 3 summarize the computation time. 'zdd' indicates the method based on naive combinatorial model using ZDD, and 'dc' indicates the method based on graph enumeration technique using degree constrained subgraph model. In addition, tables 2 and 4 summarize the number of valid assignments in each instance. The computation time of the latter includes time

of constructing graph and degree constraints. The best time is written in bold letters.

The computation time results for grid based boards are shown in table 1, and table 2 shows the number of valid assignments in each instance. The number of valid assignments is over 10^{20} in quite of half instances. However, 'dc' shows the best time in most instances, and it is 100 times faster than 'zdd' in some instances. Thus, our formulation is efficient for the Minesweeper generation problem with board based on grid. But in instance consisting of 30% mine cells and 40% visible hits, 'dc' is inefficient in comparison with 'zdd'. It is thought that the reason depends on the complexity of the graph generated by the board; it is supposed that the degree constraints is easily complicated by the state of the connection of the vertexes.

Table 1: Computation time (in second) for grid based board

mine	10%		20%		30%	
hint	zdd	dc	zdd	dc	zdd	dc
10%	3.602	**0.006**	2.917	**0.006**	2.872	**0.005**
20%	15.306	**0.023**	16.124	**0.095**	16.921	**0.024**
30%	19.812	**0.149**	23.974	**0.732**	22.268	**3.472**
40%	24.636	**0.051**	34.478	**1.226**	**36.521**	79.872
50%	23.334	**0.041**	33.440	**0.331**	37.740	**1.884**
60%	18.915	**0.033**	23.872	**0.065**	35.572	**0.273**
70%	11.826	**0.028**	17.801	**0.031**	23.063	**0.035**
80%	6.013	**0.019**	12.659	**0.024**	18.790	**0.030**
90%	1.788	**0.013**	6.128	**0.018**	11.642	**0.023**

Table 2: Number of valid assignments for grid based board

hint\mine	10%	20%	30%
10%	1.53×10^{30}	6.33×10^{47}	1.92×10^{55}
20%	3.76×10^{30}	1.35×10^{55}	8.41×10^{71}
30%	1.12×10^{20}	8.29×10^{37}	2.81×10^{51}
40%	4.09×10^{7}	5.24×10^{21}	5.90×10^{39}
50%	512	1.73×10^{7}	6.89×10^{25}
60%	16	65536	3.52×10^{6}
70%	1	64	6912
80%	1	2	4
90%	2	1	1

The computation time results for graph based boards are shown in table 3, and table 4 shows the number of valid assignments in each instance. 'dc' shows the best time in all instances, and it is 100 times faster than 'zdd' in some instances. The computation time of 'zdd' is not stable compared with grid instances. It is thought that this result is caused by dispersion of the degree in the original graph, which affect number of adjacent cells. In contrast, number of adjacent cells of each cell in grid is approximately constant. On the other hand, the computation time of 'dc' is stable. Thus, our formulation is also efficient for the Minesweeper generation problem with board based on sparse graph.

Table 3: Computation time (in second) for graph ($|V| = 300, |E| = 900$) based board

mine	10%		20%		30%	
hint	zdd	dc	zdd	dc	zdd	dc
10%	0.171	**0.001**	0.629	**0.002**	52.186	**0.001**
20%	0.816	**0.004**	89.602	**0.024**	82.493	**0.141**
30%	1.156	**0.008**	100.224	**0.025**	49.926	**0.650**
40%	5.859	**0.012**	12.410	**0.071**	60.105	**2.592**
50%	1.189	**0.008**	81.190	**0.007**	65.981	**0.017**
60%	0.917	**0.006**	2.065	**0.007**	147.352	**0.234**
70%	0.461	**0.006**	0.799	**0.008**	2.206	**0.008**
80%	0.203	**0.005**	0.500	**0.008**	0.737	**0.007**
90%	0.051	**0.003**	0.184	**0.005**	0.398	**0.005**

Table 4: Number of valid assignments for graph based board

hint\mine	10%	20%	30%
10%	10251360	3.63×10^{9}	1.23×10^{15}
20%	2419200	7.65×10^{10}	1.75×10^{14}
30%	4	7.97×10^{7}	7.15×10^{10}
40%	90	11520	1.28×10^{8}
50%	4	12	864
60%	1	4	4
70%	1	1	2
80%	1	1	1
90%	1	1	1

6. Analysis of Mine Probability

Calculating the probability of mine placement on each cell is useful for considering a strategy of playing Minesweeper. We can easily calculate it after generating a ZDD of all the solutions.

We define C as the number of all valid assignments , and also define C_i as the number of valid assignments whose i-th covered cell is mine. Then, following formula calculates the mine probability for i-th cell (we call R_i).

$$R_i = \frac{C_i}{C} \tag{3}$$

We can calculate the cardinality of a set of combinations in a linear time for the ZDD size, and we may use various algebraic operations for extracting a set of combinations which satisfies some additional constraints.

The items in a ZDD generated by frontier-based search corresponds to the edges in the input graph. If the i-th covered cell has a mine in some assignments, the subgraphs of those assignments are sure to use all edges which connect b_i. Then, we get following algorithm 3 for calculating (3).

7. Conclusion

In this paper, we considered the Minesweeper generation problem to generate all possible solutions for a given Minesweeper board, and proposed a formulation of the problem using degree constrained subgraph model. For solving the problem, we used ZDD-based graph enumeration techniques. Experimental results showed that our formulation is effective for many instances of the Minesweeper boards.

Algorithm 3 The method for calculating mine probability of the i-th cell.

Input: $MG = (B \cup H, E)$, Z_{all} (ZDD of all valid assignments), i

Output: mine probability of i-th cell

1: $L \leftarrow$ cardinality of Z_{all}
2: $e' \leftarrow \forall e \in E$ which satisfies $b_i \in e$
3: $Z_i \leftarrow Z_{all} \cap (Z(e') * 2^{(E \setminus \{e'\})})$
4: $C_i \leftarrow$ cardinality of Z_i
5: **return** $\frac{C_i}{C}$

As a future work, we can also consider an online problem for Minesweeper, where the hints are given one by one. Solving this online problem is close to actual play of the Minesweeper, and an efficient online method is desired.

Acknowledgment

For implementing frontier-based search, we coded the program based on the software library *TdZdd* (in https://github.com/kunisura/TdZdd, [8]). Our work is partly supported by JSPS KAKENHI Scientific Research(S) - Number 15H05711.

References

[1] R. Kaye, *Minesweeper is NP-complete*, ser. The Mathematical Intelligencer. Springer-Verlag, 2000, vol. 22, pp. 9–15.
[2] P. Nakov and Z. Wei, "MINESWEEPER, #MINESWEEPER," http://www.minesweeper.info/articles, 2003.
[3] S. Golan, *Minesweeper on graphs*, ser. Applied Mathematics and Computation, 2011, vol. 217, pp. 6616–6623.
[4] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," *Proc. of 30th ACM/IEEE Design Automation Conf. (DAC 1993)*, pp. 272–277, 1993.
[5] J. Kawahara, T. Inoue, H. Iwashita, and S. Minato, "Frontier-based search for enumerating all constrained subgraphs with compressed representation," Hokkaido University Graduate School of Infomation Science and Technology, Tech. Rep., Sept. 2014.
[6] J. D. Fix and B. McPhail, "Offline 1-minesweeper is np-complete," http://www.minesweeper.info/articles, 2004.
[7] O. Coudert, "Solving graph optimization problems with zbdds," in *European Design and Test Conference*, Mar. 1997, pp. 224–228.
[8] H. Iwashita and S. Minato, "Efficient top-down zdd construction techniques using recursive specifications," Hokkaido University Graduate School of Infomation Science and Technology, Tech. Rep., Dec. 2013.

Visualizing Intrinsic Space for Spatial Data via Input Regularized Gaussian Process Latent Variable Models

Tomoharu Iwata and Naonori Ueda
NTT Communication Science Laboratories, Kyoto, Japan

Abstract— *We propose the input-regularized Gaussian process latent variable model for visualizing a latent intrinsic input space that improves interpolation performance in regression tasks. The proposed model assumes that a latent location is associated with each observed input location, and the covariance function is determined by distance between the latent locations. The latent locations are estimated so that the output covariance of the given data is appropriately captured by the latent locations while preserving the neighbor relationships between the observed input space and the latent space by input regularization. When the input regularization is omitted, the proposed model reduces to the Gaussian process latent variable model. When the input regularization is strong enough to perfectly preserve the neighbor relationships, the proposed model becomes Gaussian process regression. The degree of the regularization is controlled by a hyperparameter, which can be automatically selected by cross-validation using the given data. We demonstrate the effectiveness of the proposed model with real-world spatial data sets in terms of interpolation performance of multiple output values.*

Keywords: Gaussian processes, latent variable models, visualization, spatial data

1. Introduction

Analyzing spatial data is an important task in a wide variety of fields such as geology, ecology, climatology, sociology and urban planning. Gaussian process regression [1], or which is known as Kriging [2] in geostatistics, is a representative method for analyzing and interpolating spatial data. In Gaussian process regression, a covariance function plays a crucial role to define its behavior. With spatial data, kernels that solely depend on distance between two locations, e.g. Gaussian kernels, are usually used for covariance functions, since closely located points are assumed to have similar output values. However, some closely located locations can have different output values, and distant locations can have correlated output values. For example, weather at coastal area would be different from inland area in the same city, people's cultural behavior in two cities divided by a river would be dissimilar, and seismic activity would be related in distant areas when they share a fault line. In other words, the observed input space would be different from its intrinsic space that reflects relationships among locations. Revealing the intrinsic space is beneficial to understand the given spatial data as well as to improve interpolation performance.

In this paper, we propose the input-regularized Gaussian process latent variable model (IGPLVM) that visualizes the latent intrinsic space. The proposed model assumes that each observed input location has its own latent location, and the covariance function is determined by distance between the latent locations. Since neighbor relationships in the latent space should be similar to those in the observed input space as long as the output covariance of the given data is captured by the latent locations, latent locations are regularized to preserve the neighbor relationships. When the regularization is omitted, the proposed model becomes the Gaussian process latent variable model (GPLVM) [3], [4], which is an unsupervised method that learns latent locations using output values without input information. When the regularization is strong enough to perfectly preserve the neighbor relationships, it corresponds to Gaussian process regression, where input locations are assumed to be noise free. The proposed model adaptively uses input information so that output variables are modeled properly.

The proposed model improves interpolation performance by flexibly defining the covariance function by adjusting latent locations. Since the proposed model learns a common latent intrinsic space shared by multiple output variables, it can be used for multi-task learning. The latent space learned by using output values in a task helps to interpolate output values in another related task, even if data are too sparse to learn latent locations with a single task. Note that, although we motivate the proposed model for analyzing spatial data, the proposed model is applicable to other data for any regression problems where input and output values are contained, such as time-series, spatio-temporal, high dimensional input, and multiple output data.

The remainder of this paper is organized as follows. In Section 2, we present our task and introduce Gaussian process regression, on which the proposed model is based. In Section 3, we formulate the proposed model and provide its learning procedures. We review related work in Section 4. In Section 5, the effectiveness of the proposed method is demonstrated by experiments with real-world spatial data sets in terms of interpolation performance of multiple output values. Finally, we present concluding remarks and a discussion of future work in Section 6.

2. Preliminaries

Suppose that we have a set of input and output instances, $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{R}^L$ and $\mathbf{y}_n \in \mathbb{R}^D$. For example, in a case of spatial climate data, $\mathbf{x}_n$ is th nth location vector such as latitude and longitude, and $\mathbf{y}_n$ is an observed weather condition vector at the location such as temperature, humidity and precipitation.

With standard regression methods, an output $\mathbf{y}_n$ is assumed to be generated by mapping input $\mathbf{x}_n$ using nonlinear functions,

$$y_{nd} = f_d(\mathbf{x}_n) + \epsilon, \tag{1}$$

where $f_d(\cdot)$ is the nonlinear function for the dth feature, and $\epsilon \sim \mathcal{N}(0, \beta^{-1})$ is a Gaussian noise. Gaussian process regression uses a Gaussian process for a prior distribution of the nonlinear function,

$$f_d(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \tag{2}$$

where $m(\mathbf{x})$ is a mean function, which is often set to zero, and kernel function $k(\mathbf{x}, \mathbf{x}')$ specifies the covariance of outputs between two locations as follows,

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f_d(\mathbf{x}) - m(\mathbf{x}))(f_d(\mathbf{x}') - m(\mathbf{x}'))]. \tag{3}$$

Since the kernel determines the behavior of the nonlinear functions, it is important to use an appropriate kernel for the given data. For real-valued input data, such as time-series and spatial data, kernels that are negatively correlated to distance between two locations $\parallel \mathbf{x} - \mathbf{x}' \parallel$ are often used, such as Gaussian kernels. However, output values can be different even if two locations are close together, and they can be similar even if two locations are far apart.

3. Input-regularized Gaussian Process Latent Variable Models

We propose the input-regularized Gaussian process latent variable model (IGPLVM), which is a method to obtain kernels that appropriately capture the output covariance between inputs by distorting the input space in the Gaussian process regression framework. The distorted input space reveals intrinsic characteristics of the input space.

The proposed model assumes that an output $\mathbf{y}_n$ is generated from its latent intrinsic location $\mathbf{z}_n \in \mathbb{R}^K$, instead of input location $\mathbf{x}_n$, as follows,

$$y_{nd} = f_d(\mathbf{z}_n) + \epsilon. \tag{4}$$

The dimensionality of the latent space K can be different from that of the input space L. Then, the probability of the output observations $\mathbf{Y} = (\mathbf{y}_1, \cdots, \mathbf{y}_N)^\top$ given the latent locations $\mathbf{Z} = (\mathbf{z}_1, \cdots, \mathbf{z}_N)^\top$, by integrating out the nonlinear functions, is given by

$$p(\mathbf{Y}|\mathbf{Z}, \boldsymbol{\theta}) = (2\pi)^{-\frac{DN}{2}} |\mathbf{K}|^{-\frac{D}{2}} \exp\left(-\frac{1}{2}\mathrm{tr}(\mathbf{Y}^\top \mathbf{K}^{-1} \mathbf{Y})\right), \tag{5}$$

where $\mathbf{K}$ is the $N \times N$ covariance matrix defined by the kernel function $k(\mathbf{z}_n, \mathbf{z}_{n'})$, and $\boldsymbol{\theta}$ is the kernel hyperparameter vector. In this paper, we use a Gaussian kernel with an additive noise term,

$$k(\mathbf{z}_n, \mathbf{z}_{n'}) = \alpha \exp\left(-\frac{1}{2\ell^2} \parallel \mathbf{z}_n - \mathbf{z}_{n'} \parallel^2\right) + \delta_{nm}\beta^{-1}, \tag{6}$$

where $\delta_{nm} = 1$ if $n = m$ and $\delta_{nm} = 0$ otherwise, and $\boldsymbol{\theta} = (\alpha, \ell, \beta)$ are the kernel parameters. Note that the GPLVM [3] finds latent locations $\mathbf{Z}$ that minimizes the negative log likelihood of (5),

$$E_{\mathbf{Y}} = -\log p(\mathbf{Y}|\mathbf{Z}, \boldsymbol{\theta}), \tag{7}$$

where input information $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_N)^\top$ is not used.

We assume that neighbor relationships in the latent space should be similar to those in the input space as long as the output covariance that is specified by the latent space is appropriate to the given data. The proposed method models the neighbor relationships by defining a probability of being selected as neighbors as defined in stochastic neighbor embedding [5], [6]. In the latent space, the probability that n selects n' as its neighbors is given by

$$p(n'|n, \mathbf{Z}) = \frac{\exp(-\frac{1}{2} \parallel \mathbf{z}_n - \mathbf{z}_{n'} \parallel^2)}{\sum_{n'' \neq n} \exp(-\frac{1}{2} \parallel \mathbf{z}_n - \mathbf{z}_{n''} \parallel^2)}, \tag{8}$$

where locations with small Euclidean distance $\parallel \mathbf{z}_n - \mathbf{z}_{n'} \parallel$ in the latent space are likely to be selected as its neighbors. Similarly, in the input space, the neighborhood probability is given by

$$p(n'|n, \mathbf{X}) = \frac{\exp(-\frac{1}{2} \parallel \mathbf{x}_n - \mathbf{x}_{n'} \parallel^2)}{\sum_{n'' \neq n} \exp(-\frac{1}{2} \parallel \mathbf{x}_n - \mathbf{x}_{n''} \parallel^2)}. \tag{9}$$

The neighbor relationships are preserved when these two probabilities are matched. This is achieved by minimizing the following sum of Kullback-Leibuler divergences between the probabilities,

$$E_{\mathbf{X}} = \sum_{n=1}^{N} \sum_{n' \neq n} p(n'|n, \mathbf{X}) \log \frac{p(n'|n, \mathbf{X})}{p(n'|n, \mathbf{Z})}. \tag{10}$$

The proposed method finds latent locations that properly capture the output covariance while preserving the neighbor relationships by minimizing the following sum of (7) and (10),

$$E = E_{\mathbf{Y}} + \lambda E_{\mathbf{X}}, \tag{11}$$

where $\lambda > 0$ is a hyperparameter that controls how the neighbor relationships are preserved. When $\lambda = 0$, it corresponds to GPLVM. When $\lambda = \infty$ and dimensionality of the latent space is the same with the input space $K = L$, it corresponds to Gaussian process regression since the latent locations become the same with the input locations $\mathbf{Z} = \mathbf{X}$. The proposed method can be seen as a multi-task

learning method based on Gaussian process regression. Since the latent locations are learned by using all of the output variables, the learned covariance matrix can improve multi-task regression performance when the output variables are related.

A local optimum solution of latent locations $\mathbf{Z}$ and kernel hyperparameters $\boldsymbol{\theta}$ is obtained by minimizing (11) using gradient-based optimization methods such as the quasi-Newton method [7]. The gradients of the GPLVM term $E_{\mathbf{Y}}$ with respect to a latent location are calculated by

$$\frac{\partial E_{\mathbf{Y}}}{\partial \mathbf{K}} = \frac{1}{2} D \mathbf{K}^{-1} - \frac{1}{2} \mathbf{K}^{-1} \mathbf{Y} \mathbf{Y}^{\top} \mathbf{K}^{-1}, \tag{12}$$

$$\frac{\partial k(\mathbf{z}_n, \mathbf{z}_{n'})}{\partial \mathbf{z}_n} = -\frac{\alpha}{\ell^2} \exp\left(-\frac{1}{2\ell^2} \parallel \mathbf{z}_n - \mathbf{z}_{n'} \parallel^2 (\mathbf{z}_n - \mathbf{z}_{n'}) \right), \tag{13}$$

using the chain rule. The gradients of the regularization term $E_{\mathbf{X}}$ with respect to a latent location are calculated by

$$\frac{\partial E_{\mathbf{x}}}{\partial \mathbf{z}_n} = \sum_{n' \neq n} \Big(p(n'|n, \mathbf{X}) - p(n'|n, \mathbf{Z}) \Big) (\mathbf{z}_n - \mathbf{z}_{n'}), \tag{14}$$

where a sum of forces pulling or pushing $\mathbf{z}$ depends on difference of the neighborhood probability $p(n'|n, \mathbf{X}) - p(n'|n, \mathbf{Z})$.

We can select the hyperparameter value λ by cross-validation on an interpolation problem of the output matrix $\mathbf{Y}$. The cross-validation procedure is as follows. The elements of the output matrix are randomly split into multiple subsets. While the elements in a subset are supposed to be missing, the latent locations and kernel hyperparameters are estimated with a fixed hyperparameter value λ. The interpolation performance of the learned model is evaluated by the rooted mean squared error for the missing elements. The error is averaged over different subsets, and a hyperparameter value that achieved the lowest error is selected. When y_{nd} is missing, its estimated value is given by $\hat{y}_{nd} = \mathbf{k}_n^{\top} \mathbf{K}^{-1} \mathbf{y}_d$, where $\mathbf{k}_n = (k(\mathbf{z}_n, \mathbf{z}_1), \cdots, k(\mathbf{z}_n, \mathbf{z}_N))^{\top}$ and $\mathbf{y}_d = (y_{1d}, \cdots, y_{Nd})^{\top}$, and they are calculated using the observed data. Note that even when all of the output variables are missing with an instance, the latent location can be estimated by using its neighbor relationships.

The proposed model can be seen as a probabilistic generative model. An output value y_{nd} is generated from a Gaussian distribution with mean $f_d(\mathbf{z}_n)$ using the latent location $\mathbf{z}_n$, where the nonlinear function $f_d(\cdot)$ is generated from a Gaussian process prior. The input locations $\mathbf{x}_n$ are not directly generated, but we can consider that the neighbors of the input locations are generated by a multinomial distribution, where the multinomial parameters are defined by latent locations $\mathbf{Z}$ as in (8).

Although we considered that the input variables are real-valued, the proposed method is applicable to other types of input variables that can calculate similarity between two input locations to define neighbor relationships in (9). For example, we can use normalized tree kernels and graph kernels for tree and graph data, respectively, instead of normalized Gaussian kernels in (9).

4. Related Work

With Gaussian process regression, the covariance function is usually defined parametrically, e.g. Gaussian kernels, and its hyperparameters are estimated, e.g. length-scale and noise variance in Gaussian kernels, by maximizing the marginal likelihood. However, the hyperparameter tuning would not be enough to capture the covariance structure when the intrinsic space is different from the observed input space. The proposed model can define the covariance function by flexibly tuning latent locations. A complex form of covariance functions is learned by using multiple kernels [8], by concatenation different kernels [9], and by modeling a spectral density with Gaussian mixtures [10]. These methods are not designed for revealing the latent intrinsic space.

The manifold Gaussian process [11] is related to our work. It transforms input locations by multi-layer neural networks so as to model complex nonsmooth functions. Since the proposed model optimizes a latent location for each input location nonparametrically so as to preserve the neighbor relationships, it can be used for visualizing the latent intrinsic space. In addition, by tuning a single hyperparameter, the proposed model can become the GPLVM and Gaussian process regression, which are successfully used in a wide variety of applications [12], [13], [14], [15]. The warped Gaussian process [16] transforms the output space. Gaussian process regression with input noise [17] assumes that each input location is independently corrupted by Gaussian noise, and does not give latent locations explicitly. On the other hand, with the proposed model, distortion of each location is not independent since latent locations are estimated so as to preserve the neighbor relationships. This enables us to estimate latent locations even without output values.

The GPLVM [3] is an unsupervised method for imputating output values by assuming latent locations. The GPLVM is not designed for regression tasks, and it does use input information. The discriminative GPLVM [18] is a type of regularized GPLVMs. It is used for classification, and outputs are required to be categorical variables. The Gaussian process latent variable set model [19] is a regression method that learns latent locations to model a flexible covariance function. It assumes bag-of-features input, and each feature has a latent location. On the other hand, the proposed model can take any input type, such as real-valued, tree, graph, bag-of-features data, as long as similarity can be calculated, and each instance has a latent location.

Many visualization methods have been proposed, such as multidimensional scaling, principle component analysis, GPLVM, Isomap [20], Laplacian eigenmap [21] and stochastic neighbor embedding (SNE) [5]. They embeds

high dimensional unlabeled data into a low dimensional space. Supervised visualization methods have been also proposed, such as canonical correlation analysis (CCA), kernel CCA [22], Fisher linear discriminant analysis and discriminant GPLVM. For example, CCA embeds input and output pairs into a low dimensional space so as to maximize the correlation. The proposed model is a supervised visualization method that reveals an intrinsic input space for better output interpolation. The proposed model can be seen as a combination of GPLVM and SNE, where GPLVM is used for modeling output values, and SNE is used for preserving the neighbor relationships of input locations. In particular, the objective function is simply defined by a weighed sum of the objective functions of GPLVM and SNE.

5. Experiments

5.1 Data

We evaluated the proposed method by using a real-world spatial data set: the comprehensive climate data of North America (NA) [1]. The NA data set consists of monthly climate reports from 1990 to 2002 [23], [24]. We used 16 output variables, such as carbon dioxide and temperature, which were interpolated on 2.5×2.5 degree grid with 125 locations. We normalized all output values to mean zero and unit variance, and conducted experiments with data for each month, where the input variables were latitude and longitude.

5.2 Comparing Methods

We compared the proposed IGPLVM (Proposed) with the following five methods: GP, MTGP, GPLVM, MF and KNN. GP is a Gaussian process regression method, which assumes that output values are determined by the input locations using nonlinear functions with Gaussian process priors. MTGP is a multi-task Gaussian process regression method [25], which learns relationships between output variables. We used the code provided by the authors[2]. GPLVM is a Gaussian process based nonlinear matrix imputation method [3]. MF is a matrix factorization method [26]. It imputates missing values by the product of two low-rank matrices. Both GPLVM and MF do not use input information. KNN is a k-nearest neighbor regression method, which estimates a missing value by the average value of its four neighbors. The dimensionality of the latent space with the proposed method, GPLVM and MF was set at $K = 2$, which is the same with that of the input space. With the proposed method, we selected a hyperparameter λ for each output variable by five-fold cross-validation from $\{0, 1, 10, 10^2, 10^3, 10^4, 10^5\}$. The latent locations were initialized by the input locations.

[1] http://www-bcf.usc.edu/~liu32/data/NA-1990-20002-Monthly.csv

[2] https://github.com/ebonilla/mtgp

5.3 Results

We measured the effectiveness of the proposed method by interpolation tasks. Ten percent of output values were randomly selected as test data. The performance was evaluated by rooted mean squared error (RMSE). Table 1 shows the RMSE with the NA data set. The proposed method achieved the lowest average RMSE. This result indicates that the output covariance is properly modeled by distorting the input space with the proposed method. GP achieved low RMSE with output variables whose covariance is determined by the input locations, such as WET and DTR. GPLVM achieved low RMSE with output variables which can be estimated easily from other output variables. Since the proposed method can become the GP and GPLVM by controlling the hyperparameter for each output variable, the RMSE of the proposed method was low for all output variables.

The computational time of the proposed method was 18 seconds with a one-month data using a computer with Xeon 7350 2.93GHz CPU.

Figure 1 shows the original input space and the estimated latent space by the proposed method. When $\lambda = 0$, the latent locations were different from the input locations, since the latent locations did not regularized by the input locations at all. When $\lambda = 10^5$, the latent space was the same with the input space, since the effect of preserving neighbor relationships became dominant. With $\lambda = 10$ and $\lambda = 10^3$, the some neighbor relationships were preserved but some latent locations were transformed so as to model the output covariance. The latent locations of west coast were separated from the other locations (b,c,d), because west coast exhibits different weather from the other area.

6. Conclusion

In this paper, we have proposed a probabilistic model for discovering a latent intrinsic space. The proposed method is based on Gaussian processes, where a latent location is associated with each input location, and output values are determined by the latent locations. The latent locations are estimated so as to preserve the neighbor relationships as well as to capture the output covariance of the given data. Although our results have been encouraging, our framework can be further improved upon in a number of ways. Firstly, we would like to estimate the hyperparameter for controlling input regularization by using the variational Bayesian framework [27]. Secondly, we plan to estimate relationships among output variables using multi-task learning techniques [25]. With the proposed method a single latent space is shared by all output variables. By using estimated task relationships, we can obtain multiple latent spaces that capture the characteristics of individual output variables, and it leads to better interpolation performance for missing values. Finally, we would like to extend our method more scalable by using scalable Gaussian process techniques [28], [29].

Table 1: RMSE on interpolation tasks with the NA data set for each output variable averaged over all locations and all timestamps. The last row shows the RMSE averaged over all output variables. Values in bold typeface are statistically better at the 5% level from those in normal typecface as indicated by a paired t-test.

	Proposed	GP	MTGP	GPLVM	MF	KNN
CO2	**0.0532**	0.0536	1.3110	0.2342	0.5448	0.1474
CH4	**0.0520**	0.0528	1.3751	0.1966	0.4935	0.1473
CO	**0.0496**	0.0501	1.3807	0.2007	0.5150	0.1411
H2	**0.0479**	0.0484	1.3669	0.1614	0.4116	0.1358
WET	**0.3672**	**0.3684**	1.3502	0.5033	0.6242	0.3773
CLD	**0.2483**	0.2543	1.3201	0.3580	0.5253	0.2817
VAP	**0.1911**	0.2071	1.3712	0.2511	0.3957	0.2495
PRE	0.5289	0.5201	1.3420	0.5991	0.6857	**0.5098**
FRS	0.3502	0.4480	1.3528	**0.3371**	0.5613	0.4409
DTR	**0.4579**	**0.4611**	1.3947	0.5098	0.5873	0.4704
TMN	**0.1832**	0.3315	1.3694	0.2039	0.3661	0.3531
TMP	**0.1568**	0.3190	1.3786	0.1762	0.3294	0.3419
TMX	**0.1866**	0.3319	1.3706	0.2082	0.3435	0.3578
GLO	**0.1901**	**0.1901**	1.3755	0.2755	0.3618	0.2311
ETR	**0.0641**	0.0647	1.3677	0.1376	0.3714	0.1733
ETRN	**0.0568**	0.0572	1.3034	0.1239	0.3190	0.1547
Average	**0.1990**	0.2349	1.3581	0.2798	0.4647	0.2821

Acknowledgments

A part of the research results has been achieved by "Research and Development on Fundamental and Utilization Technologies for Social Big Data," the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN.

References

[1] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[2] N. Cressie, *Statistics for spatial data*. Wiley New York, 1993.

[3] N. D. Lawrence, "Gaussian process latent variable models for visualisation of high dimensional data," in *Advances in Neural Information Processing Systems*, 2004, pp. 329–336.

[4] N. Lawrence, "Probabilistic non-linear principal component analysis with Gaussian process latent variable models," *Journal of Machine Learning Research*, vol. 6, pp. 1783–1816, 2005.

[5] G. E. Hinton and S. T. Roweis, "Stochastic neighbor embedding," in *Advances in Neural Information Processing Systems*, 2002, pp. 833–840.

[6] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 2579–2605, p. 85, 2008.

[7] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[8] A. Melkumyan and F. Ramos, "Multi-kernel Gaussian processes," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2011, pp. 1408–1413.

[9] D. Duvenaud, J. R. Lloyd, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani, "Structure discovery in nonparametric regression through compositional kernel search," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 1166–1174.

[10] A. G. Wilson and R. P. Adams, "Gaussian process kernels for pattern discovery and extrapolation," in *Proceedings of the International Conference on Machine Learning*, 2013.

[11] R. Calandra, J. Peters, C. E. Rasmussen, and M. P. Deisenroth, "Manifold gaussian processes for regression," *arXiv preprint arXiv:1402.5876*, 2014.

[12] J. Wang, A. Hertzmann, and D. M. Blei, "Gaussian process dynamical models," in *Advances in Neural Information Processing Systems*, 2005, pp. 1441–1448.

[13] S. Hou, A. Galata, F. Caillette, N. Thacker, and P. Bromiley, "Real-time body tracking using a Gaussian process latent variable model," in *Proceedings of the 11th International Conference on Computer Vision*, 2007, pp. 1–8.

[14] M. A. Alvarez, D. Luengo, and N. D. Lawrence, "Linear latent force models using Gjaussian processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2693–2705, 2013.

[15] S. Bratieres, N. Quadrianto, S. Nowozin, and Z. Ghahramani, "Scalable Gaussian process structured prediction for grid factor graph applications," in *Proceedings of the 31st International Conference on Machine Learning*, 2014, pp. 334–342.

[16] E. Snelson, C. E. Rasmussen, and Z. Ghahramani, "Warped Gaussian processes," in *Advances in Neural Information Processing Systems*, 2004, pp. 337–344.

[17] A. McHutchon and C. E. Rasmussen, "Gaussian process training with input noise," in *Advances in Neural Information Processing Systems*, 2011, pp. 1341–1349.

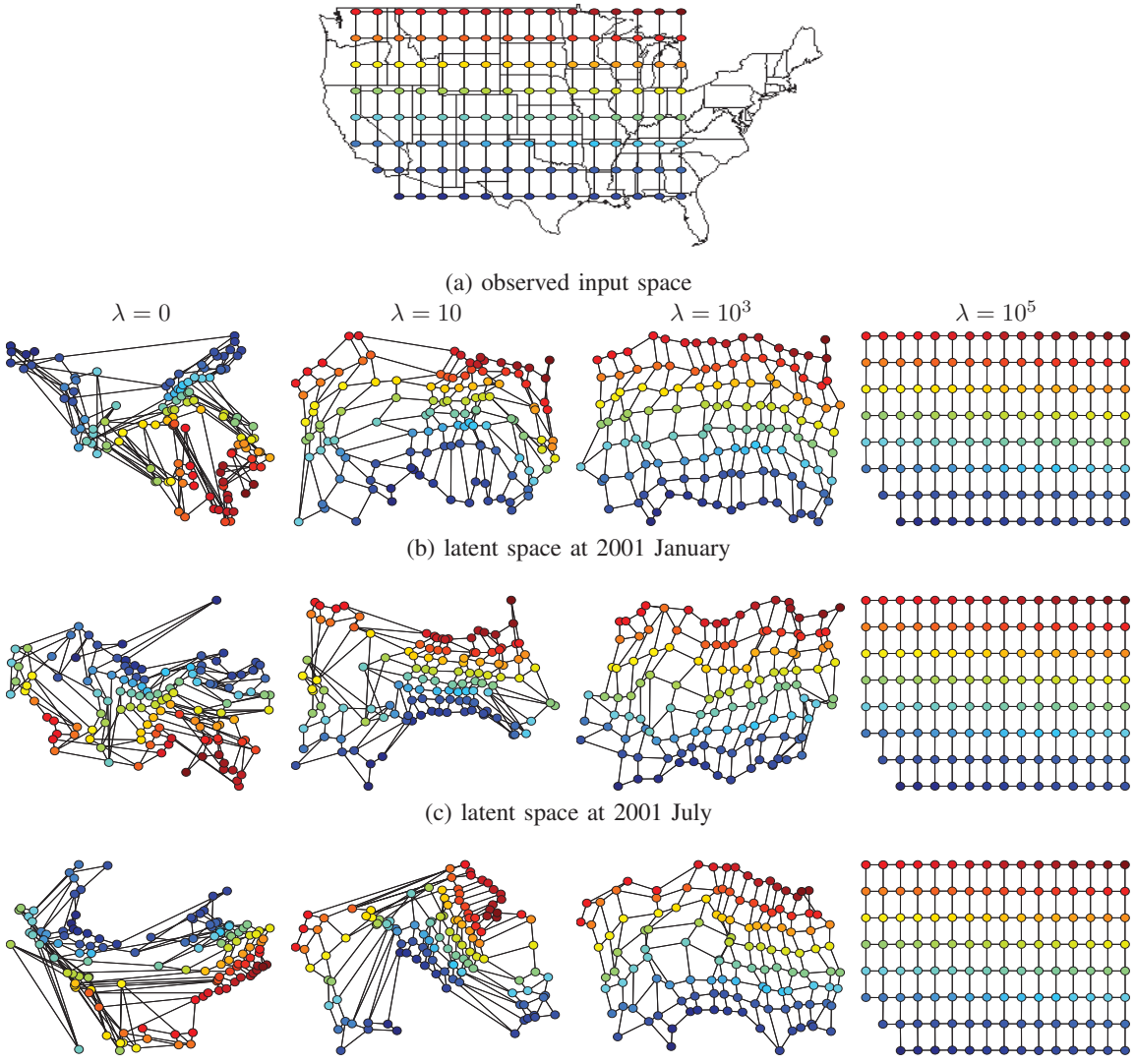
[18] R. Urtasun and T. Darrell, "Discriminative Gaussian process latent variable model for classification," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 927–934.

[19] Y. Yoshikawa, T. Iwata, and H. Sawada, "Non-linear regression for bag-of-words data via Gaussian process latent variable set model," in *Proceedings of th 28th AAAI Conference on Artificial Intelligence*, ser. AAAI, 2015.

[20] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[21] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[22] S. Akaho, "A kernel method for canonical correlation analysis," in *In Proceedings of the International Meeting of the Psychometric Society*, 2001.



(a) observed input space

(b) latent space at 2001 January

(c) latent space at 2001 July

(d) latent space at 2001 October

Fig. 1: The observed input space (a), and the latent space estimated by the proposed method (b,c,d) with the NA data set. The locations that are closely located at the input space are connected by edges. The color represents the locations in the input space.

[23] M. T. Bahadori, Q. R. Yu, and Y. Liu, "Fast multivariate spatio-temporal analysis via low rank tensor learning," in *Advances in Neural Information Processing Systems*, 2014, pp. 3491–3499.

[24] A. C. Lozano, H. Li, A. Niculescu-Mizil, Y. Liu, C. Perlich, J. Hosking, and N. Abe, "Spatial-temporal causal modeling for climate change attribution," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2009, pp. 587–596.

[25] E. Bonilla, K. M. Chai, and C. Williams, "Multi-task Gaussian process prediction," in *Advances in Neural Information Processing Systems*, 2008.

[26] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[27] M. Titsias and N. Lawrence, "Bayesian gaussian process latent variable model," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 844–851.

[28] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," in *Advances in Neural Information Processing Systems 18*, 2006, pp. 1257–1264.

[29] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," in *Uncertainty in Artificial Intelligence*, 2013.

Effect of a Label on Items for Their Popularity

Yuki Sonoda[1] and Daisuke Ikeda[2]

Department of Informatics, Kyushu University
744 Moto-oka, Fukuoka 819-0395, Japan.
[1]yuki.sonoda@inf.kyushu-u.ac.jp
[2]daisuke@inf.kyushu-u.ac.jp

Abstract— *We study popularity dissemination on items, such as products. Popularity is characterized by extreme imbalances since it is a typical rich-get-richer phenomenon. Existing researches focused on effect of consumers to dissemination, but not on effect of target items. Inspired by the result in [1], we hypothesize that some structure, called a* label*, on items increases imbalances of popularity. A category of products attached by a firm is a directly attached label while consumers can put a label on items. The goal of this paper is to confirm effect of a label on items to information dissemination. To this end, we have conducted multi-agent simulations about a virtual market in which firms produce items and consumers buy some of them. Comparing sales figures of items with labels and those without labels, we have confirmed that labels cause imbalances of popularity.*

Keywords: Multi-agent simulation, Consumer behavior, Power law distribution

1. Introduction

In this paper, we consider popularity of various things, such as smash hits of songs or movies and bursty words of blog entries, from the prospect of information dissemination. Counting target objects, we call *items*, in various fields, the common phenomenon, the power law distribution for popularity, emerges [2], [3].

In general, consumers have a lot of chances to see popular items and thus we see rich-get-richer phenomena [4], [5]. Moreover, consumers utilize information from many kinds of communication, such as word-of-mouth communication. As a result, such communications cause extreme imbalances and lead to the power law distribution. In this sense, we can think that popularity is caused by information dissemination [2]. From the view point of effective dissemination, existing researches focused on users, their network structures, and interactions of them [6], [7], [8], while the structure of target items has been ignored.

Generally speaking, however, an extremely popular trend does include many items. For example, there exists a trend of "premium" products in Japan, where a smash hit of a high-quality, high-price product yields this trend and then conversely this trend gathers similar high-quality, high-price products among different categories. We show another example of "Yuru-chara", which is

> a Japanese term for a category of mascot characters; usually created to promote a place or region, event, organization or business,

according to this page[1]. This concept became very popular after some popular characters, such as Hikonyan and Kumamon, became famous, and then the popularity of the concept created a huge number of new characters.

We can think that such a trend is a kind of categories of items. We call it a *label* of items and expect that a popular label can create an extremely large hit phenomenon. This idea was inspired by an earlier result of the authors [1], which is a research to predict potentially popular hash-tags, that is labels, of a micro-blog service. In this research, we experimentally showed that, for any label, the ratio of the number of items used with the label to the number of different items with the same label is constant over time. In other words, if the ratio is high for a label, we see many different items with the label as the label is used. Using this interesting property, we proposed a method to find latent popular labels. Although the proposed method works fine, we can not elucidate why such a phenomenon happens for labels and items.

As a first step to elucidate the mechanism of labels and items, we try to show that we see more smash hits if we can use labels. That is, the goal of this paper is to verify the following hypothesis: a label on items can increase imbalances of popularity of them. A typical label, for example, is a category of news articles while we can think anything we can attach to one or more items as a label, such as a catchy copy, an attribute, and a hash-tag.

To achieve this goal, we utilize multi-agent simulations, where we prepare consumer agents and firm ones, consumer agents can attach a label on items, both types of agents can recognize popular labels, and firm agents tends to create a new product with a popular label. Calculating sales figures in this virtual market, we compare sales of items with and without labels, and verify the effect of labels.

This paper is organized as follows: After reviewing related work in Section 2, we will explain our assumption about

[1]https://en.wikipedia.org/wiki/Yuru-chara

labels and the model for simulation in Section 3. Then we will show experimental results in Section 4. Finally, we conclude in Section 5.

2. Related Work

Popularity of items has been extensively studied in terms of information dissemination.

Some researches focused on consumers and proposed types of consumers, according to their behaviors. In [6], diffusion of innovations were studied and consumers were categorized, such as innovators and early adopters. Similary, the concept of connectors, mavens, and salesmen was proposed in [8].

There exists researches that studied dynamics of changes in popularity, basically using differential equations. For example, the Bass model was proposed in [7], considering innovation and imitation effects of consumers. Similarly, in [9], both direct communication and indirect one, such as the rummor effect, were considered and shown that the proposed equation well describes hit phenomena of movies.

Some other researches, such as [10], focused on the network structure of consumers.

However, these existing researches studied effects of communication among consumers, but not effects of structures among target items, w.r.t information dissemination. Of course, practitioners in marketing segments must know about effects and impacts of catchy slogans and copies, which can be seen as labels, since we see a lot of them on the media. However, there does not exist a quantitative research on effect of labels w.r.t information dissemination as far as the authors know.

3. Methods

In this section, we first explain, using examples, our assumption on labels for consumer behaviors, and then show our model of simulations.

3.1 Assumption of Effect on Labels

Consider that there exists a label attached with some items (see Fig. 1). Originally, the main target of a consumer is items and a label is just an attachment. For example, consider mascot characters, called "Yuru-chara." Although this term was coined in the early 2000's, the term was not so popular initially. In this case, the term "Yuru-chara" is just an attribute of mascot characters (the top figure of Fig. 1). However, after appearance of some attractive characters, such as Kumamon and Hikonyan, which are left- and right-hand side chracters in Fig. 1, the popularity of the term is risen sharply. Then we see this term many times on the media and many new characters were created. Now consumers first recognize the label on items (the bottom figure of Fig. 1). This is a *reverse phenomenon* since a popular label itself has its own popularity and increases popularity of items with the label.

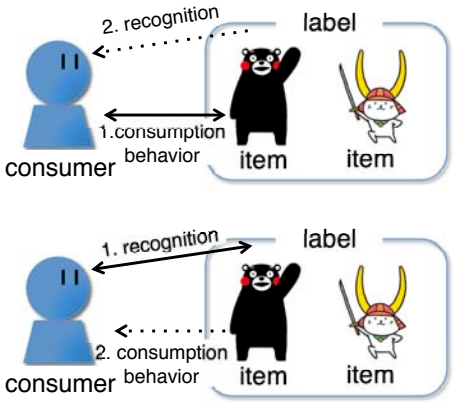


Fig. 1: Two processes of consumption behaviors of items and recognision of labels on them

One import property of labels is that the same label can be used in different fields of products. In other words, firms of other types of products can receive a label as a message. Then they can produce other types of products with the same label. An example of this phenomenon, we use "The Premium Malt's"[2]. From its name, outlook, and price, many consumers recognize "premium" as a label for it. The product was a blockbuster in Japan and made "premium" label as a popular label. In fact, we saw many high-quality, high-price products, such as premium pet food, premium canned coffee, and premium bananas, after this product. Therefore consumers have a lot of chances to see a popular label and so the popularity of the label could be drastically increasing. Then some consumers choose some products because they have the popular label.

We should note that hit products do not always create popular labels. In fact, a similar high-quality and blockbuster beer "Yebis"[3], which started to sell long before "The Premium Malt's" and a long-seller product. But it did not cause similar labels.

From the above observation, we hypothesize that a popular label on items encourages hit products and this causes imbalances of popularity among items.

3.2 Our Model

There exists two types of agents, consumer and firm agents. A consumer agent has perceived recognition rate for each label [11], choose items based on information, including the rate and rumors, and exchange information about items with other agents, where each consumer agent has randomly decided receiving and sending rates. A firm agent receives information about consumers' recognition of labels via market researches, recognize labels of product

[2] http://the-premium.jp/pc/index.html
[3] http://www.sapporobeer.jp/yebisu/

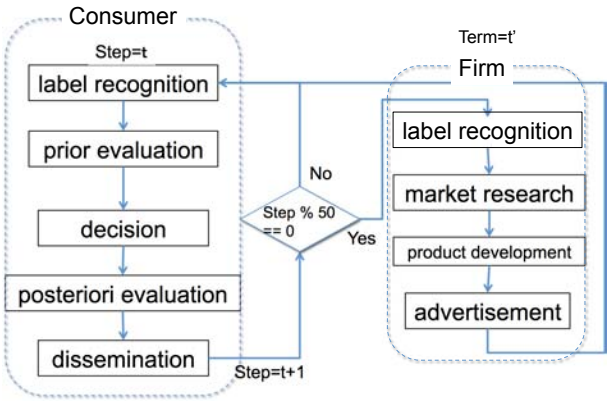


Fig. 2: The flow chart of our simulation

created by other firms, development a new product based on the sales of the last year.

The recognition of labels of other firms' product can be seen as information exchange among firms. In a simulation with labels, popularity of labels is important to develop new products, that is, a firm is likely to create a product with a popular label.

Fig. 2 shows the flow chart of consumer and firm agents. Based on AISAS model, which is a hierarchy model of advertisements and says consumer behavior changes in order attention, interest, search, action, and share [12], we decide one step of consumer agents as follows: they first recognize labels, then evaluate items before purchase, then choose an item, then evaluate purchased item, and finally disseminate information about the item. Similarly, we decide one step of firm agents as follows: they first recognize labels, then conduct market researches, then develop new products, and finally advertise them. In simulations without labels, both types of agent ignore tasks related to labels.

Every step of the firm agents is executed after 50 steps of the consumer agents because one step of firm agents takes longer time, compared to daily consumer behavior. For simplicity, one firm creates only one product and thus if a firm creates a new product, the current product of the firm is removed.

4. Results

After explaining environments for our experiments, we show results of our experiments, including preliminary ones for choosing some parameters, such as the number of trials.

4.1 Environments

Based on the model described in the previous section, we implemented the simulation program in Python and compiled with Cython. All experiments were executed on MacBook Pro (OS:Mac OS X 10.8.5, CPU:2.9GHz Intel Core i7, Memory:8GB 1600MHz DDR3).

Table 1: Parameters for our simulation

parameter	its value
# consumers	900
# firms	100
# types of labels	50

Table 1 shows the values for some parameters used in our simulation program. The initial values for the following parameters are real values randomly decided in $[0, 1]$: preference of consumers, sending and receiving moods of word-of-mouth, and advertisements of firms.

To evaluate results of simulations, we basically use histograms of sales among all firms since popularity of products are known to follow the power law distribution.

4.2 Preliminary Experiments

In this section, we show two results of preliminary experiments and decide the values for the following two parameters: the number of trials for stable results and the number of types of labels. To reduce the times required by preliminary experiments, we set the number of consumers to be 100 while it is 900 in the main experiments.

As described above, random numbers are used for some parameters. To obtain stable results, we should create histograms after several trials of our simulation program. First, to decide the number of trials, we compare two histograms of sales for 30 and 100 trials.

Fig. 3 shows two histograms: red one is from 30 trials and blue one 100. The horizontal axis is bins of sales, where

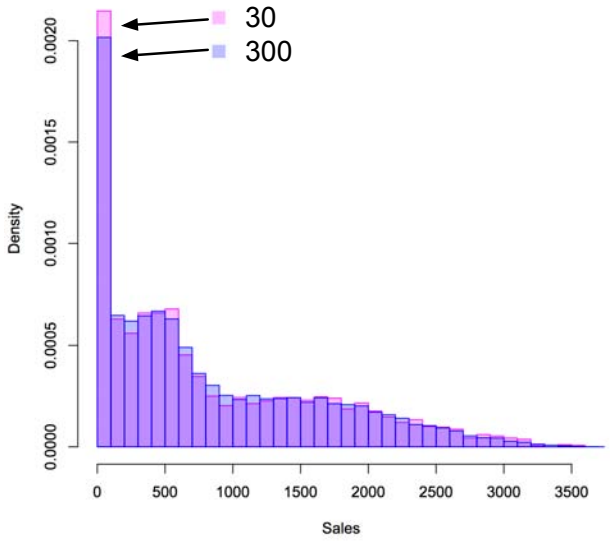


Fig. 3: Two histograms of sales for 30 and 100 trials, red one is from 30 trials and blue one 100.

the width of each bin is 100, and the vertical axis is the

probability of firms, which shows how many firms go into one bin.

From Fig. 3, we can see that 30 trials are enough stable, compared to 100 ones.

Since the goal of this paper is to show effect of labels for popularity of items, we compare simulations with and without labels. In this perspective, we need the large number of different labels. However, we expect that there exists two or more items which have the common label since the label is a category of items. In this sense, the number of different labels must be smaller than the number of items, which is equal to the number of firms in our setting.

Thus the second preliminary experiment is to decide the number of different labels for main experiments. In this experiments, we compare histograms of simulations in case that the number of different labels is 5, 20, or 50. Fig. 4 shows the three histograms, where the vertical axis shows

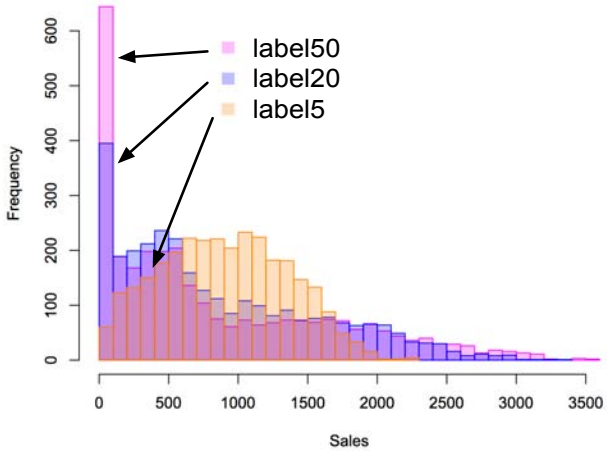


Fig. 4: Three hisgrams of sales in case that the number of types of labels is 5, 20, or 50.

the number of firms, that is the frequency, whose sales fall in the corresponding bin, the horizontal axis is bins of sales, where the width of bins is 100, and yellow (resp. blue and red) histograms are in case that the number of different labels is 5 (resp. 20 and 50).

From Fig. 4, we find that the shape of a histogram become skewed and the mode value become smaller as the number of different labels is increasing.

Table 2 shows statistics, such as averages, in case that the number of labels is 5, 20, or 50. As the number of different labels is decreasing, values of the average and median is increasing. In this sense, the fewer the number of different labels is, the more total sales are achieved. On the other hand, the skewness become much larger as the the number of different labels is increasing. In this sense, the number of labels has impact on imbalances of popularity. Considering our goal of this paper, we use 50 as the value for the number of different labels.

Table 2: Statistics in case that the number of labels is 5, 20, or 50.

# types of labels	average	median	skewness
5	906.2183	900	0.1032524
20	849.9577	631	0.8267709
50	834.0787	549	0.9537798

4.3 Main Results

As the main results, we show three types of graphs: one is dynamics of one trial and the other two are distributions of all trials.

Fig. 5 shows transitions of sales for each label in one trial of simulation, where the horizontal axis shows the total 1500, which equals to 50 steps times 30 trials, and the vertical axis the sales. In this graph, we see that the sales of the blue label

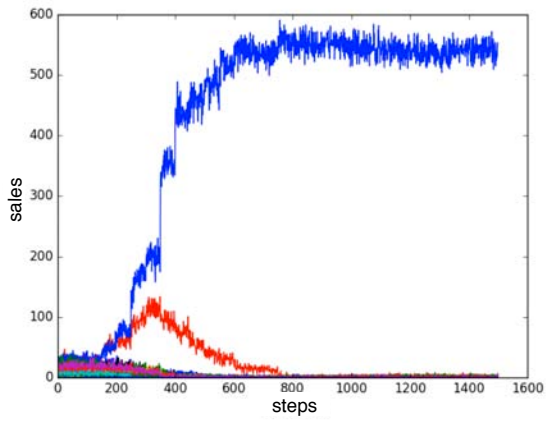


Fig. 5: Transitions of sales for each label in one trial of simulation, where each line shows the total sales of the products with the same label.

is drastically risen around 400 steps. This is a typical rich-get-richer phenomenon. We confirmed similar phenomena in all the other trials.

We created two types of graphs from 30 trials: one is rank-size plots (see Fig. 6) and the other histograms of sales (see Fig. 7).

Fig. 6 shows two rank-size plots, where the vertical axis shows sales for firms and the horizontal axis ranks of firms in decreasing order of sales. The left-hand (resp. right-hand) side graph is one created from 30 trials of simulations without (resp. with) labels.

We find that the amount of sales at the top rank in the right-hand side graph is about twice of that in the left-hand one, and the curve in the left-hand graph is smoothly declined, compared to that in the right-hand one.

Fig. 7 shows two histograms, where the horizontal axis shows sales bins, each of whose width is 300, and the vertical

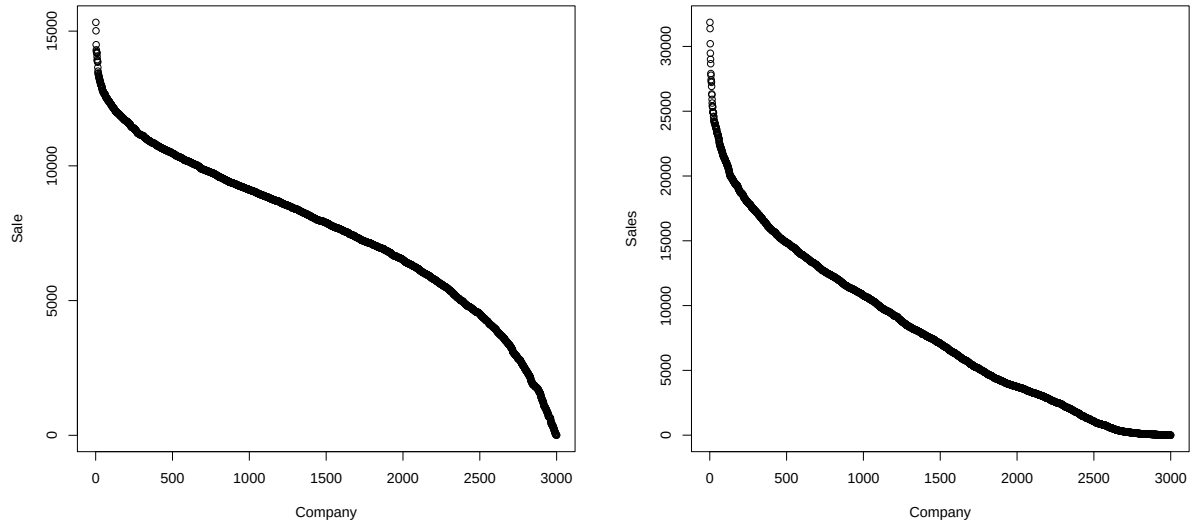


Fig. 6: Totals sales of 100 firms in 30 trials without labels (left-hand side) and those with labels (right-hand side) are plotted in decreasing order.

axis the frequency, that is, the number of firms whose sales figures is in some bin. The blue (resp. red) histogram is created from 30 trials of simulations without (resp. with) labels.

We find that the blue histogram is unimodal, where the mode is around the average value (see Table 3). In the red one, the mode is at the lowest bin of sales and there exists many firms achieving much higher sales, compared to top sales in the blue one.

From Fig. 7 and Fig. 6, we can conclude that labels cause imbalance of pupularity.

5. Conclusion

In this paper, we have considered impact of the label on items w.r.t popularity of them, conducting multi-agent simulations. We have compared two types simulations, that is, one without labels and the other with them. Although the histogram created from the simulation without labels is unimodal and the mode is at around the average, only introducing the structure into target items by labels causes imbalance of popularity of items. While existing researches about popularity of items focused on interactions and/or structures of consumers, imbalance of popularity can be achieved by the structure of items. As far as the authors knows, this is the first result which reveals that the structure of items is critical to popularity of items.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 15H02787.

References

[1] Y. Sonoda and D. Ikeda, "Predicting Latent Trends of Labels in the Social Media Using Infectious Capacity," *International Journal of Future Computer and Communication*, vol. 4, no. 6, pp. 374–380, 2015.

[2] M. E. J. Newman, "Power Laws, Pareto Distributions and Zipf's Law," *Contemporary Physics*, vol. 46, pp. 323–351, 2005.

[3] D. Sornette, *Rank-Ordering Statistics and Heavy Tails*. Springer, 2000, ch. 6 of Critical Phenomena in Natural Sciences.

[4] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, pp. 509–512, 1999.

[5] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press, 2010.

[6] E. M. Rogers, *Diffusion of Innovations*, 3rd ed. Free Press, 1982.

[7] F. M. Bass, "A New Product Growth for Model Consumer Durables," *Management Science*, vol. 15, no. 5, pp. 215–227, 1969.

[8] M. Gladwell, *The Tipping Point: How Little Things Can Make a Big Difference*. Little Brown, 2000.

[9] A. Ishii, H. Arakaki, N. Matsuda, S. Umemura, T. Urushidani, N. Yamagata, and N. Yoshida, "The 'Hit' Phenomenon: a Mathematical Model of Human Dynamics Interactions as a Stochastic Process," *New Journal of Physics*, vol. 14, 2012.

[10] J. J. Brown and P. H. Reingen, "Social Ties and Word-of-Mouth Referral Behavior," *Journal of Consumer Research*, vol. 14, no. 3, pp. 350–362, 1987.

[11] H. Yamamoto, S. Nishida, S. Morioka, and S. Yamakawa, "The Effect of "Perceived Recognition Rate" on Consumer Behavior," *Journal of Marketing Science*, vol. 19, no. 1, pp. 73–89, 2011, in Japanese.

[12] R. Akiyama and K. Sugiyama, *Holistic Communication*. Sendenkaigi, 2004, in Japanese.

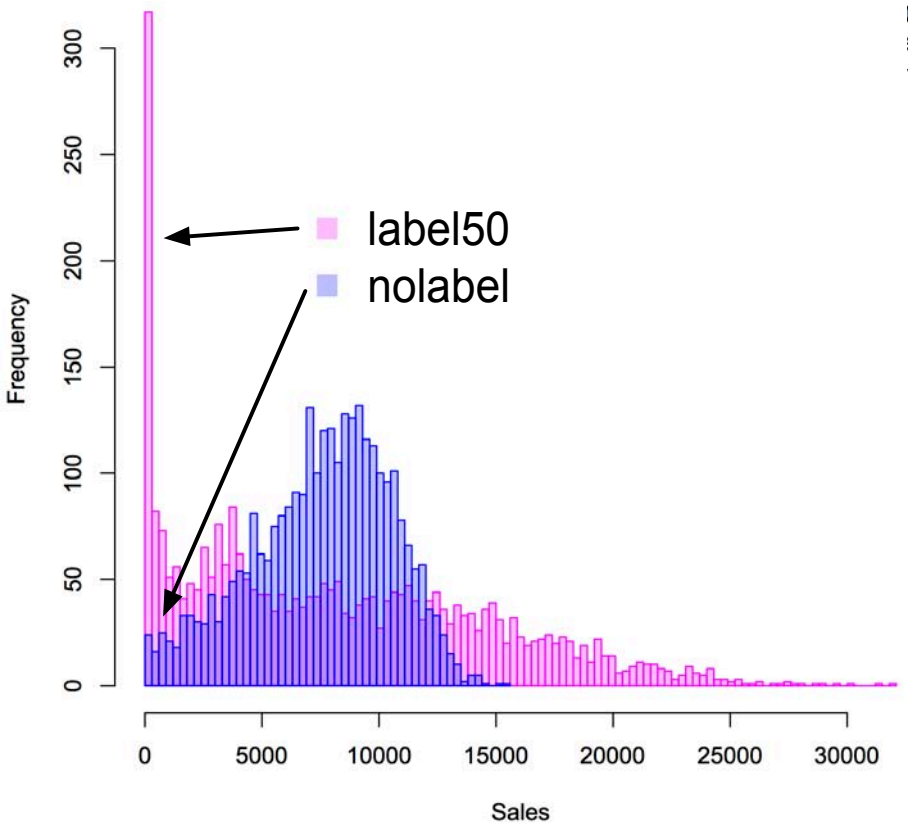


Fig. 7: Histogram of sales for simulations without labels (the blue histogram) and with labels (the red one)

Table 3: Statistics with or without labels.

	average	median	skewness
without labels	7540.16	7895	-0.4017727
with labels	8042.85	7056	0.6286133

Time Series Analysis on the Determinants of Environmental Costs Expenditure Using Text Mining Technique

Toshiyuki MAEDA*, Naoya KAWAKAMI*, Yoshimi CHUJO*, and Eunjee PARK†

* Faculty of Management Information, Hannan University, Japan

† Faculty of Economics, Kagawa University, Japan

***Abstract*—This study aims to read from environmental report which corporate management policy and strategy would promote the motivation of corporate behavior, especially the environment-related activities by listed companies. To do that, we analyze the relation between qualitative data and quantitative description using text mining technique. Our study aims to derive the result of more detailed analysis to confirm the change of positiveness factor by analyzing with text mining technique such as corresponding analysis, for environmental report in 2000-2011 in Japan. Time series analysis reveals that as CSR concept is widely disseminated, the focus of massage shifts from pollution aids to more comprehensive activities, which implicitly indicates that the environmental report is currently recognized as a useful tool to effectively communicate a firm's social activities.**

Keywords: Environmental report; Corporate Social Responsibility; Text mining; Corresponding analysis

I. Introduction

This study aims to read from environmental report which corporate management policy and strategy would promote the motivation of corporate behavior, especially the environment-related activities by listed companies. To do that, we analyze the relation between qualitative data and quantitative description using text mining technique.

These days, disclosure of social-related information by company has been expanding by dissemination of Corporate Social Responsibility (CSR). In other hand, the method of text mining in various researches has spread as main tool of information gathering. However, this text mining has hardly been applied in management study fields except marketing.

Shirata et al. analyzed the annual securities report qualitatively and descriptively, and tried to read the type of corporate behavior from text data. In general, many studies which have analyzed the annual securities report are interested in quantitative data, the financial information included. Her study used the text mining with several kinds of indices to extract keywords to characterize the bankrupt company and going concern [1]. The results from her analysis provide a new clue for management studies in Japan that they are able to analyze the dynamism of corporate behavior in a different viewpoint.

To retain and improve various society and environment-related activities is becoming a factor to gain an advantage of the corporate competitiveness. This attribute will be reflected in the information of company's disclosure [2]. For example, Kitora [3] analyzed the descriptive answer of CSR policy in "the CSR Corporate Survey 2006" of Toyo Keizai Shinpo Sya with the text mining.

In contrast, Murai et al. even analyzed the factor to define the positiveness. They regarded the amount that companies pay for the environment conservation cost as a proxy of the positiveness for environmental activities in Japanese companies. They extracted directly keywords which support the positiveness from the description of environmental report, and then analyzed with the environmental information [4]. To develop further, our study aims to derive the result of more detailed analysis to confirm the change of positiveness factor by analyzing with text mining technique for environmental report in 2000-2011 in Japan.

II. Environmental report

A. Definition of the environmental report

Ministry of the Environment in Japan defines as follows in the Environmental Reporting Guidelines (2012 Version) [5];

Environmental reporting allows enterprises to fulfill their accountability to society as businesses that use natural resources, to provide stakeholders with useful information that may affect their judgment, and to promote environmental communication with them.

These days, environmental report is becoming a basic tool in environmental communication between company and stakeholders including consumer. Many companies disclose "the sustainability report," or "the social and environmental report," or "the CSR report" except "the environmental report" which describes initiatives intended to fulfill CSR. We study to regard them as environmental report because any report used for environmental reporting, regardless of name, is regarded as "an environmental report" in the guidelines [5].

The main contents of the environmental report are always "environment issues" , but they also add economic and social aspects related the environment in the 2012 version. Our study

omits explanations for particular items because the subject is not all of them. The environmental accounting information of our study's subject is in "the Economic Contexts of environmentally focused management" in the new guideline.

B. Environmental accounting

Ministry of the Environment in Japan defines as follows in the Environmental Accounting Guidelines (2005 Version) [6];

Environmental accounting aims at achieving sustainable development, maintaining a favorable relationship with the community, and pursuing effective and efficient environmental conservation activities. These accounting procedures allow a company to identify the cost of environmental conservation during the normal course of business, identify benefit gained from such activities, and provide the best possible means of quantitative measurement (in monetary value or physical units) and support the communication of its results.

In other words, environmental accounting is the system to measure how much capital investment and what kind of the environmental conservation activities, and provide them to inter- and intra-company stakeholders. Even if the inter-company stakeholders can read the environmental accounting and will understand the situation of environmental conservation activities of the company. These days, many companies in Japan adapt this environmental accounting system.

The reasons why companies introduce the environmental accounting and disclose the information are as follows. First, companies need to grasp and evaluate cost vs benefit in aspects of environmental activities for themselves. These can be by using in monetary value or physical units for environmental conservation activities, environmental conservation benefit and economic benefit. Second, companies have an accountability widely for society in aspects of CSR, so companies disclose the information by environmental accounting as one of methods to accomplish the accountability. Third, the society with comprised various stakeholders asks the environment-related information. Consumers who are interested in the environmental issue would positively buy the environmentally focused products and services. Investors who are interested in the environmental issues would positively have one of information when they choice the invested company. Like these, companies and stakeholders would need to use the environmental accounting [7].

The contents of environmental accounting information in environmental report are comprised of "the Environmental Conservation Cost" , "the Environmental Conservation Benefit" , and "the Environmental Benefit Associated with Environmental Conservation Activities" . Our study target that companies pay "the Environmental Conservation Cost" of them. The environmental conservation cost is divided into the investment and costs, measured in monetary value, allocated for the prevention, reduction, and/or avoidance of environmental impact, removal of such impact, restoration following the occurrence of a disaster, and other activities [6].

III. Research methodology

A. Research target

In this study, we use environmental reports published in 2010, which are based on the period from April in 2009 to March in 2010. We especially focus on "top messages" in environmental reports as the targets of text mining analysis.

"Top messages" present swearness of CEO to consumers and stake-holders, and concentrated Those imply that contents of environmental reports

Among the companies which present environmental reports for 12 years on Web, we sorted companies which make groups more than five companies per one group.

We thus have ten groups such as home electric industry, precise industry, chemical industry, transportation industry, communication industry, control industry, other electric industry, construction industry, food industry, metal industry.

We collect the environmental report data from various Web sites such as corporation Web pages linked from METI (Ministry of Economy, Trade and Industry), Internet Archives, and so on.

B. Research method

We extract the top messages of the environmental reports mentioned above, and make groups from those reports with three high ranks and low ranks at the environmental cost ratio with the sales amounts.

We furthermore divide into three periods of each four-year time so that it is easy to capture the trend of changes, and then analyses with respect to the change of the top messages for 12 years.

The reason we analyze the ratio of environmental cost in the sales, is that analysis should be independent with company size, and should find the characteristics of the environmental cost ratio for each industry.

In addition, the reason for the split 12 years into three periods by four years for the analysis of time series is that the number of documents for each period is easily comparable because of the same amount of documents, and the number of groups is not so much for the analysis of the time series variation.

Extracted data from the top message are grouped and normalized as follows.

1) Cost ratio is it be classified in descending order and small order in three companies group of every industry. Each group is named as "upper group" and "lower group" respectively.
2) 12 years are divided into the period of 2000 - 2003, 2004 - 2007, and 2008 - 2011 by the year of report publication with the above groups. Each period is named as "former period", "middle period", and "later period" respectively.
3) Morphemes extracted from the top message are limited only to nouns (except proper noun, person's name, and place name) and adjectives.
4) The word usage frequency for each report is normalized with 200 words, and the top 50 words are extracted by

usage frequency, though some keywords, which do not make sense for analysis purpose such as "year", "billion" and so on, are omitted.

5) Above extracted Keywords are checked, and the keywords that are used in more than one group are picked up.

After above arrangement, the corresponding analysis are performed to confirm visually the relationship between the keywords and the frequency of respective group.

In addition, the analysis for multi-sector is performed as described above, and then groups are re-summarized based on the per-cost ratio. After that the correspondence analysis are performed in the same manner as above.

To enable to compare numerically with the environmental conservation costs of the sample companies, we calculate the "cost rate (%) = [environmental conservation costs] / [sales] * 100".

This allows us to compare how much the environmental conservation cost is spent by companies comparing to the percentage expenditure to sales.

C. Analysis method

First of all, sentences must be split into words to analyze with the text mining techniques. If words are separated by white space as English, it is quite easy. However, we do not write Japanese sentences leaving some space between words or grammatical units so that it is very difficult to separate the words correctly.

Therefore, we use a technique called morphological analysis in order to extract words accurately from the sentence in Japanese. The morpheme is the "the smallest semantic unit" in terminology used in linguistics [8]. Accuracy to split from the sentence to the morphemes (keywords) is achieved over 90 percent in the current state of the art.

D. Analysis environment

We introduce "RMecab", "R" with "Mecab" module. "R" is a programming language and its developing environment born in 1992, and that has graphics making functions and statistical techniques. In this study, "MeCab" is used for morphological analysis, because word or phrase sets are said to be better than letters to extract semantic information in Japanese.

IV. Results and Discussion

A. Comparison of EC ratio

Table I exhibits the average proportion of environmental costs expenditure to sales revenue (hereafter EC ratio) for each industry. The mean EC ratio across industries is 0.90%, varying from 0.34% (precision equipment) to 1.87% (construction). It is argued that over half of the sample firms expend over time for the environmental conservation by as much as 1% of the amount of sales.

B. Frequency of key words

Typically, the result of text mining is accompanied with the frequency of key words gauged by the morphological analysis. Focusing on those key words which appear many times in the text usually provides a practical measure with which to know the general characteristics describing the overall text. Hence, this study initially classifies the sample firms of each industry into 3 groups, which consist of top (HIGH), medium (MID) and bottom (LOW) 3 firms in terms of EC ratio, to investigate whether there is any relationship between environmental costs expenditure and the word choices. As is explained in the previous section, the sample period (2000-2011) is divided threefold: first (2000-2003), second (2004-2007), and third (2008-2011) period. Given the 3 firm groups and the 3 time periods, this study normalizes the morphemes as well as the key words extracted from the management message section included in the environmental report. Then, it compares the common characteristics of the words or morphemes across these groups.

C. Correspondence analysis of industry specific words

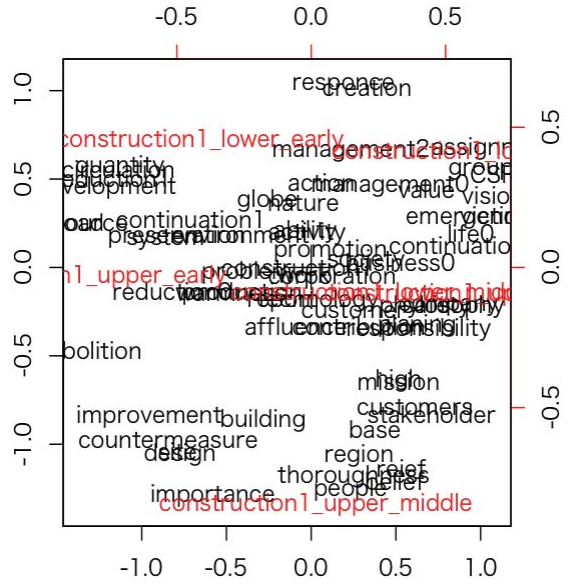


Figure 1: Correspondence analysis of Construction.

Figure 1 shows the correspondence analysis of construction industry. Construction is located in the left-hand side for the first period and in the right-hand side for the second and third period, indicating that the emphasis of the top message has significantly changed between the periods. The HIGH group appears for the first time near an array of words such as 'reduction of environmental load' or 'reduction of wastes', while the Great East Japan Earthquake and subsequent economic stagnation likely make this group more involved in the CSR activities. The LOW group also recognized issues regarding the pollution from early stages, and started actions

Type	Construction	Chemistry	Transport	Iron & Metal	Home Electric	Food
Proportion(percent)	1.87	1.56	1.49	0.81	0.70	0.64

Type	Control Machinery	Communication	Precision Machinery	Others	All
Proportion	0.53	0.49	0.34	0.52	0.90

Table I: Average proportion of environmental costs expenditure.

to resolve the issues by adopting more efficient environmental techniques, but due to the occurrence of the earthquake such activities come to be largely constrained. For transportation device, subsequent to the first time period, there arises a significant difference between the EC ratio groups. Especially, the HIGH group consistently stresses the importance of the 'product concepts' such as 'diesel engine' or 'truck', which results in the nontrivial distance from the LOW group.

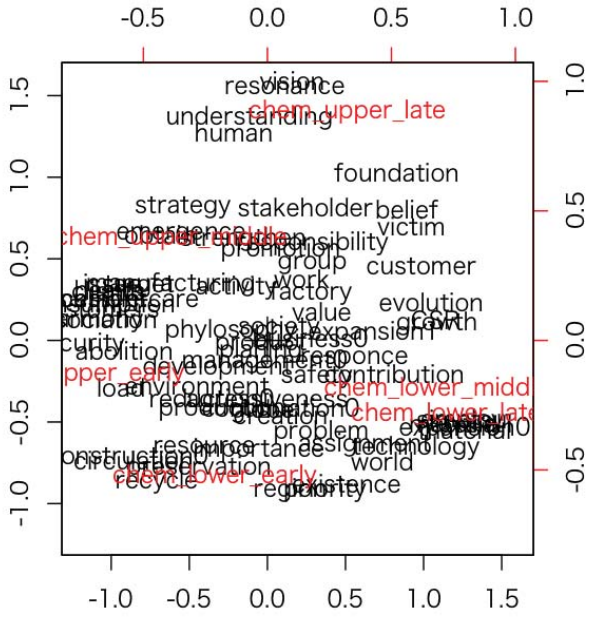


Figure 2: Correspondence analysis of chemical industry.

Figure 2 shows the correspondence analysis of chemical industry. That indicates a disperse deployment of characteristic words between time periods. From the second period, the HIGH group begins to depart from the remaining two groups. It is suggested from the words that this group tries to shift its focus toward a new direction following the occurrence of the great earthquake. Likewise, in the iron non-ferrous industry, the HIGH group initially promotes activities such as 'energy-saving' or 'recycling', whereas switches the target into 'CSR activities' from the second period. On the other hand, in the latter periods, the LOW group is surrounded by those words, the contexts of which are somewhat inconsistent. Consumer electronics is initially centered with how to resolve the environmental problems, but from the second period, there arises a significant difference between the HIGH and LOW group: the former heads toward 'innovation' and the latter toward 'CSR activities'.

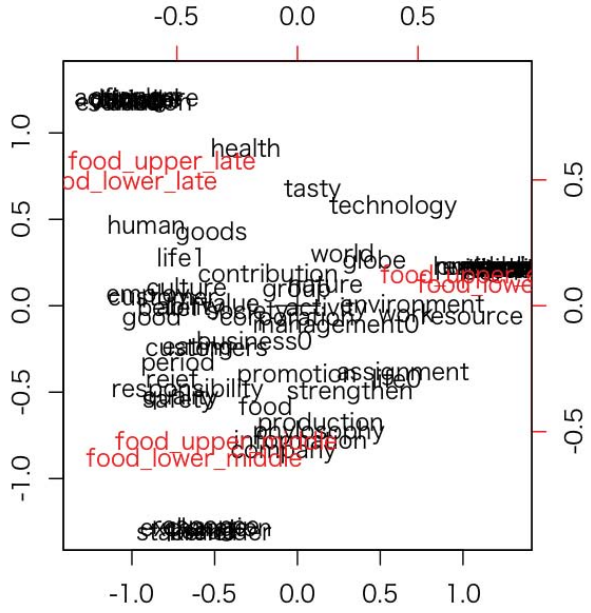


Figure 3: Correspondence analysis of food industry.

Figure 3 shows the correspondence analysis of food industry. Food would be a peculiar case in that there is no identifiable difference except for the inter-temporal one. Alternatively, whether to spend for the environment does not matter for food firms, and common targets such as 'responsibility' (second period) and 'consumers' health' (third period) are shared. Anyhow, the words seem to commonly represent the strategic characteristics of this industry.

Figure 4 shows the correspondence analysis of control equipment industry. As for controlled equipment, the words appearing in the third period considerably departs from those characterizing the beginning two periods. Currently the HIGH group mainly advocates innovation and the LOW group is stuck to customer services.

Figure 5 shows the correspondence analysis of communication industry. As for communication, similar to the construction, the distinction of words manifests itself between the first and subsequent time periods but there is little difference between the EC ratio groups. In the latter periods, the disaster recovery actions might become a common objectives to undertake for both HIGH and LOW groups.

Figure 6 shows the correspondence analysis of precision machine industry. As for precision mechanical equipment, the same trend can be observed in that there is a significant

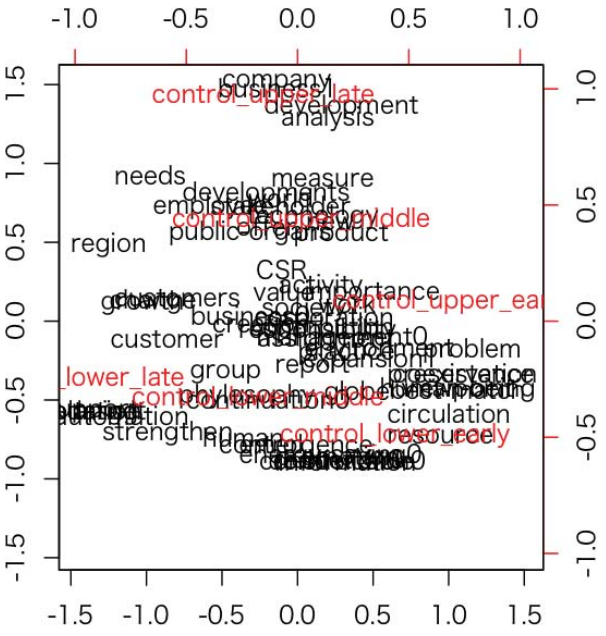


Figure 4: Correspondence analysis of control equipment industry.

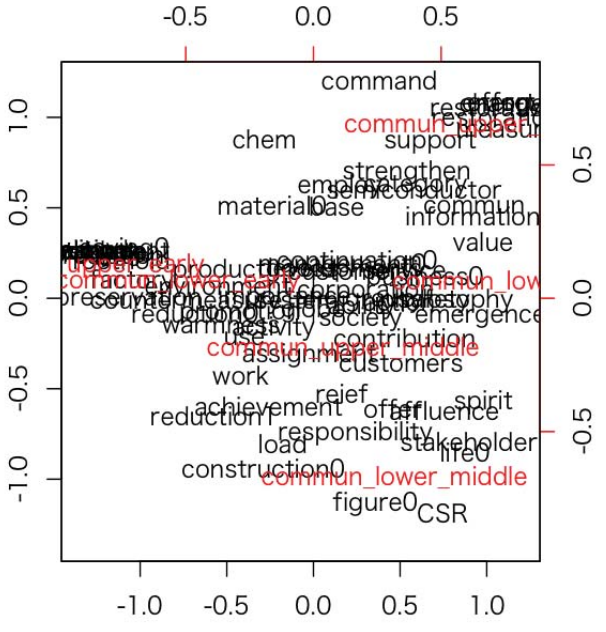


Figure 5: Correspondence analysis of communication industry.

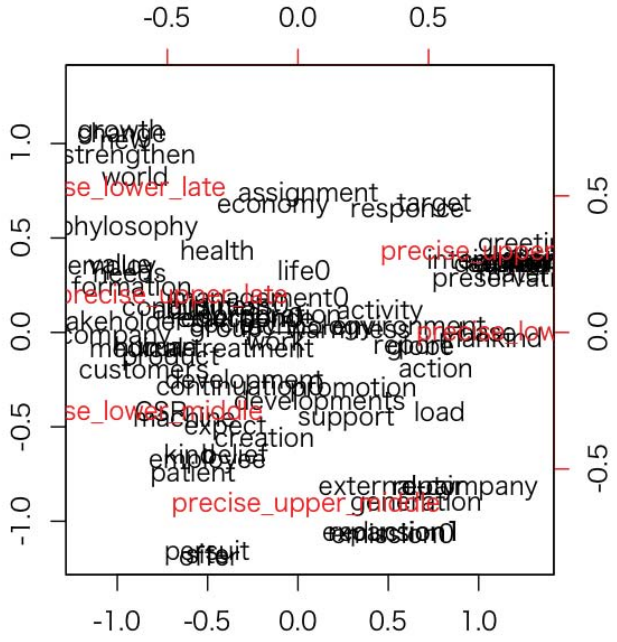


Figure 6: Correspondence analysis of precision machine industry.

divergence between the first and subsequent periods. Simultaneously, even in the latter period, while the HIGH group is located near many words addressing various kind of visions this group encompasses, the LOW group is surrounded by a few words, which indicates that this group is encountered with some difficulties in actively conducting environmental activities since the occurrence of the earthquake.

V. Concluding Remarks

This study applies the text mining to exploring what kind of key words exclusively characterize the top management message attached to the environmental report which Japanese firms have disclosed. Not only addressing the key words, it also visualize the relationship between the words and the firm group based on how much expenditure is devoted for environmental activities.

As Murai et al. suggest, the more a firm expends for environment, the more involved it would be in developing new products and commercializing such products as well as more interested in the CSR activities [4]. Oppositely, the less expenditure for environment means that the firm does not envisage any clear objectives to conduct the environmental conservation and CSR activities.

Time series analysis reveals that as CSR concept is widely disseminated, the focus of massage shifts from pollution aids to more comprehensive activities, which implicitly indicates that the environmental report is currently recognized as a useful tool to effectively communicate a firm's social activities. In addition, despite the variability of targets in developing new products across industries, Japanese firms become more

sensitive regarding the extent to which they should understand the needs of their surrounding society as well as they can receive the approval from the community. In this sense, this study gives some realities to CSR literature by linking the amount of environmental cost expenditure to the text with which management speaks to not only shareholders but also communities.

The limitations of the study should also be noted. The sample included in the analysis consists of only ten industries which are deemed to be substantially subject to environmental regulations, so the results derived above do not suffice to address a general conclusion about the motivations for Japanese firms to invest more in environmental activities. Further, this study deals with the environmental report available on line: those reports undisclosed via website are hardly acquired. A potential sample selection bias might work toward favoring those key words reported in the previous section. The future study would be directed to incorporate the missing samples and elaborate on the results presented in this study.

ACKNOWLEDGMENT

Part of this research was supported by JSPS KAKENHI Grant Number 15K03802. This research was also partially conducted under a sponsorship of Grant of Institute of Industrial and Economic Research, Hannan University. The authors greatly appreciates the supports.

REFERENCES

[1] Y. Shirata, H. Takeuchi, S. Ogino, and H. Watanabe, "Financial analysis using text mining technique : Empirical analysis of bankrupt companies (in japanese)," *Business Analysis Association Annual Report*, vol. 25, pp. 40–47, 2009.

[2] T. Toyosumi, *Strategic Environment Management*. Chuo Keizaisha, 2007.

[3] Y. Kitora, "Classification of companies based on the basic corporate social responsibility (csr) policy by using the text mining approach (in japanese)," *Journal of Socio-Informatics*, vol. 13, no. 1, pp. 17–29, 2009.

[4] T. Murai, Y. Chujo, E. Park, and T. Maeda, "An analysis of environmental reports using text mining methods [in japanese]," in *Abstracts of Annual Conference of Japan Society for Management Information 2011f*, 2011, pp. F3–2.

[5] G. o. J. Ministry of Environment, "Environmental reporting guidelines (fiscal year 2012 version)," http://www.env.go.jp/en/policy/economy/erg2012.pdf, 2012.

[6] ——, "Environmental accounting guidelines 2005," https://www.env.go.jp/en/policy/ssee/eag05.pdf, 2005.

[7] H. Ishibashi, *Environment and Consumers (in Japanese)*. Keiou Gijuku University Publishing, 2010.

[8] M. Ishida, *Introduction of Text Mining Using R (in Japanese)*. Morikita Shuppan, 2009.

Feature Selection for Diffuse Lung Disease using Exchange Markov Chain Monte-Carlo Method

Makoto KOIWAI[1], Nodoka IIDA[1], 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— *Diffuse lung disease (DLD) in high resolution computed tomography images show a lot of variations even in the same class, and this variations make difficulty in diagnosis. In this study, we treat a effective feature selection problem for this DLD pattern classification using machine learning approach. In order to obtain the best feature selection for classification, we should search whole combination of features, which requires exponential order calculation cost. Recently, Nagata et al. proposed an application of Exchange Markov Chain Monte Carlo (ExMCMC) method for this problem, and suggested that they reveals hidden feature structures for classification. Thus, we tried their method to select the effective feature combination for each DLD classification from 39 types of features, which are obtained from typical texture analysis method in the image processing. As the result, we obtained the effective feature combination candidates for each DLD classification problem.*

Keywords: Feature Selection, Medical Image, replica exchange MCMC

1. Introduction

In this research, we apply a feature selection method to classify the diffuse lung disease (DLD). The DLD is a kind of inflaming which is spread in the wide are of the lung. In the last stage of the DLD, the disease site lose function of lung and the patient becomes hard to recover, so that early detection of the DLD is desired[1][2]. To diagnose the DLD, the high resolution computed tomography (HRCT) images are regarded as the effective detection of DLD, because diagnosing physician can diagnose the spreading site of the candidate area from several directions. However, there exists large varieties of the DLD patterns on the HRCT image. Thus, the early detection influenced to the skill of physician who should diagnose whole lung volumes, which has over hundreds HRCT slice images. Moreover, the introducing the second opinion system increase the burden of the physician. So the computer aided diagnosis (CAD) system for DLD is desired to construct. In order to apply this requirement, many researchers introduce pattern classification technology into the DLD diagnosis[3][4][5][6].

In the field of machine learning, the pattern classification technology consists of both feature extraction part and classifier part[7]. In these decades, several classifiers, such like support vector machine (SVM), logistic regression, Bayes method and so on, are discussed. However the discussion about feature extraction and selection method looks not enough. So, we focus on the feature selection method to find a good feature set for classification for DLD patterns. In the previous works, Sugata *et al.* proposed a set of texture features for DLD pattern representation and apply it for classification with Naive Bayes method[1]. Wada and Hayakawa applied semi-supervised learning method for this feature representation[2][8]. In their research, they pointed out that using the full set of feature representation makes worse classification rather than that of some selected features. Wada *et al.* uses only about 4 selected feature for their experiment. The excess feature representation makes classification performance worse. This phenomenon is known as "curse of dimensionality". so that, the feature selection is important factor for classification performance essentially. The most rigid method for feature selection is using a Brute-force style method. Ichikawa *et al.* applied the exhaustive search of features for classifying attention deficit hyper-activity disorder (ADHD) from electroencephalogram (EEG)[9]. Unfortunately, the feature selection method is a kind of combinatorial problem, which requires exponential order calculation for exact solution search, which is sometimes called Brute-force search.

Therefore, the larger the number of whole feature set becomes, the more difficult the feature selection becomes. For this feature selection problem, Nagata *et al.* applied a Markov Chain Monte Carlo (MCMC) sampling with replica exchange system[10][11]. Hereinafter, we call that MCMC method as ExMCMC (replica Exchange MCMC) method. The ExMCMC is known as a powerful sampling method,

ISBN: 1-60132-444-8, CSREA Press ©

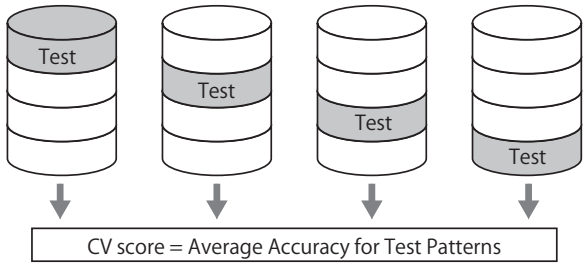


Fig. 1: Measuring method for generalization error using cross validation. Cross validation divides dataset into two parts called "training" and "test" samples. The training set is applied for constructing the classifier, and the test is for evaluation.

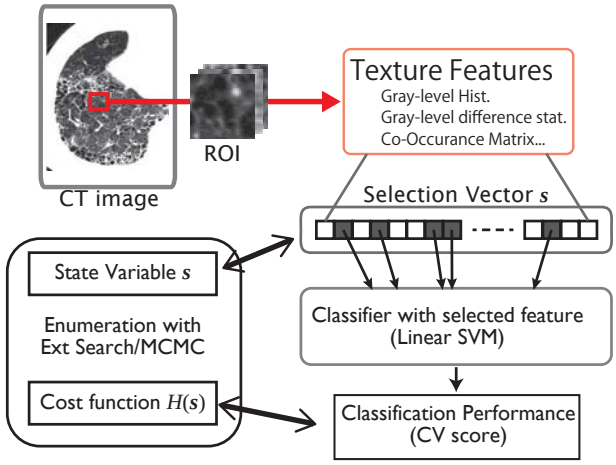


Fig. 2: Schematic diagram of ES-SVM method with MCMC. Regarding the CV score as a kind of cost function $H(\boldsymbol{s})$, our purpose is to find better solution candidates $\{\boldsymbol{s}\}$, which provides low $H(\boldsymbol{s})$

which runs the parallel MCMC in several relaxed conditions, and exchange states between parallelized run when several conditions are satisfied. Nagata *et al.* introduces the ExMCMC method for detecting information carrier neuron in the brain[10]. Thus, in this research, we apply ExMCMC based feature selection method for texture feature representation for DLD patterns, and evaluate the feature for classification.

2. Method

In this section, we explain feature selection method. The key idea comes from feature selection with exhaustive search with SVM [9][10], and ExMCMC method for sample enumeration.

2.1 Feature extraction with Exhaustive Search SVM method

The most sure method for feature selection is to apply exhaustive search which means "Brute-force" search. Here let us consider the following situation, that is, we have D features in the observation and want to find the most effective features combination for classifying. Before constructing a classifier, we must choose a feature set for the input of the classifier. When we choose a feature set with some method, we can evaluate the performance of the classifier with cross validation (CV) method. Cross validation is a kind of measure for generalization error[12]. Fig. 1 shows a concept for cross validation method. When we divide K sub-dataset, we can choose one subset as a 'test set' and the other as a 'training set'. We train the classifier with training set and evaluate the classification performance with test set. Thus, we can regard the classification performance for the novel input pattern. But it contains some arbitrary selection for the test set, so that we evaluate whole combination of the training set and test set. The cross validation score means the average of the whole combination of the classification performance. This is the concept of the K-fold CV method.

The most sure way for feature selection is to search the feature set, which shows the best CV score, from the whole combination of feature sets. Ichikawa *et al.* search the most effective for classifying ADHD disease from the 24 channel EEG[9]. Also, Nagata *et al.* introduce this feature selection method for picking up a neuron set in the brain to determine the information carrier for face recognition.

Unfortunately, the calculation cost for search whole combinations becomes $O(2^D)$, so that this method requires exponential order calculation cost. The more convenient way is to introduce some sparse prior, such like $L1$ prior, automatic relevant determination (ARD), and so on. However, Nagata pointed out the results of the L1 sparse logistic regression and the one with ARD showed different results. From the viewpoint of the accuracy of the feature set, we should not discard the exhaustive search method if we can calculate whole combination.

We introduce linear SVM as the classifier in this research. Thus, hereinafter, we call the exhaustive search method as "*ES-SVM*" method. The linear SVM is a simple linear classifier which divide input space into a hyper-plane, which is called decision boundary or discrimination plane, characterized normal vector $\boldsymbol{w}$ and interception b. The decision boundary is formulated as $y(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0$. When a novel input $\boldsymbol{x}_{\mathrm{novel}}$ is input to the system, the SVM evaluate whether the novel input is included in the target class or not with the value of $y(\boldsymbol{x}_{\mathrm{novel}})$. If $y(\boldsymbol{x}_{\mathrm{novel}}) > 0$, the novel input

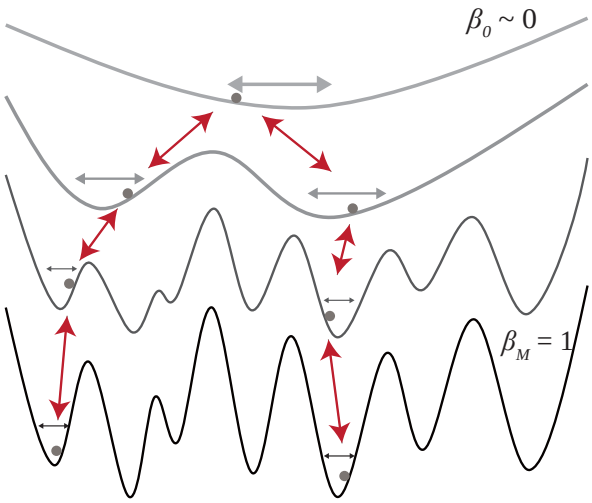


Fig. 3: Landscape of the the replica exchange MCMC method. The horizontal axis shows the state space, and the vertical shows the cost function $H_m(\boldsymbol{s})$. Preparing parallelized Monte Carlo sampling system with different inverse temperature systems, the state vectors move into another local minima easily via low β system.

is in the target class. Thus, the training of the SVM is to find an appropriate set of $\boldsymbol{w}, b$ with the principle of margin maximization[13].

2.2 Sample Enumeration with replica exchange Markov-chain Monte Carlo method

The problem for the exhaustive search for feature selection is the exponential calculation cost. We introduce a kind of Markov chain Monte-Carlo (MCMC) method for the enumeration of the feature set. Fig.2 shows the schematic diagram for the ES-SVM with MCMC method. In the figure, we have D feature candidates. Thus, we represent a feature set as state variable $\boldsymbol{s} = \{s_d\}_{d=1\ldots D}$ in which each element s_d has binary value $s_d \in \{0, 1\}$ that means the selected feature or not. Our purpose is to enumerate the $\boldsymbol{s}$, which minimize the CV score in the previous section. Thus, we introduce the cost function $H(\boldsymbol{s})$ as the CV score. Using optimization method for minimization $H(\boldsymbol{s})$, we can only find one vector set of $\boldsymbol{s}$. Our purpose is to find some candidates set of $\boldsymbol{s}$, so that, enumeration method is better rather than the optimization. In this research we adopt a Markov chain Monte Carlo (MCMC) sampling method for enumeration. The procedure for the MCMC is summarized as following:

1) Select one site $s_i^{(t)}$ in the state vector $\boldsymbol{s}^{(t)}$ where t means the time index.
2) Prepare a candidate vector $\boldsymbol{s}^*$ in which only $s_i^{(t)}$ is inverted from the vector $\boldsymbol{s}^{(t)}$.
3) Calculate the costs $H(\boldsymbol{s}^{(t)})$ and $H(\boldsymbol{s}^*)$ and evaluate the probability

$$r = \min\left(1, \frac{\exp(-H(\boldsymbol{s^*}))}{\exp(-H(\boldsymbol{s^{(t)}}))}\right). \quad (1)$$

4) Generate an unit random value $u \in [0, 1]$, and compare u with the r. If $u < r$ then accept the state $\boldsymbol{s}^*$ as a new state $\boldsymbol{s}^{(t+1)}$, and the other case the state is hold as $\boldsymbol{s}^{(t+1)} = \boldsymbol{s}^{(t)}$
5) Goto the 1st step while t satisfies the iteration limit

This method is known as Metropolis-Hasting (MH) method[14], and hereinafter we call this successive procedures as Monte Carlo step (MCS). Using the MCMC method, we can obtain samples $\{\boldsymbol{s}^{(t)}\}$ which obeys the probability $p(\boldsymbol{s}) \propto \exp(-H(\boldsymbol{s}))$.

The MH method is a strong method for enumerating, however, it requires long calculation time to sample from wide spreading multiple peak distribution. For transition from a peak to another, there exists low probability region in any transition paths. The driving force of the MCMC depends on the odds ratio of the pre- and post- state in eq.(1). So that, too much low probability region prohibits desirable transition.

The replica exchange MCMC method is to overcome the transition problem[11][10]. We introduce temperature parameter $T > 0$ and its inverse $\beta = 1/T$. Considering the probability with inverse temperature β of probability, we can re-define the probability with weight by inverse temperature $p(\boldsymbol{s}) \propto \exp(-\beta H(\boldsymbol{s}))$. The temperature $\beta = 1$ means our original cost function. When β becomes small, the efficacy of the cost function $H(\boldsymbol{s})$ also becomes small. So the landscape of the weighted cost function $\beta H(\boldsymbol{s})$ becomes smooth. Fig.3 shows the concept of the replica exchange method. We prepare L parallel replicated MCMC system, and we run each MCMC with different temperature T_l. After several MCS, we exchange several replica states. As the result, we can obtain sample from wide spreading multiple peak distribution via low temperature Markov-chain transitions. The procedure for the ExMCMC is summarized as following:

1) Prepare M replicated systems, and assign appropriate inverse temperature $0 < \beta_0 < \beta_1 < \cdots < \beta_{M-1} = 1$. Denoting each system status variable as $\boldsymbol{s}_m$ where m means the index of system.
2) Carrying out several MCSs under the probability of $p(\boldsymbol{s}_m) \propto \exp(-\beta_m H(\boldsymbol{s}_m))$ for mth system. Now, we describe the exchange timing as τ.
3) Select one temperature site denoted as j.

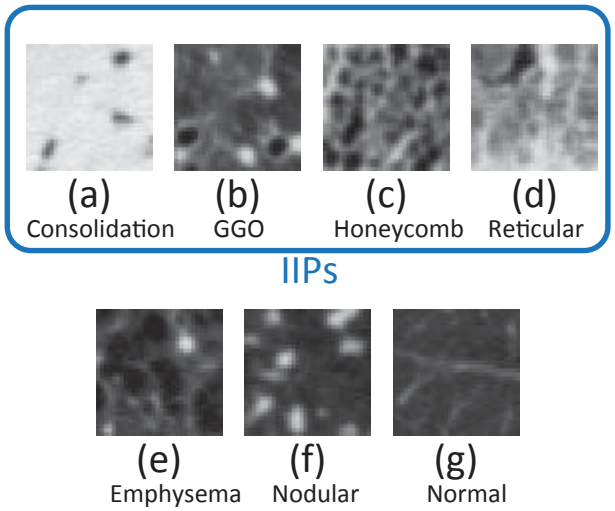


Fig. 4: Typical ROI samples for each class. Classes from (a) to (d) are involved in the idiopathic interstitial pneumonias (IIPs). Classes (e) and (f) is another disease. The class (g) means the normal class.

4) Calculate transition probability $W(\boldsymbol{s}^{(\tau)}{}_j, \boldsymbol{s}^{(\tau)}{}_{j+1})$ as

$$W(\boldsymbol{s}^{(\tau)}{}_j, \boldsymbol{s}^{(\tau)}{}_{j+1}) = \min(1, \exp(-\Delta)) \quad (2)$$

$$\Delta = (\beta_j - \beta_{j+1})(H(\boldsymbol{s}^{(\tau)}{}_j) - H(\boldsymbol{s}^{(\tau)}{}_{j+1})) \quad (3)$$

5) Generate a unit random variable $u' \in [0, 1]$, and compare it with $W(\boldsymbol{s}_j, \boldsymbol{s}_{j+1})$.
6) If $u' < W(\boldsymbol{s}^{(\tau)}{}_j, \boldsymbol{s}^{(\tau)}{}_{j+1})$, exchange the states $\boldsymbol{s}^{(\tau)}{}_j$ and $\boldsymbol{s}^{(\tau)}{}_{j+1}$.
7) Goto 2 for several MCSs.

So that, this parallel MCMC mechanism work as a outer loop of the each MH method. Applying the replica exchange MCMC method, escape from the the local minima is just easier rather than that of the single MCMC method.

3. Experiments

3.1 Materials

In this research, we prepare 360 labeled images. Each class has following number of images: 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: Each images are obtained from Toshiba "Aquilion 16" 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.4 shows segmented images of typical examples of each disease in HRCT image. The CON and 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 RET pattern which sometimes includes GGO patterns is also appeared in the NSIP. The HCM pattern has more rough mesh structure rather than that of the crazy-paving, and it appeared in the idiopathic pulmonary fibrosis (IPF) or the usual interstitial pneumonia (UIP).

3.2 Texture features from Region of Interest

We introduce several texture representations proposed by Sugata *et al.* for features[15][1]. 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[1]. From each of the gray-level histogram, gray-level difference statistics, Fourier power spectrum for the radial direction and for the angle, we extract mean, contrast, variance, skewness, kurtosis, energy and entropy. From the co-occurrence matrix, we extract energy, contrast, correlation, variance, entropy. From the run length matrix, we extract short/long run emphasizes, gray level no-uniformity, run length no-uniformity, and run percentage.

3.3 Configuration of replica exchange Markov Chain Monte Carlo method

We prepare $M = 7$ temperature replica systems, and iterate $T_{\max} = 20,000$ times. We use '*libsvm*' as the linear SVM implementation. We use the default parameters for the SVM. For the CV method, we apply 10-fold CV score as the cost function $H(\boldsymbol{s})$. The number of target class is 7, so that we adopt 'one-versus-rest' (OVR) classification method. The OVR method construct the class specific classier, so that, the one classifier identify the input is belongs to the class or not.

4. Results

Fig.5 shows the result of the density of $H(\boldsymbol{s})$ with ExMCMC method. Each figure shows the density histogram, and the horizontal axis shows the CV score, and the vertical one shows the density. The solid bar shows the histogram and the red line shows the estimated density with Gaussian kernel method. The top row shows the results for CON, GGO, HCM, RET classes, and the bottom shows the ones for the EMP, NOD, and NOR classes. The left limit of each

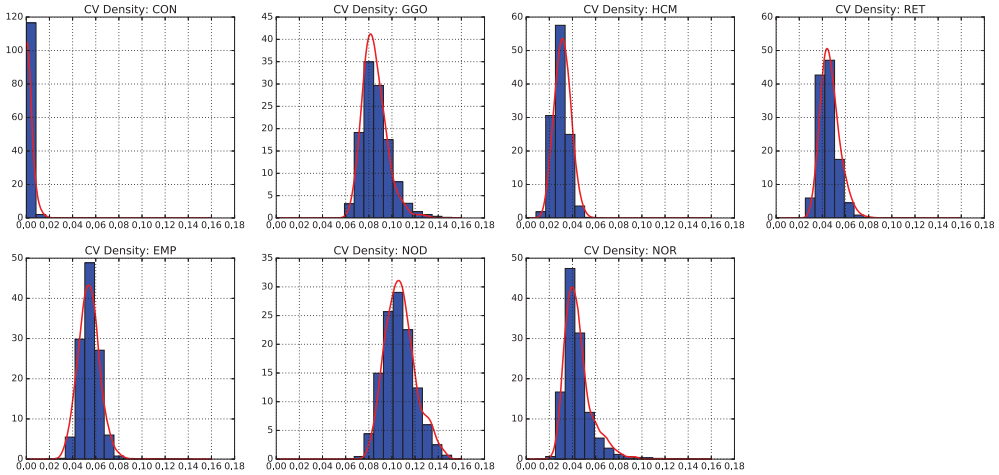


Fig. 5: Density of target cost function $H(\boldsymbol{s})$ for each class. The horizontal axis shows the CV error, and the vertical shows the density. In each figure, the solid bars show the density histogram, and the red curve shows the estimated density with a Gaussian kernel.

Fig. 6: Selected features for each class with top-5 CV scores. The left column shows selected features for CON, HCM, EMP, and NOR classes. The right shows for GGO, RET, and NOD classes. The black boxes show the locations of the selected features in each class.

histogram shows the best CV score for the class. We can see the CON class is easy to classify since it has a lot of $CV = 0$ state. Moreover, the CON class is insensitive to the feature selection because it has a lot of state in the $CV = 0$ mode. On the contrary, we can see the construction of the other classifier is not so easier than the CON class. The minimum CV state has not so very few state, and we confirm the performance is very sensitive to the feature selection. Especially, we can also see the both GGO and NOD classes are hard classify since the left limit value of the histogram has the just larger than the other histograms.

Fig.6 shows the selected features for top-5 CV scores.

The left column shows the result for CON, HCM, EMP, and NOR classes. The right one shows for GGO, RET, and NOD classes. Each horizontal axis shows the feature indices. Features from 0 to 5 come from co-occurrence matrix, from 6 to 10 come from run-length matrix, from 11 to 17 come from gray-level histograms, from 18 to 24 come from gray-difference statistics, and from 25 to 38 come from Fourier power spectrum. For CON class, the almost all features except coming from run-length matrix looks effective features. For GGO class, the gray-difference statistics, and the Fourier power spectrum in the angle direction looks important. For HCM class, gray-level histogram, gray-difference statistics, and Fourier power spectrum in the angle direction are important. For RET class, gray-level histograms, gray-difference statistics, and Fourier power spectrum in the radial direction are important. For EMP class, the co-occurrence matrix, and Fourier power spectrums might be important. For NOD and NOR classes, the co-occurrence matrix, and gray-level histogram might be important.

5. Conclusion & Discussion

In this research, we propose a feature selection method in the manner of Nagata's method. The original idea of this feature selection method comes from exhaustive search, however the calculation complexity of the feature selection problem is $O(2^D)$ where D means the number of total features. For small D, we can search the best combination with exhaustive search, but it becomes hard when D becomes large. So, we introduce replica exchange MCMC method for enumeration. Applying the MCMC method, we can obtain the density curve as well as the quasi-optimal solution. We can see the how many solutions around the quasi-optimal solution, so that, we can guess the difficulties of the classification problem.

In the future work, we should compare the result with the feature selection using some classification methods with sparse prior. Introducing a sparse prior is a powerful method, however, sometimes the solutions comes from different sparse priors show different feature selection[10]. So that, we should take it carefully. In such case, the result for the ExMCMC method might be a good indicator.

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) 16K00328, and Innovative Areas 16H01452, MEXT, Japan.

References

[1] 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/

[2] M. Wada, H. Shouno, and S. Kido, "An idiopathic interstitial pneumonia classification for ct image by use of a semi-supervised learning," in *Intl. Forum on Medical Imaging in Asia (IFMIA)*, November 2012, pp. P1–34.

[3] M. J. Gangeh, L. Sorensen, S. B. Shaker, M. S. Kamel, M. de Bruijne, and M. Loog, "A texton-based approach for the classification of lung parenchyma in ct images," in *MICCAI*, ser. LNCS 6363, no. 3. Springer-Verlag Berlin Heidelberg, 2010, pp. 595–602.

[4] 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.

[5] H. Shouno and M. Okada, "Bayesian Image Restoration for Medical Images Using Radon Transform," *Journal of the Physical Society of Japan*, vol. 79, p. 074004, 2010. [Online]. Available: http://jpsj.ipap.jp/link?JPSJ/79/074004/

[6] H. Shouno and S. Kido, "Semi-supervised based learning for Idiopathic Interstitial Pneumonia on High Resolution CT images," in *In Proc. PDPTA*, Jul. 2015, pp. 270–275.

[7] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd edition)*, 2nd ed. Wiley-Interscience, 2000.

[8] Y. Hayakawa, H. Shouno, and S. Kido, "Classification of Idiopathic Interstitial Pneumonias using Transductive Support Vector Machine (in Japaese)," IEICE Tech. Rep. (MI), Tech. Rep. 271, Oct 2012. [Online]. Available: http://ci.nii.ac.jp/naid/110009636793/

[9] H. Ichikawa, J. Kitazono, K. Nagata, A. Manda, K. Shimamura, R. Sakuta, M. Okada, M. Yamaguchi, S. Kanazawa, and R. Kakigi, "Novel method to classify hemodynamics response obtained using multi-channel fNIRS measurements into two groups: Exploring the combinations of channels," *Frontiers in Human Neuroscience*, vol. 8, p. 480, 2014.

[10] K. Nagata, J. Kitazono, S.-i. Nakajima, S. Eifuku, R. Tamura, and M. Okada, "An exhaustive search and stability of sparse estimation for feature selection problem," *IPSJ Transactions on Mathematical Modeling and Its Applications*, vol. 8, no. 2, pp. 23–30, 2015.

[11] H. Koji and N. Koji, "Exchange monte carlo method and application to spin glass simulations," *Journal of the Physical Society of Japan*, vol. 65, no. 6, pp. 1604–1608, 1996. [Online]. Available: http://dx.doi.org/10.1143/JPSJ.65.1604

[12] M. Stone, "Cross-validation: A review." *Math.Operations.Stat.Ser.Stat*, vol. 9, no. 1, pp. 127–139, 1978.

[13] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 1995.

[14] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[15] 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.

Architecture Design of Deep Convolutional Neural Network for Diffuse Lung Disease Using Representation Separation Information

Satoshi Suzuki[1], **Nodoka Iida**[1], **Hayaru Shouno**[1], **and Shoji Kido**[2]
[1] Graduate School of Informatics and Engineering, University of Electro-Communications,
Chofugaoka 1-5-1, Chofu, 182-8585, JAPAN
[2] Applied Medical Engineering Science, Graduate School of Medicine, Yamaguchi University
Tokiwadai 2-16-1, Ube, 755-8611, JAPAN

Abstract— *In this work, we propose a new architecture design of Deep Convolutional Neural Network (DCNN) with representation separation information of intermediate layers. The DCNN is one of the multi-layer neural network models. In recent years, the DCNN is attracting attention by its state-of-the-art performance in the image and speech recognition tasks. For example, Krizhevsky et al. showed the state-of-the-art performance in the large scale image recognition in 2012 [5]. However, the design for the architecture of the DCNN has not been discussed much since we have not found effective guideline to construct. We try to modify the architecture of the DCNN for the Diffuse Lung Disease (DLD) image classification task [9], [8], and confirm that the modified DCNN shows better performance than that of the original one.*

1. Introduction

In recent years, the performance of the image classification task has dramatically improved by Deep Convolutional Neural Network (DCNN) models [3]. Most notable work, which is known as the AlexNet that was proposed by Krizhevsky *et al.*, shows the highest performance in the image recognition task contest called the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) 2012 [5]. In the contest, the AlexNet shows far and away the best score, which is 16.4% error rate, compared to the 2nd place, which is 26.1%. The DCNN has the almost same structure to the Neocognitron proposed by Fukushima [4], [6], which can automatically extract feature expression from input data. The DCNN is now going to become a *de facto standard* classification tool in the image classification field, and the related research has been increased [3]. Many of the learning algorithms for the DCNN have been proposed various method since Fukushima proposed the basic structure [7], however, in recent years, the error back propagation (BP) method is typically used [6]. On the other hand, there is few design guidelines with respect to the architecture of DCNNs. So that, if we obtain a good clue for the architecture design, we can reduce the number of trial-and-error times to obtain a good DCNN for the specific purpose. For example, Zeiler *et al.* focused on the weight shapes, which is described as convolution kernel. They reduced the useless kernel, and obtain the improved performance [10].

In this work, we focus on a within-class variance of support vector machine (SVM) histogram representation which is proposed in our previous work [8] for each layer. In the previous work, we found the shape of the SVM histogram becomes narrower throughout the DCNN pattern transformation. Hence, we try to apply a quantity for representation of the shape of SVM histogram as a clue of the DCNN architecture design. In this paper, we apply the idea to the classification problem of diffuse lung disease (DLD) patterns, which typically appear with idiopathic interstitial pneumonias (IIPs). Fig.1 shows an example of the CT images of the DLD. The IIP spreads in the lung and is hard to cure at the last stage. Thus, early detection and classification are desired. For classifying DLD patterns, a high-resolution computed tomography (HRCT) image is considered to be effective since we can observe any cross section of the lung. Unfortunately, the IIPs site is difficult to diagnose, since the DLD patterns on HRCT image show a lot of variations in terms of texture. So we apply DCNN for such complex texture classification. However the obtaining cost of such labeled data is expensive in the medical imaging since it requires physicians' decision for proper disease labels. It is known that it is difficult to directly apply to large-scale DCNN like AlexNet [8].

2. Method

2.1 Deep Convolutional Neural Network

Deep Convolutional Neural Network (DCNN) has the similar structure to Neocognitron, which has a characteristic connection weight structure described as a convolution operation [4]. In this work, we fixed the training method for the DCNN as the BP, which is used as the standard training method in these decades [6]. Generally, the DCNN for the visual task takes 2-dimensional image input, and the input is transformed to the representation throughout the layers. The output layer of the DCNN provides class probability for the input image. Fig.2 shows a CaffeNet architecture used in previous work. This CaffeNet is used as the baseline DCNN. This CaffeNet has a similar architecture to the AlexNet [5].

ISBN: 1-60132-444-8, CSREA Press ©

Fig. 1: An example of the CT images of a patient diagnosed with Diffuse Lung Disease (DLD). The right lung portion on the left side, it can be seen that spreads out the white lesions in the red border.

The DCNN consists of following four types of layers, that is, "convolution", "rectified linear unit (ReLU)", "normalized", and "Polling" layers. The Convolution layer extracts a feature representation from the previous layer with a set of learned filters. The ReLU layer modifies the output of the convolution layer. The response of the ReLU layer is described as a rectified linear function ($relu(x) = \max(x, 0)$). The normalized layer is used as the optional layer, which is for reducing the contrast variance. The pooling layer applies max-pooling operation, which is used for reducing the effect of the local pattern deformation.

2.2 Layer Response Representation with SVM Histogram

This section explains about the SVM histogram for layer representation in the DCNN [8]. In the previous works, we introduced linear SVM for each layer in order to observe how representations in the DCNN develops throughout layer transformations. Let us consider about the linear SVM which is for finding a descision boundary. The decision boundary is decribed as $y(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}) + b = 0$, where $\boldsymbol{\phi}(\boldsymbol{x})$ denotes the layer representation for the input pattern $\boldsymbol{x}$. Here we introduce t as the teacher signal, which is described as $t_n \in \{1, -1\}$ in the case of two-class classification problem where n means the index for the test class pattern. In the feature space, the decision boundary of the linear SVM is obtained by maximizing the margin, $1/||\boldsymbol{w}||\ \min_n[t_n(w^T\boldsymbol{\phi}(\boldsymbol{x_n})+b)]$ [1].

In our previous work, we introduced a distance from the decision boundary for each layer representation $\phi(\boldsymbol{x})$ as a measure of discriminability, which is described as $y(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) + b$ where $\boldsymbol{w}$ and b are optimized by the linear SVM. Then, we can obtain a test class projection $\{y(\boldsymbol{x}_n)\}$. We analyze the projection $\{y(\boldsymbol{x}_n)\}$ as a histogram in the previous work[8]. Hereafter, we call it as "*SVM histogram*" for layer representation. Using the SVM histogram for each layer, we can visualize the distribution of each class data for intermediate layer of the DCNN. Here, Fig.3 shows the overview of the SVM histogram.

According to the previous work, the SVM histogram becomes narrower through the hierarchical development of representation in the DCNN layers. As the result, we concluded the narrower representation of each class plays important role for the class discriminability [8]. Therefore, in this study, we focus on the within-class variance of the SVM histogram as a design guideline for the DCNN architecture.

3. SVM Histogram of the CaffeNet

First of all, in order to evaluate the SVM histogram of the CaffeNet, trained by only DLD images, we make the SVM histogram of it.

3.1 Experiment Data

In this work, we used the HRCT images which were intended for DLD, provided by Osaka University Hospital. Under the doctor guidance, we labeled them into seven classes. Fig.4 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 also imply the NSIP, sometimes appears together with partial GGO patterns. The honeycomb (HCM) pattern has more rough mesh structure rather than that of the reticular pattern, and it appears in idiopathic pulmonary fibrosis (IPF) or usual interstitial pneumonia (UIP). Both of the nodular (NOD) and emphysema (EMP) are not DLDs, however, these patterns sometimes confuse physician, so that, we include these classes into the experiment. The normal (NOR) pattern appears in the normal tissue.

Under the doctor guidance, we split the HRCT images of the DLD into the two data sets, "Dataset 1" and "Dataset 2" in Table 1, and collected training data of the DCNN and validation data respectively. The seven-type lesions of DLD patterns on each HRCT images on the selected slices separately are marked by three radiologists. We extracted the patch images of 32 $\times$ 32 [pixel] from these lesions, and used as data. We call these images the ROI (Region of Interest) images. However, since the lesions are not always cut out in a square area of 32 $\times$ 32 [pixel], we selected the ROIs to be

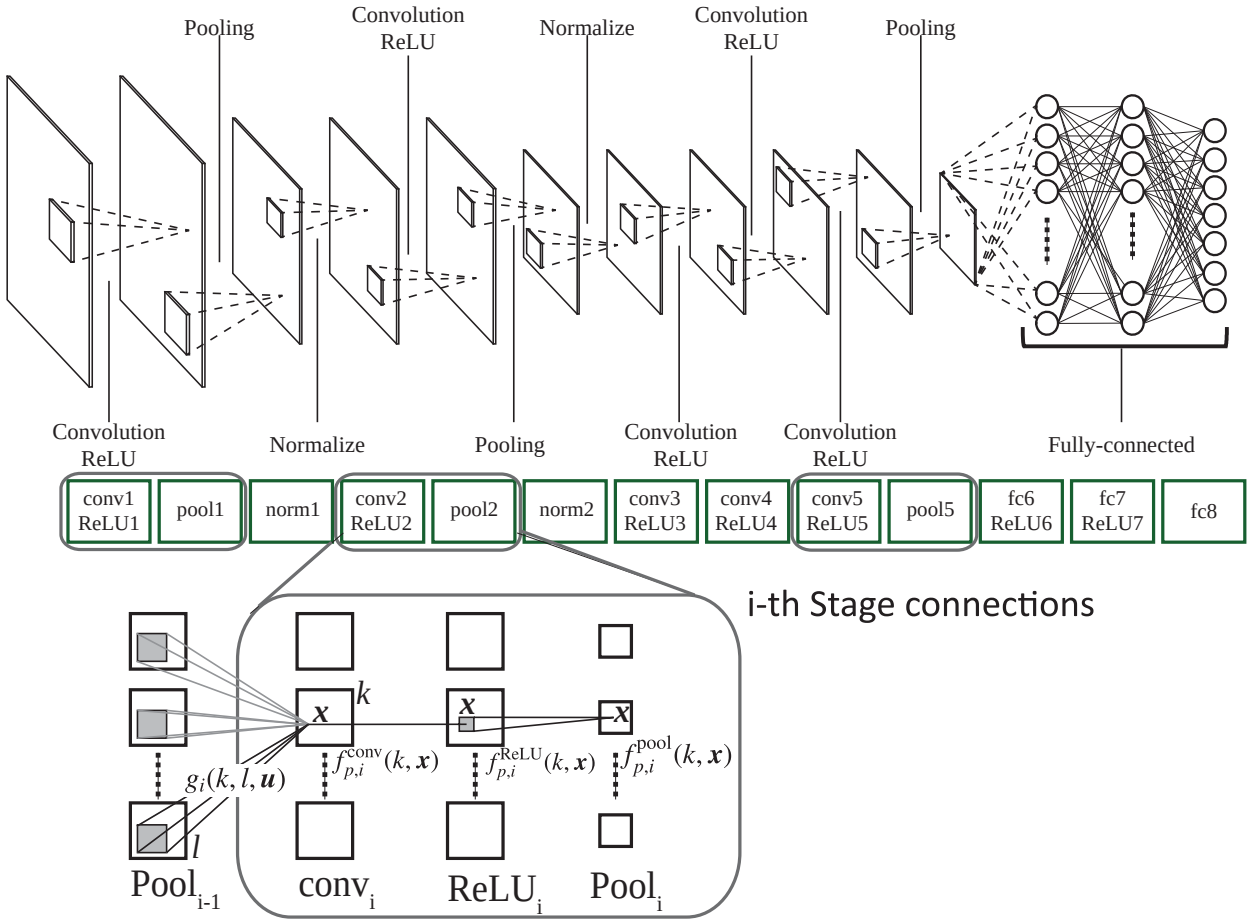


Fig. 2: Top: The summary of a kind of DCNN model CaffeNet. Bottom: The details of the process. In general, DCNN model acquire feature representation by repeating the processing Convolution → ReLU → Pooling.

occupied by the marked lesion with 80% area. Table 1 shows the number of collected ROI images. In Table 1, "Train" data and "Validation" has different order of numbers of the ROI images; this phenomenon is caused by the difference of the extraction. "Train" data are ROIs extracted from HRCT; ROIs are allowed to be overlapped itself when extracted. In contrast, "Validation" data are no-overlapped ROIs.

3.2 Experiment

In this experiment, we train the CaffeNet with Dataset 1 Train data in Table 1 and we make the SVM histogram with feature representation extracted from each intermediate layer, explained in section 2.2, of this CaffeNet with Dataset 2 Validation data in Table 1.

3.3 Result and Evaluation

The figure on the left of the Fig.5 shows the result of the SVM histogram for HCM vs RET classifier. Going through the hierarchy the overlapping part of the SVM histogram reduces, which implies that the classification performance is improving. Further, double-headed arrows shown on the top of the histogram are connecting the maximum and the minimum value of the SVM histogram. In our previous work, we found that it gets narrower through the hierarchical development of representation in the DCNN layers. In this figure, we can see that the SVM histogram becomes narrower in Pool1 and Pool2 than previous layers respectively. However, after Conv3 layer, the shape of the SVM histogram is almost unchanged, and class separation does not progress at all. We consider that this phenomenon prevents the improvement of the classification accuracy of the CaffeNet.

However, these discussions are only from the qualitative evaluation from the SVM histogram. In addition to this qualitative evaluation, we also carried out quantitative evaluation. We made all combination of the SVM histogram like HCM vs RET classifier shown in Fig.5, and summarize their within-class variances as boxplot in the figure on the right

Table 1: Dataset size for training and evaluation. The dataset is provided by Osaka University Hospital. Each item shows the number of ROIs, and the number of patients. For DCNN representation training, we extract square ROIs from the area pointed to by the physicians with overlapping. For validation of SVM histogram, we also extract ROIs without overlapping.

	Dataset 1			Dataset 2		
	Patients	Train	Val.	Patients	Train	Val.
CON	13	143	26	14	247	16
GGO	14	609	46	14	443	53
HCM	10	282	73	9	782	32
RET	8	210	66	9	210	66
EMP	10	4406	296	11	3760	296
NOD	9	762	65	10	591	85
NOR	11	5371	355	11	4595	406
Total	——-	11783	927	——-	10628	916

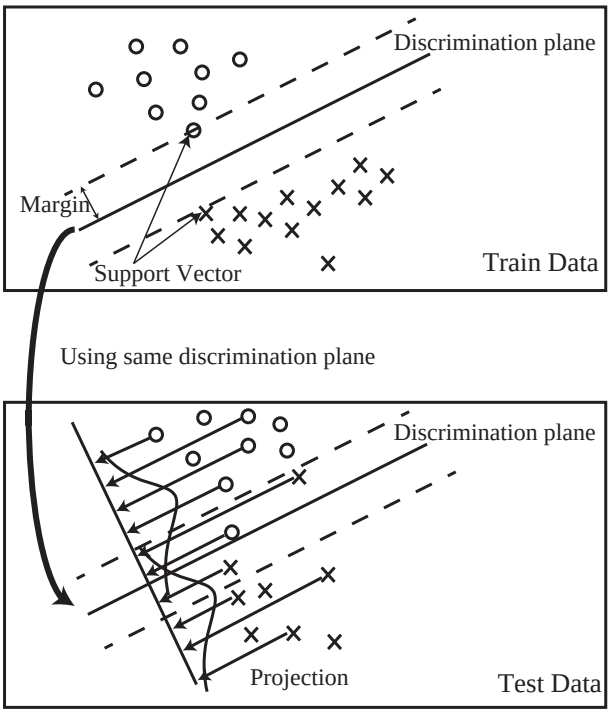


Fig. 3: Linear SVM finds the support vector which maximize the margin from feature space input data, and it make discrimination plane between the support vectors (Top). We create the histogram of the distance to the test data of discrimination plane (Bottom).

of the Fig.5. The within-class variance got smaller in all pooling layers, but it got larger in layers higher than Conv3.

4. Proposal Method and Accuracy Evaluation

In previous work, we attributed the improvement of the classification performance of DCNN to the phenomenon that the SVM histogram got narrower through the DCNN hierarchy [8]. On the other hand, section 3.3 shows that the CaffeNet with DLD ROIs cannot have such mechanism. Specifically, we can observe that within-class variance did not decrease in layers higher than Conv3. We think that the phenomenon is due to the architecture tuning of the CaffeNet not for DLD ROI recognition but for natural images recognition. Therefore, we attempt to modify the architecture design using the within-class variance for the guideline of the SVM histogram shown in Fig.5. In Fig.5, we can see that the within-class variance increase in Conv3 and Conv4 layer. So we think that these two layers do not adapt for DLD ROIs. Hence we propose a new DCNN architecture by removing the Conv3 and Conv4 layer from CaffeNet as shown in Fig.6 to tune for the DLD ROIs. In this section, we compare the CaffeNet and our new architecture for DLD ROIs classification.

4.1 Accuracy Evaluation

We train our new architecture shown in Fig.6 using train data in Table 1 of section 3.1, and calculate the classification accuracy of this DCNN with validation data in Table 1. Here, we used the 2-fold Cross-Validation (CV) method using the dataset 1 and 2 in Table 1 for calculation for classification accuracy. Table 2 compares the accuracy between the conventional CaffeNet and our proposal architecture. Here, we use top-1 accuracy calculation.

In Table 2, our proposal architecture beats the conventional CaffeNet. So this result shows that we succeeded to modify the architecture for DLD ROIs.

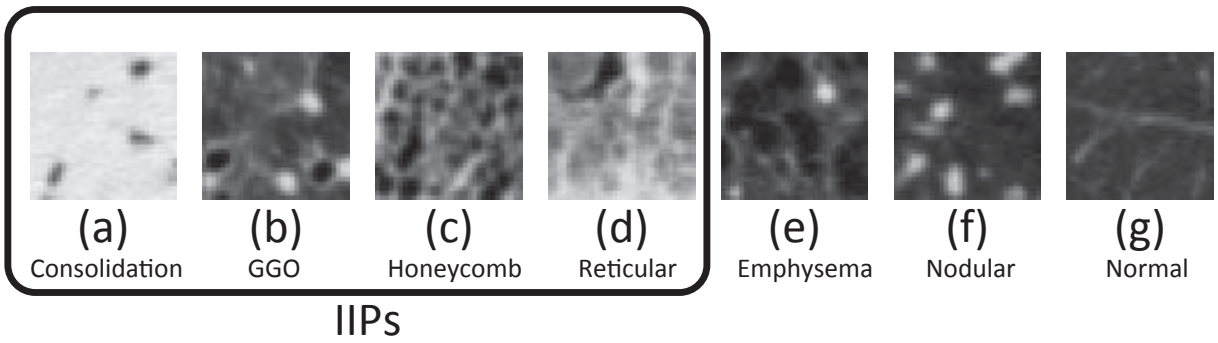


Fig. 4: Typical CT images of diffuse lung diseases: The top row shows each overview, and the bottom shows the magnified part (ROI) of each lesion. (a) to (g) represent “Consolidation (CON)”, “GGO”, “Honeycomb (HCM)”, “Reticular (RET)”, “Nodular (NOD)” “Emphysema (EMP)”, and “Normal (NOR)” image, respectively.

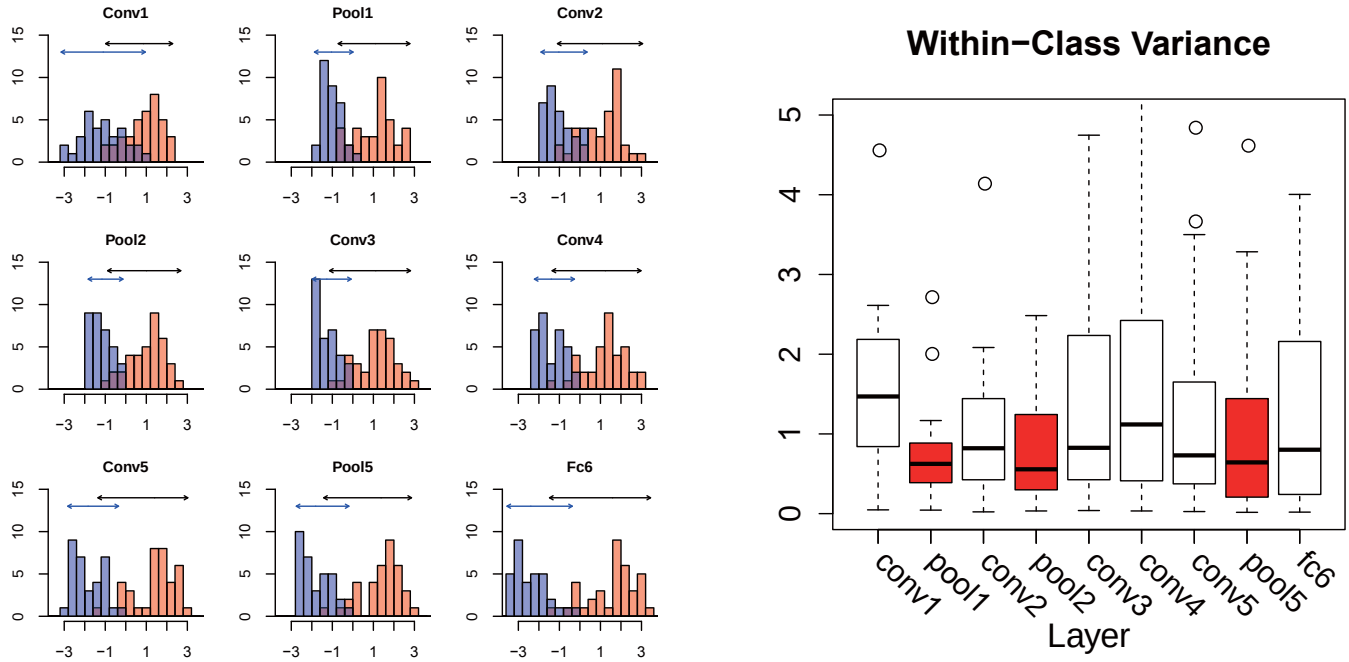


Fig. 5: Left: The SVM histogram of HCM class and RET class in CaffeNet. In the figure, each column shows the same layer results, and each row shows same epochs results. In each graph, the horizontal axis shows the distance from the decision plane, where the origin indicates the decision boundary, and the vertical axis shows the frequency of the test examples. Right: Boxplot of the within-class variance of SVM histogram in all of the combination 7 classes in the feature representation extracted from CaffeNet. The horizontal axis show layers, the vertical axis show the within-class variance.

Table 2: The accuracy comparison in the Top-1 accuracy of our model and CaffeNet.

DCNN	Accuracy
CaffeNet (5-conv + 3-full connect)	85.26 ± 2.16 [%]
Ours (3-conv + 3-full connect)	**88.39 ± 0.01 [%]**

4.2 Separation Analysis Using SVM Histogram

We try to modify the architecture using the within-class variance of the SVM histogram. We confirm whether the within-class variance of our proposal architecture decrease or not. Hence we make the SVM histogram and boxplot such as Fig.5 in Fig.7. The figure on the left of Fig.7 also shows the result of the SVM histogram of HCM vs RET classifier. In this figure, the SVM histogram of our proposal architecture looks narrower than CaffeNet. Most notably, though the SVM histogram of the CaffeNet becomes flat in Fc6 layer, the SVM histogram of our proposal architecture make clear clusters. Therefore, as Table 2 it is

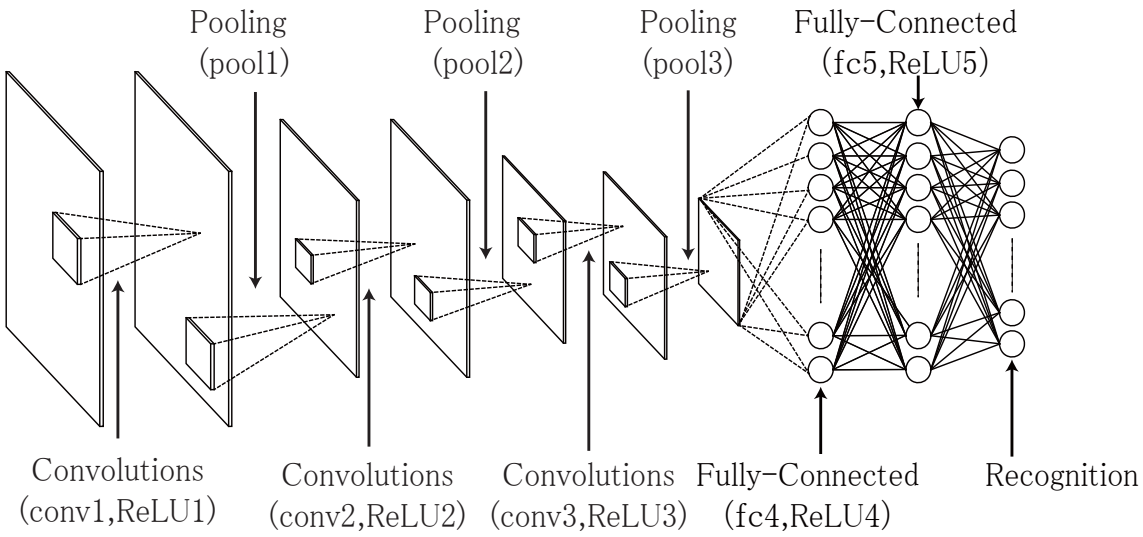


Fig. 6: Overview of the new DCNN architecture removing the conv3,4 layer from CaffeNet. The number of convolution filters and neuron of fully-connected layer is not changed.

considered that the classification accuracy of our proposal architecture beats the conventional CaffeNet. In the view point of quantitative evaluation, the figure on the right of Fig.7 shows that the within-class variance of our proposal architecture decrease in all Pooling layers and is smaller than the conventional CaffeNet. These results suggest that our proposal architecture works as expected.

5. Summary and Discussion

In this study, we proposed a new architecture guideline using the representation separation for the architecture design of DCNN, and modified the CaffeNet for DLD classifier. As a result, the classification accuracy of our proposal architecture beats that of the CaffeNet.

Architecture design of DCNN has not been made much research, because it is difficult to determine the design guideline, and the architecture design of DCNN had been a black box. On the other hand, it is difficult to directly apply DCNN to small datasets such as medical images, because generally, DCNNs such as AlexNet and CaffeNet have large scale free parameter. Thus in previous work, we used transfer learning with natural images for training. On the other hand, it is known that we are able to obtain better classification accuracy when extracted features from 3D images such as DLD HRCT images, note that it is different from DLD ROIs. We cannot apply the transfer learning method for 3D datasets, because natural images usually 2D images. However, our architecture design does not use transfer learning method, hence, our method may be extended for 3D images.

The residual problem is that the within-class variance in Fig.7 is not lower than our previous proposal model [8]. We think this problem is caused by not tuning various parameters as well such as the number of filters, filter size and kernel stride size. Hence, we consider that this problem will be solved by tuning such parameters using Zeiler's method [10].

Acknowledgment

This work is partly supported by MEXT/JSPS KAKENHI Grant number 26120515 and 16H01542. We thank Osaka University Hospital for providing valuable data for the experiment.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] L. Deng and D. Yu. Deep learning: Methods and applications. Technical Report MSR-TR-2014-21, Microsoft Research, May 2014.

[4] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, Vol. 36, No. 4, pp. 193–202, 1980.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[6] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, Vol. 1, No. 4, pp. 541–551, 1989.

[7] H. Shouno. Recent studies around the neocognitron. In M. Ishikawa, K. Doya, H. Miyamoto, and T. Yamakawa, editors, *Neural Information Processing, 14th International Conference, ICONIP 2007, Kitakyushu, Japan, November 13-16, 2007, Revised Selected Papers, Part I*, Vol. 4984 of *Lecture Notes in Computer Science*, pp. 1061–1070. Springer, 2007.

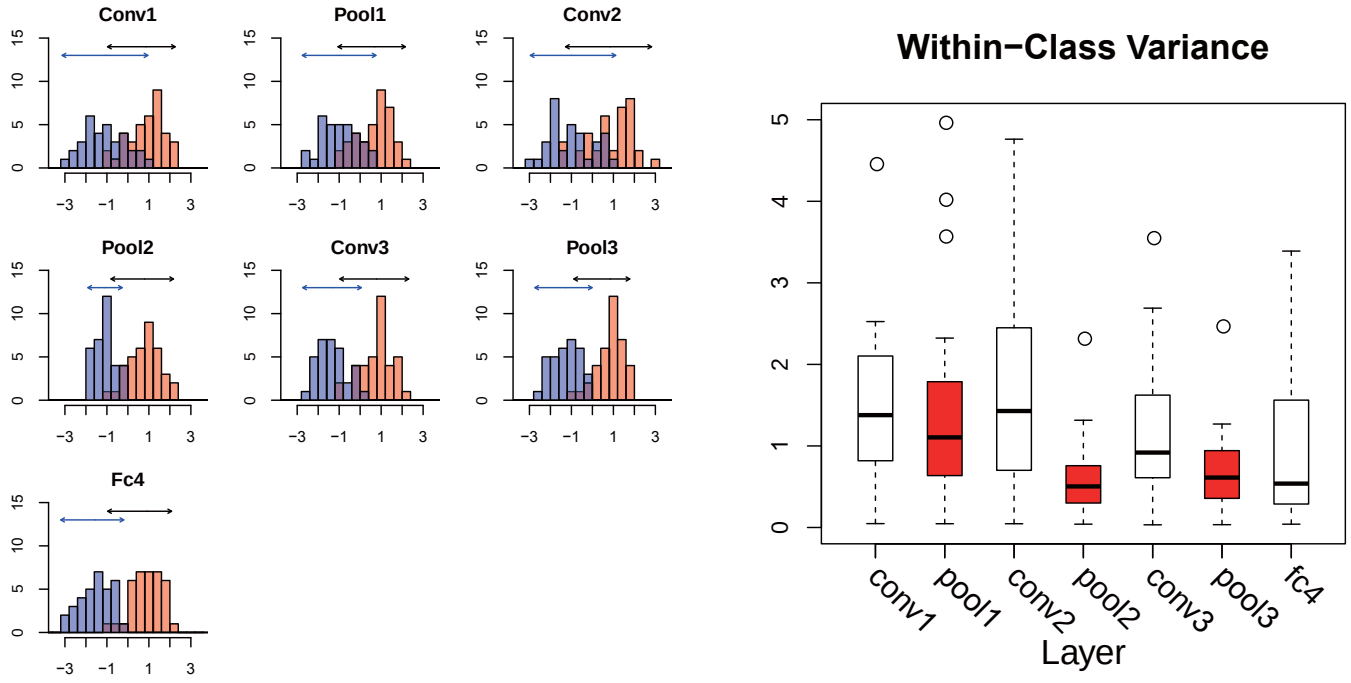


Fig. 7: Left: The SVM histogram of HCM class and RET class in CaffeNet. The manners of figure is followed Fig.5. Right: In the feature representation extracted from our proposal architecture, boxplot of the within-class variance of SVM histogram which in all of the combination 7 classes.

[8] H. Shouno, S. Suzuki, and S. Kido. A transfer learning method with deep convolutional neural network for diffuse lung disease classification. In *Neural Information Processing, 22nd International Conference, ICONIP 2015, Istanbul, Turkey, November 9-12, 2015, Proceedings, Part I*, Vol. 9489 of *Lecture Notes in Computer Science*, pp. 199–207. Springer, 2015.

[9] 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, pp. 183–190. Springer-Verlag Berlin Heidelberg, 2011.

[10] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pp. 818–833, 2014.

Biometric Authentication based on Multi-feature Combination using EEG

Yu Ishikawa, Kaori Nishibata, Masami Takata, Hiroyasu Kamo, Kazuki Joe
Nara Women's University, Nara, 630-8506, Japan

Abstract – *The Brain-Machine Interface (BMI) technology directly linking brain and machine has been actively studied. As authentication technologies are used in BMIs, we propose a personal authentication method using electroencephalogram (EEG). Research on EEG-based personal authentication has advanced in many fields. In this paper, we try to improve the accuracy of an EEG-based authentication with multi-feature combination proposed in existing researches. An ensemble-learning algorithm AdaBoost is used for multi-feature combination. A more suitable classifier is generated by considering the combination patterns of both features and electrode placements. In addition, we propose an authentication method using reliability calculated with AdaBoost. This method obtained an Equal Error Rate of 2.0%.*

Keywords: Biometric authentication, Brain waves, feature extraction, AdaBoost

1 Introduction

The Brain-Machine Interface (BMI) technology that links directly brain and machine has attracted attention. Non-invasive BMIs using EEG have been widely developed because it is safe and simple to be used for healthy people. Today, researches have advanced in various fields such as artificial arms and feet operated by EEG, wheelchair control systems that autonomously move using recorded EEG, and direct brain communication from person to person. In addition, some BMI technologies are already commercially available such as in "Necomimi" [1] and "MindRDR" [2]. Necomimi is a device that moves by reading user's emotions. MindRDR is an application that operates Google Glass by EEG. As just described, the BMI technology permeates daily life and improvement of authentication technology is thus demanded. In existing researches, personal authentications that use ID and password are widespread. However, such methods are easily at risk of spoofing when authentication is attempted from plagiarism or brute-force attacks. Therefore, biometrics are used nowadays to prevent authentication spoofing.

Biometric authentications are personal authentications that use biometric information. Today, biometric authentications through fingerprint, iris, face, and voiceprint analysis are studied and developed. It is difficult to plagiarize compared to existing password authentications. The authentications using fingerprint or iris offer high performance and are already put to practical use. However, reports emerge that authentication systems using them have been falsified [3]. One of the main reasons is the fact that such information is always exposed to the outside.

Therefore, authentication using EEG is devised. Since EEG includes the information inside the body, we need a dedicated measurement equipment. Thus, EEG authentication is difficult to plagiarize. In addition, considering its use in an BMI, since the authentication uses the same EEG as the BMI, it is more efficient than other biometric authentication methods. Researches of biometric authentications using EEG have already advanced in many fields. For example, a report proposed a personal authentication method with an accuracy of 80% by using EEGs of forty subjects during open-eye and closed-eye periods [4], and another report presented an accuracy of above 90% by analyzing EEG rhythms of four subjects during closed-eye periods [5]. There are other reports where personal authentication uses visual evoked potential [6] and verbal recall problems or potential recall movement [7]. As described above, various features and learning methods are proposed in existing researches.

In addition, the progress of electroencephalograph technology has been significant in recent years. Until now, multi-channel electroencephalographs were used only in the medical field and were applied by a professional engineer. However, personal measurement is now possible because many types of multi-channel electroencephalographs are developed and available in daily life. Advantages of using multi-channel electroencephalographs include the acquisition of space information that cannot be provided by a single-channel. Space information is widely applied in fields related to emotion identification using EEG [8] [9]. In this paper, in order to utilize space information as a feature, we perform authentication with a multi-channel electroencephalograph.

The rest of the paper is organized as follows: Section 2 reviews the related works about authentication using EEG. Section 3 describes the feature extraction used in this paper. Section 4 proposes a personal authentication method using the features extracted in Section 3. Section 5 reports the evaluation results of the proposed method. Section 6 concludes by discussing future work.

ISBN: 1-60132-444-8, CSREA Press ©

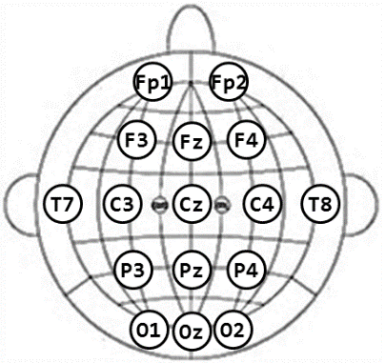


Figure 1 Electrode placement

2 Related works

Necessary processes for personal authentications with EEG include feature extraction, feature learning, and authentication decision. Existing methods to accomplish these are introduced below.

Current methods for EEG authentications use the power spectrum obtained from frequency analysis as the main feature. However, not only the power spectrum is always used. There are researches that use content rates by dividing frequency bands into multiple intervals [10] [11], the differences of spectrum between two hemisphere electrodes [12], convexo-concave patterns, maximal values, variances obtained from spectrum [13], and so on. In addition, there are several methods based on space information that use cross-correlation coefficients and mutual information to calculate similarities between electrodes [14] [15] [16]. Moreover, coherence is evaluated to show the phase-amplitude relationship [14] [15] and used as a feature. In this paper, we make efficient use of these features.

Machine learning has become the mainstream methods such as Linear Discriminant Analysis (LDA), Support Vector Machine (SVM), and Neural Network (NN) [10] [17] [18]. Besides machine learning, there are other methods using Gaussian Mixture Model (GMM) [7] and Bayesian Probability Model [19] too. In this paper, we classify EEGs using an SVM. Since SVM is usually used for classification, and not authentication, it is necessary to determine the acceptance or rejection against approval requests from classification results when applied as an authentication method.

A method in existing research [10] classifies each data by SVM after measuring multiple data. The number of correctly classified data is used for authentication decision. The authentication is accepted if the number is above a certain threshold, otherwise the authentication is rejected. This method makes SVM-based authentication available. However, this method requires long measurement times to set the threshold compared to the classification itself. Therefore, we propose in this paper an authentication method using AdaBoost to perform the authentication only from a one-time measurement.

3 Features extraction

3.1 EEG measurement and preprocessing

In this paper, we measure EEGs by using BioSemi that is a commercially available multi-channel electroencephalograph. A bipolar lead method is used for deriving the reference electrode. The sampling frequency of BioSemi is up to 2,048 Hz, and the electrode number is up to 256. In our measurement, we use a sampling frequency of 2,048 Hz with 16 electrodes. The electrodes are arranged according to the International 10-20 system [20] as shown in Figure 1.

With the following preprocessing, the measured data are corrected to the data suitable for the feature extraction described in subsection 3.2:

i. Band-pass filter,

ii. Noise reduction,

iii. Normalization.

First, the EEG frequency band of 4-40 Hz is extracted from the measured data using a band-pass filter with a Hamming window function. Although 1-3 Hz are defined as δ waves, in this paper, we exclude them because they include myopotential, oculogyration, and artifacts. In addition, the artifacts caused by the environment such as AC interference appearing at 60 Hz are also removed by the band-pass filter. Next, in order to remove pulse noises appearing temporarily, standard deviations are calculated from the filtered data and the data above 3σ are converted to 3σ. Finally, the data are normalized to [0,1] after noise reduction. Applying this preprocessing, the data are reconstructed with removing the bias and making them suitable for authentication.

3.2 Features

3.2.1 Power spectrum

As like in most popular EEG analysis methods, the power spectrum calculated by Fast Fourier Transform (FFT) is used. EEGs are classified according to frequency bands. It is known that EEG's characteristics are different depending on their classes. Therefore, the frequency analysis is the most efficient way to obtain features from EEG.

When applying FFT, the data length must be a power of two. As in the preprocessing, a window function is applied to data before FFT. The power spectrum (PS) is calculated with the real part (Re) and the imaginary part (Im) obtained from FFT given in the following equation:

$$PS = \sqrt{Re^2 + Im^2}.$$

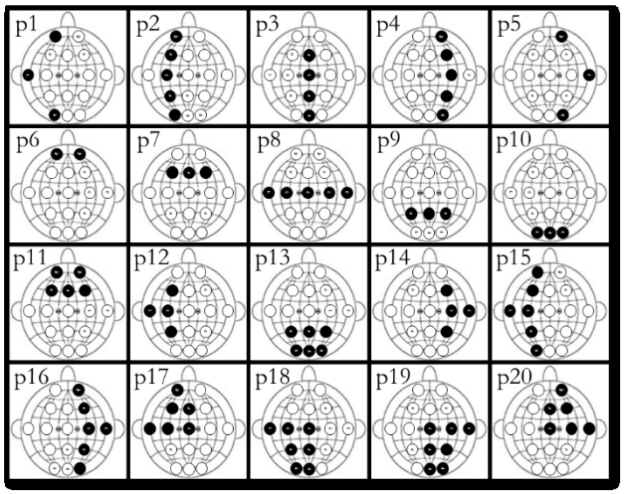


Figure 2 Electrode combination patterns

The power spectrum is obtained for each channel. Average content rates are calculated for each frequency band, including θ waves (4-8 Hz), α waves (8-14 Hz), β waves (14-26 Hz), and γ waves (26-40 Hz) which are obtained from the results of the power spectrum. We use these four frequency bands as features.

3.2.2 Cross spectrum

The cross spectrum is used for frequency analysis between two electrodes. FFT is applied to the data after multiplying them by a window function as in the power spectrum evaluation. The cross spectrum (CS) between two electrodes (a and b) is presented in the following equation:

$$CS = \sqrt{(Re_a \cdot Re_b + Im_a \cdot Im_b)^2 + (Re_a \cdot Im_b + Im_a \cdot Re_b)^2}.$$

As in the power spectrum evaluation, the average content rates are calculated for each frequency band from the cross spectrum and are used as features.

3.2.3 Coherence

Coherence (COH) is used to obtain the phase-amplitude relationship of waves obtained from two electrodes.

$$COH = \frac{CPS_{ab} * CPS_{ab}}{PS_a * PS_b}.$$

As in the power spectrum evaluation, the average content rates are calculated for each frequency band from the coherence and are used as features.

3.2.4 Cross-correlation

By using cross-correlation, the similarities between electrodes are used as features. Cross-correlation is applied in a wide range of fields, such as emotion estimation of pleasant/unpleasant states and EEG analysis during exercise.

The cross-correlation (CC) between two electrodes is calculated according to the following equation:

$$CC = \frac{\sum_{i=1}^{N}(a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{N}(a_i - \bar{a})^2}\sqrt{\sum_{i=1}^{N}(b_i - \bar{b})^2}},$$

where N is the data length. Two electrodes are positively correlated when the cross-correlation is 1, negatively correlated when -1, and not correlated when 0. The length of each data set is divided into four parts and the cross-correlation is calculated for each part.

3.2.5 Mutual information

The mutual information (MI) is calculated to evaluate the interdependence between two electrodes as in the following equation:

$$MI = \sum_{i=1}^{N}\sum_{j=1}^{N} p(a_i, b_j) \log \frac{p(a_i, b_j)}{p(a_i)p(b_j)}.$$

Discretization of data is necessary as preprocessing. The data length is divided into four parts, and the mutual information of each part is calculated.

3.3 Electrode combination

In order to confirm if the electrodes are suitable for personal authentication, we have to consider the combination patterns of electrodes. Figure 2 shows a list of the electrode combination patterns. In addition to the 20 patterns in the list, we use a total of 21 patterns by including an extra pattern using all 16 channels.

4 Authentication method

4.1 AdaBoost

We apply AdaBoost to perform the personal classification using EEG. AdaBoost is one of ensemble-learning algorithms which combine some weak classifiers by Boosting. In particular, it is a method which adaptively updates learning data weight. The final classifier is determined by a weighted majority vote of the weak classifiers. The AdaBoost algorithm is as follows, with M the number of learning epochs:

1 Calculate feature $x_n(n = 1, \dots, N)$ from learning data and label each subject.

2 Initialize the learning data weights w_n to $1/N$.

3 Calculate candidate weak classifiers $f_l(l = 1, \dots, L)$.

4 Repeat 4.1 to 4.4 with $m = 1, \dots, M$:

4.1 Repeat 4.1.1 with each candidate $l = 1, \dots, L$:

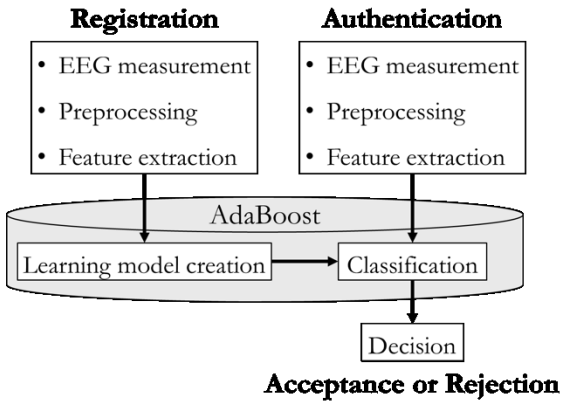


Figure 3 Authentication system flow

4.1.1 Calculate the error rate ϵ_m from each candidate weak classifier. The target value of $f_l(x_n)$ is defined as t_n:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n I\left(f_l(x_n)\right)}{\sum_{n=1}^{N} w_n},$$

$$I\left(f_l(x_n)\right) = \begin{cases} 0, & f_l(x_n) = t_n \\ 1, & otherwise \end{cases}.$$

4.2 Choose a weak classifier y_m which has the smallest error rate among the candidate weak classifiers.

4.3 Calculate the reliability α_m using the error rate of the chosen weak classifier:

$$\alpha_m = \frac{1}{2}\ln\left(\frac{(1-\epsilon_m)}{\epsilon_m}\right).$$

4.4 Update weights. The total of the weights is 1.

5 Constitute a strong classifier:

$$Y(x) = \sum_{m=1}^{M} \alpha_m y_m(x_n).$$

As features, we use a combination of 5 different features, as described in subsection 3.2, and the 21 electrode placement patterns described in subsection 3.3. By applying an SVM to these features, 105 candidate weak classifiers are created. The SVM uses the Radial Basis Function (RBF) kernel. In this paper, the number of learning epochs *M* is 80. The created strong classifier is used as the learning model for personal authentication.

4.2 Authentication system

We perform authentication with the personal classification results obtained using AdaBoost. The

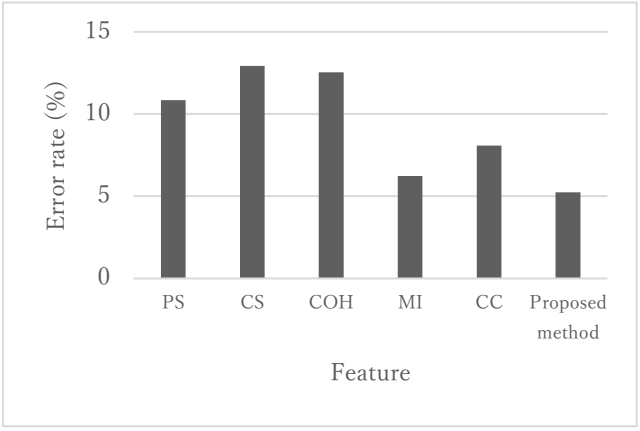


Figure 4 Classification rate of each feature

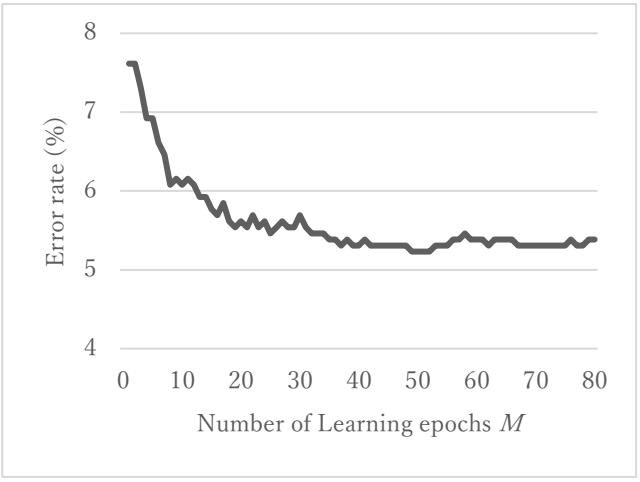


Figure 5 Error rate by number of learning epochs using AdaBoost

authentication system flow is shown in Figure 3. The system is composed of a registration phase and an authentication phase. In the registration phase, multiple features are extracted from the measured data after the preprocessing, as described in section 3. The learning model is then created based on these obtained data by using AdaBoost, as presented in subsection 4.1. In the authentication phase, as in the registration phase, the features are extracted from the measured data. Using the learning model created in the registration phase, authentication data are classified into a registrant with the highest reliability. The decision is given by the total of reliabilities calculated in AdaBoost:

$$R = \sum_{m=1}^{M} \alpha_m.$$

If the total reliability *R* exceeds a threshold, the data are accepted and considered as a registrant. Otherwise, the data are rejected and considered as a non-registrant.

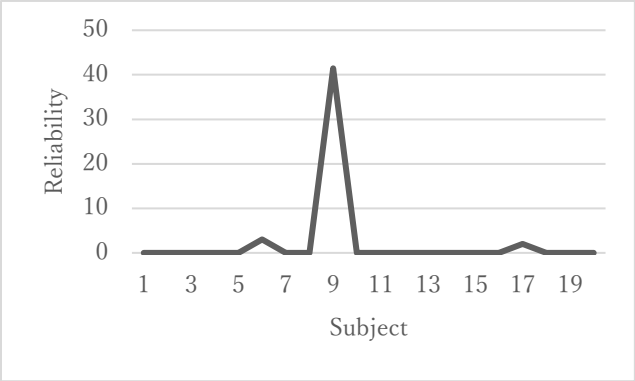


Figure 6 Reliability of approval request data (in the case of subject 9)

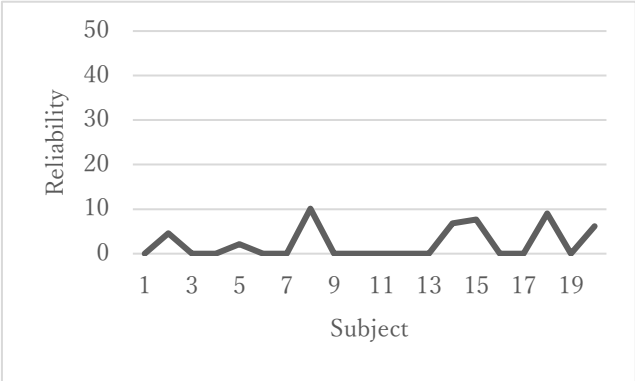


Figure 7 Reliability of approval request data (in the case of an intruder)

5 Results

In this paper, we show two kinds of the classification and the authentication results. The test subjects are 26 females in their twenties. The subjects are measured in a sitting state and a relaxing state. A ten-second measurement is performed five times in one session per subject. This is repeated ten times. The total number of data is $26 * 5 * 10 = 1{,}300$. Since the data length must be a power of two, we use the data of the first eight seconds ($2{,}048$ Hz $* 8$). The number of data for the classification is 1,300 of 26 subjects. The number of data for the authentication is 1,000 of 20 registrants and 300 of 6 intruders. We use 10 cross-validation to calculate these results.

5.1 Classification results

The classification accuracy is validated as follows. First, the classification error rates of the five features (PS, CS, COH, CC, MI) are calculated by using an SVM. We use electrode data from all 16 channels. Next, the classification rate of the proposed method using AdaBoost is calculated. Figure 4 shows a graph of the classification rates for PS, CS, COH, CC, MI, and the proposed method using AdaBoost. The vertical axis shows the error rate. Among the five features, MI provides the best results, followed by CC, PS, COH, and CS. In these features, the classification error rates using frequency analysis are relatively low. On the other hand, classification rates exceeding 90% using only one feature are obtained from CC and MI.

Next, the results of the proposed method using AdaBoost are considered. The error rate is 5.2% when the number of learning epochs is from 49 to 51 and is the best. Figure 5 shows the relation between the number of learning epochs and the classification rate. The graph shows the average after 10 cross-validations. At the first learning, the error rate is 7.6 %, which is comparable to the results using one feature. The error rate gradually decreases as repeating the learning. The error rate then decreases down to 5.2% at the 49th learning epoch, and saturates around 5.2-5.4% as the number of learning epochs increases. From this result, we set the number of learning epochs used for the authentication at 50.

5.2 Authentication results

To examine the authentication accuracy, it is necessary to perform two types of validation:

A) A registrant authentication test,

B) An intruder authentication test.

A) validates the case of legal and impostor authentications of registrants. B) validates the case of impostor authentications of intruders.

Figure 6 and Figure 7 show the results of two case examples acquired by calculating the reliability of the approval request data. The horizontal axis shows the subject number among the 20 registrants, and the vertical axis shows the reliability. Figure 6 shows the case where subject 9 requests an approval. According to this graph, the reliability of the subject 9 is 41, which is highest value among the all registrants. Although the results of registrants 4 and 17 show a small reaction, their reliabilities are low enough. Figure 7 shows the case where an intruder requests an approval. A reaction is obtained from multiple registrants (8, 14, 15, 18, 20). However, all of the reacts are less than 10. Therefore, we consider that we can use the classification as an authentication by setting the threshold of reliability between 10-40.

Figure 8 and Figure 9 show the reliability of all of approval request data. The horizontal axes show the subjects of approval request data which is obtained from registrants and intruders, and the vertical axes show the registrants. Figure 8 shows the reliability of validation A). There is clear difference in reliability between the legal and impostor approval requests. A lot of data with a reliability of more than 30 appear in the case of legal authentications. On the other hand, the reliability of most data is less than 15 in the case of impostor authentications. Figure 9 shows the reliability of

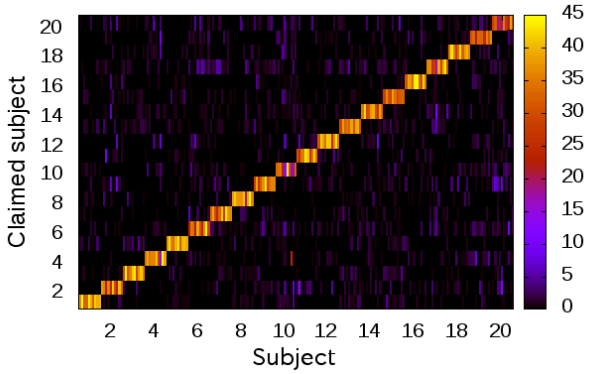


Figure 8 Reliability of validation A)

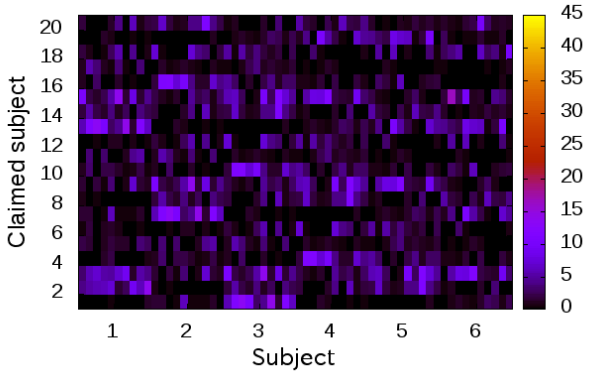


Figure 9 Reliability of validation B)

validation B). No data are considered reliable as the reliability of almost all data is less than 15.

In order to determine the acceptance or rejection of approval requests, the relation between the authentication rate and the threshold variation is investigated. The authentication performance is evaluated by Equal Error Rate (EER). EER is calculated from False Rejection Rate (FRR) and False Acceptance Rate (FAR). FRR is calculated from the correct data of registrants ($20 * 5 * 10 = 1000$), and FAR is calculated from the impostor data of registrants ($20 * 19 * 5 * 10 = 19000$) and the intruder data ($6 * 5 * 10 = 300$). By definition, EER is the value obtained when FRR equals FAR. Figure 10 shows the results of FRR and FAR, with EER at their intersection. When the threshold is 8.3, EER is 2.0%. Thus, a high authentication rate is obtained by using this method. When setting a threshold of higher reliability, although FRR increases, FAR stays reasonably low. Therefore, safety becomes higher. From these results, we consider that it is possible to prevent the misrecognition of impostor data by setting a suitable threshold.

6 Conclusions

In this paper, we proposed a biometric authentication based on multi-feature combination using EEG. We apply AdaBoost using SVM to data as the method to combine

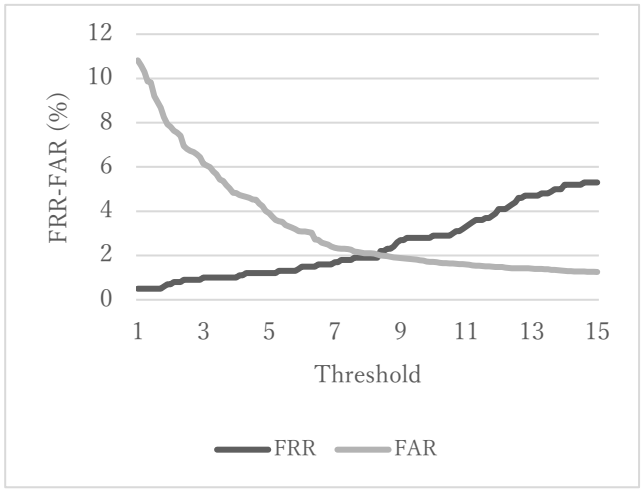


Figure 10 FRR and FAR

multi-feature. Therefore, it is possible to perform the authentication from one-time measurement. A combination of 5 different features and 21 electrode placement patterns are used as features to create the candidate weak classifiers applied to AdaBoost. The classification rate result is 94.8%. In general authentication methods using SVM, we need to determine the acceptance or rejection by a majority vote from multiple measured data. In this proposed method, we are able to perform the approval decision from a one-time measurement using the reliability provided by AdaBoost. We obtained an EER of 2.0% when validating the proposed system results with 20 registrants and 6 intruders. From these results, we consider that it is possible to prevent impostor data by setting a suitable threshold.

Future work will focus on improving the authentication accuracy by selecting more suitable features. Although eight seconds of data are still needed for our present authentication method, we may perform faster authentications using shorter measurement by selecting the features appropriately. In addition, we need to consider the setting method of the suitable threshold.

Acknowledgments

This work was partly supported by Grant-in-Aid for JSPS Fellows (16J10436).

References

[1] Neurowear, Projects/necomimi, <http://neurowear.com/projects_detail/necomimi.html>, accessed 5 Aug 2016

[2] This Place, MindRDR, < http://mindrdr.thisplace.com/static/index.html>, accessed 5 Aug 2016

[3] Matsumoto T., Matsumoto H., Yamada K., & Hoshino S. (2002, April). Impact of artificial gummy fingers on fingerprint systems. In Electronic Imaging 2002 (pp. 275-289). International Society for Optics and Photonics.

[4] Paranjape R. B., Mahovsky J., Benedicenti L., & Koles Z. (2001). The electroencephalogram as a biometric. In Electrical and Computer Engineering, 2001. Canadian Conference on (Vol. 2, pp. 1363-1366). IEEE.

[5] Poulos M., Rangoussi M., Chrissikopoulos V., & Evangelou A. (1999, September). Parametric person identification from the EEG using computational geometry. In Electronics, Circuits and Systems, 1999. Proceedings of ICECS'99. The 6th IEEE International Conference on (Vol. 2, pp. 1005-1008). IEEE.

[6] Palaniappan R., & Mandic D. P. (2007). Biometrics from brain electrical activity: A machine learning approach. IEEE transactions on pattern analysis and machine intelligence, 29(4), pp. 738-742.

[7] Marcel S., & Millán J. D. R. (2007). Person authentication using brainwaves (EEG) and maximum a posteriori model adaptation. IEEE transactions on pattern analysis and machine intelligence, 29(4), pp. 743-752.

[8] Schaaff K., & Schultz T. (2009, September). Towards emotion recognition from electroencephalographic signals. In 2009 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops (pp. 1-6). IEEE.

[9] Petrantonakis P. C., & Hadjileontiadis L. J. (2011). A novel emotion elicitation index using frontal brain asymmetry for enhanced EEG-based emotion recognition. IEEE Transactions on Information Technology in Biomedicine, 15(5), pp. 737-746.

[10] Yoshikawa T., Nakanishi I., Li S., (2013). Person Authentication Using EEG -Verification Based on 1vs1SVM with Divided EEG Spectra-. Proc. of the 2013 International Workshop on Smart Info-Media System in Asia, pp. 367-371.

[11] Chuang J., Nguyen H., Wang C., & Johnson B. (2013, April). I think, therefore I am: Usability and security of authentication using brainwaves. In International Conference on Financial Cryptography and Data Security (pp. 1-16). Springer Berlin Heidelberg.

[12] Palaniappan R. (2008). Two-stage biometric authentication method using thought activity brain waves. International Journal of Neural Systems, 18(01), pp. 59-66.

[13] Nakanishi I., Baba S., & Miyamoto C. (2009, January). EEG based biometric authentication using new spectral features. In Intelligent Signal Processing and Communication Systems, 2009. ISPACS 2009. International Symposium on (pp. 651-654). IEEE.

[14] Riera A., Soria-Frisch, A., Caparrini M., Grau C., & Ruffini G. (2007). Unobtrusive biometric system based on electroencephalogram analysis. EURASIP Journal on Advances in Signal Processing, 2008(1), pp. 1-8.

[15] Safont G., Salazar A., Soriano A., & Vergara L. (2012, October). Combination of multiple detectors for EEG based biometric identification/authentication. In Security Technology (ICCST), 2012 IEEE International Carnahan Conference on (pp. 230-236). IEEE.

[16] Ursulean R., & Lazar A. M. (2009). Detrended cross-correlation analysis of biometric signals used in a new authentication method. Electronics and Electrical Engineering. Kaunas: Technologija, (1), 89.

[17] Ashby C., Bhatia A., Tenore F., & Vogelstein J. (2011, April). Low-cost electroencephalogram (eeg) based authentication. In Neural Engineering (NER), 2011 5th International IEEE/EMBS Conference on (pp. 442-445). IEEE.

[18] Brigham K., & Kumar B. V. (2010, September). Subject identification from electroencephalogram (EEG) signals during imagined speech. In Biometrics: Theory Applications and Systems (BTAS), 2010 Fourth IEEE International Conference on (pp. 1-8). IEEE.

[19] He C., Lv X., & Wang Z. J. (2009, April). Hashing the mAR coefficients from EEG data for person authentication. In 2009 IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 1445-1448). IEEE.

[20] Jasper H. H. (1958). The ten twenty electrode system of the international federation. Electroencephalography and clinical neurophysiology, 10, 371-375.

Correlation of proximity voluntary muscles EMG and EEG

Hitomi Oigawa [1], Yu Ishikawa[1], Tomohiro Umeda [2], Masami Takata[1], Kazuki Joe[1]
[1]Nara Women's University, Nara, JAPAN
[2]Nara Medical University, Nara, JAPAN

Abstract - *In this paper, we study correlations of proximity voluntary muscles and EEG. We simultaneously measure the acceleration of fingertips, EMG of a forearm and EEG. The respective time-series data are preprocessed for the correction of positional deviation. We apply frequency analysis and cross-correlation coefficient analysis. As the results, we find correlations between fingers, arms and brain waves. In particular, the correlations of acceleration/EMG and acceleration/EEG seem to be effective. Also we find meaningful difference between right and left hands.*

Keywords: BMI, BCI, EEG, EMG, Acceleration

1 Introduction

As a communication pathway between brain and external machines in recent years, Brain-Machine Interface (BMI) or Brain-Computer Interface (BCI) is widely known. The existing BMIs are divided in two groups. The first group is to control the neurotransmission directly from computer to brain. For example, Deep Brain Stimulation (DBS) directly sends electrical pulses to a chip embedded in a brain and it manipulates brain activity [1]. The second group is to transmit some information from brain to machine. For example, wheelchair manipulation techniques use electroencephalogram (EEG) [2].

A lot of challenges are in progress such as reproduction technologies for physical function, say prosthetic hands. A disable person may comfortably and freely walk in a virtual space for the sake of such reproduction technologies. However, the challenges have not been succeeded yet. Brain is too complicated to get real pattern recognition information. Especially, human's hands and fingers are very sensitive even in everyday life. They say a large part of brain may be used for the movement of hands and fingers [3]. We think they are deeply related, but no one proved it so far. Of course, there are intellectual activities of human's hands and fingers but they are far away from the current technology. For example, studies with invasive EEG or fMRI with a constraint state are under heavy restrictions on the measurement conditions for use in real life [4]. Although the averaging method using for pattern extraction of EEG is common, it requires a lot of trials and long time.

To study brain, we propose a sort of data mining applied to brain waves. We call the method EGG pattern mining. Our research purpose is to increase the efficiency of the EEG pattern mining. We try to achieve it in a reasonably short period of time with a high degree of accuracy. As a solution, we consider deep learning neural network (DL).

Neural network is a kind of machine learning using a neuron model. The techniques for classifying EEG pattern have been proposed in [5]. DL learns to increase the layers in the structure of a conventional neural network in a reasonably short period of time with a high degree of accuracy. It has great achievements in the field of image recognition [6]. As the processing capacity of computer has been improved, it becomes a very useful technique. However, determination of the structure and parameter choices in neural network is known as common problem. So, we use DL based on vital data obtained from a human body as well as EEG for the solution.

In this paper, as the first step toward the above solution, we study correlations of proximity voluntary muscles and EMG. We simultaneously measure the information of proximity voluntary muscles and brain waves. The information of proximity voluntary muscle refers to the acceleration of fingertips and EMG from forearm that is obtained from a multi-channel sensor device. EMG obtained from forearm receives a command from the brain and plays the fundamental for generating a motion of fingers. As for extraction of EMG patterns, there are many research reports [7] [8]. In this paper, since we use a multi-channel sensor device Myo, we can get several states of closely spaced voluntary muscles. The acceleration of fingertips captures the movement of fingers to be moved by some proximity voluntary muscles. By correlating fingertips acceleration and EMG information with brain waves, efficient EGG pattern mining based on the vital data obtained from a body is expected.

The rest of the paper is organized as follows. In section 2, we describe related works. We describe a calculation method for correlation of proximity voluntary muscles EMG and EEG in section 3. The experimental results and discussions are presented in Section 4.

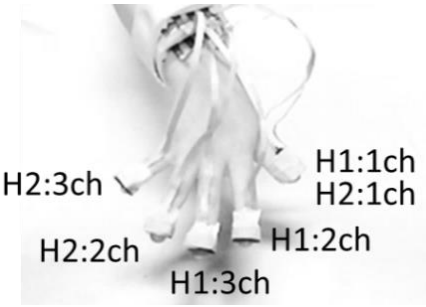


Fig.1 HapLog

Fig.2 Myo

2 Related works

There are many research reports about the effects of EEG regarding to a motion or an image of body movement. The frequency of brain waves is analyzed when stimulated in sight and hearing [9]. According to the existing research, a frequency component increases or decreases before or after the given stimulus. They are referred to Event Related Synchronization (ERS) or Event Related Desynchronization (ERD). In [10], it is reported that different body parts of motion give different brain regions where ERS and ERD are observed. The movement or its image of hands and feet varies the rhythm components of α and β waves in the motor area (central region of the brain). The difference between left and right parts of body causes the same kind of reaction appearing in the hemisphere opposite to the activity parts.

The above research results show that the movement of body parts and brain wave patterns are clearly correlated although they are from the measurement of brain waves. Namely, by combining EEG measurement and several body sensors, the relationship between body movement and EEG pattern is analyzable. We propose the following technique and present experimental results in later sections.

3 Calculation method for correlation

We propose a calculation method for the correlation of acceleration of fingertips or proximity voluntary muscles EMG with EEG. The method is shown as follows.

1. Simultaneous measurement of three sensors
2. Preprocessing
3. Analysis of measurement results

In the first step, we use three sensors to measure acceleration of fingertips, forearm EMG and EEG with opening and closing a palm several times. In the next step, we preprocess the data obtained from each sensor. Then, we correct the position deviation because the timing of the measurements of three vital data may be slightly deviated. Finally, we analyze the results to get the correlation of fingertips acceleration/proximity voluntary muscles EMG and EEG.

3.1 Measurement and Preprocessing

3.1.1 Acceleration of fingertips movement

For the acceleration of fingertips movement, we use HapLog shown in Fig.1 [11]. It has three sensor channels to measure the 3-axis direction accelerations and the finger pressures with sampling rates of 1 to 1,000Hz. In this paper, we set the sampling rate to 200Hz. Then, we get five fingertips data from a pair of HapLogs. The three sensors of the first HapLog (H1) is put to the thumb, the index finger and the middle finger while the second one (H2) is put to the thumb, the ring finger and the little finger.

In the analysis, the 3-axis direction accelerations are synthesized. We correct the position deviation using a pair of thumb data because the timing of the measurements of two Haplogs may be slightly deviated. By correcting the positional deviation on the thumb, we obtain the data measured simultaneously. We use a cross-correlation function for the correction. The expression is shown as

$$r_k = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_{i+k} - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_{i+k} - \bar{y})^2}} \quad (1)$$
$$k = 1,2,\ldots,a$$
$$n = N - a$$

where a is the range of shifting. We determine the start point k as the maximum value of the cross-correlation coefficient. In this experiment, a is 200 and k is 1 second. EMG and EEG data are extracted according to the deviation position with the same time stamp. Furthermore, we interpolate the acceleration data to adjust the EEG data length using a spline interpolation.

3.1.2 EMG

For measuring the forearm EMG, we use Myo [12]. Figure 2 shows the EMG sensor Myo with 8 myoelectric sensors, a 3-axis direction acceleration sensor, a 3-axisdirection magnetic force sensor and a sensitive 9-axis Inertial Measurement Unit (IMU) including a 3-axis direction gyroscope sensor with the sampling rate of 200Hz. The electrode with an LED light is channel 4, and the channel number decreases and increases toward the left and right direction, respectively.

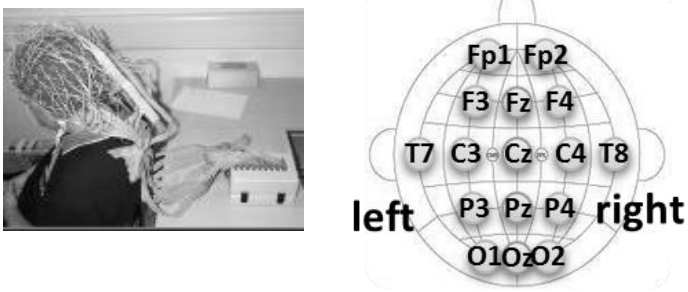


Fig.3 Biosemi(left) and the electrodes(right)

As in the same way to the correction of acceleration data, we interpolate EMG data for EEG using a spline interpolation. There is electrical activity in muscles. When muscles are tense a large amount of current flows. We capture the movement of palm opening and closing to get the average amplitude of EMG. We use the root mean square (RMS) method shown below with the range of 50 msec (N=200).

$$RMS_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i)^2} \qquad (2)$$

3.1.3 EEG

For measuring the EEG, we use a Biosemi [13] with 16 channels and the sampling rate of 2, 4, 8 or 16 kHz. In this paper, we set the sampling rate to 2,048Hz. Figure 3 shows the sensor and the electrodes placement based on International 10–20 system. A bipolar lead method is used for deriving the reference electrode. The preprocessing for EEG is as follows.

i. Cepstrum analysis
ii. Band-pass filter
iii. Addition average
iv. Comparison
v. RMS

The raw data of EEG sometimes contains envelopes because of voltage trace change. To remove the envelopes, a cepstrum analysis is applied. Next, we apply a band-pass filter of 0.5-40Hz to perform averaging. This is needed for noise reduction. Averaging EEGs is calculated with adjusting the deviation based on the measured EEGs as a bias to be averaged. It is required for the comparison with other vital data of the EMG and the acceleration. In terms of adding to the average, the position deviation is corrected by the cross-correlation coefficient. In this experiment, we adopt a = 2000 and k is 1 second.

For comparison, we subtract the relaxed state data from the active state data to reduce noises. This time we also use the cross-correlation coefficient to adjust the deviation. We use a = 2000 and k is 1 second. The electrical activity of brain wave is recorded just like EMG. Finally, we obtain the average amplitude of the data using RMS.

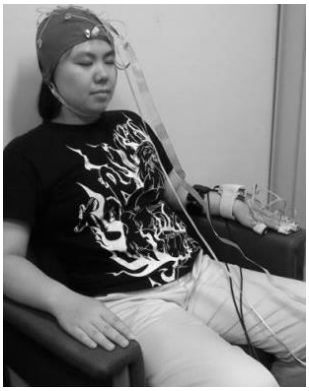


Fig.4 Wearing three kinds of sensors

3.2 Analysis

In the analysis phase, we perform frequency analysis and cross-correlation coefficient analysis. In the frequency analysis, we apply FFT to the following data.

- Fingers movement acceleration
- Forearm EMG (RMS)
- EEG (RMS)

In the cross-correlation coefficient analysis, we obtain the maximum value of cross-correlation coefficients assuming a = 2000 and k is 1 second. We compare the following three cross-correlation coefficients.

- Acceleration vs Forearm EMG (RMS)
- Acceleration vs EEG (RMS)
- Forearm EMG (RMS) vs EEG (RMS)

We compare the channels each other.

4 Experiments

4.1 Measurement

Figure 4 shows an examinee wearing HapLogs, a Myo and a Biosemi. The way of gripping is to put the thumb in the palm of her hand. The examinees are 3 females in 20s. The number of measurement trials is 100. The measuring state is sitting and eyes closed. The detail of states is as follows.

- Relaxed state
  - Silent×10 trials
  - Sound stimulus (Once a second) ×10 trials
  - Sound stimulus (Twice a second) ×10 trials
- Active state(Opening and closing movement of palm)
  - Sound stimulus (Once a second) ×10 trials
  - Sound stimulus (Twice a second) ×10 trials

Tab. 1 Concordance rate of the acceleration (Right)

R	thumb	index	middle	ring	little
A	90	95	90	90	85
B	100	100	100	100	100
C	65	75	60	65	90

Tab. 2 Concordance rate of the acceleration (Left)

L	thumb	index	middle	ring	little
A	95	95	95	95	95
B	100	100	100	100	100
C	65	65	65	65	70

Tab. 3 Concordance rate of the EMG (Right)

R	1	2	3	4	5	6	7	8
A	95	70	90	75	95	95	95	50
B	95	100	100	80	95	90	100	95
C	75	80	100	95	100	90	100	75

Tab. 4 Concordance rate of the EMG (Left)

L	1	2	3	4	5	6	7	8
A	100	65	65	95	90	95	90	90
B	100	60	100	100	100	100	100	40
C	100	90	100	95	95	40	60	45

Tab. 5 Concordance rate of the EEG (Right)

R	Fp1	Fp2	F4	Fz	F3	T7	C3	Cz
A	50	55	50	55	55	15	40	25
B	30	5	20	20	40	25	30	20
C	35	20	20	10	40	35	30	25
R	C4	T8	P4	Pz	P3	O1	Oz	O2
A	20	5	30	30	20	45	45	45
B	20	5	35	15	25	40	35	10
C	40	45	30	25	35	25	35	35

Tab. 6 Concordance rate of the EEG (Left)

L	Fp1	Fp2	F4	Fz	F3	T7	C3	Cz
A	45	25	20	25	20	30	20	5
B	50	35	30	30	30	35	35	30
C	40	20	35	30	20	30	20	20
L	C4	T8	P4	Pz	P3	O1	Oz	O2
A	30	35	35	35	30	30	30	30
B	20	25	50	30	40	25	25	30
C	20	45	40	30	30	25	25	25

The 20 trials of active state are to open and close the palm along to the sound from a metronome once a second and twice a second. Examinees do them both in the right and the left hands. We measure each task for 30 seconds and adopt 10 seconds data out of the 30 seconds data to be preprocessed.

4.2 Results

4.2.1 Frequency analysis

Tables 1-6 show the results of the concordance rates of the top frequency peak and the number of opening and closing times obtained by frequency analysis. In the tables R and L indicate the result of the right and the left hand, respectively. A, B and C represent examinees. Since opening and closing a hand is performed once or twice per second, the frequency of the operation is 1Hz or 2 Hz.

Tables 1 and 2 show the result of frequency analysis on the acceleration of fingertips. The column label shows the names of fingers. The average of the concordance rate is 87% both on the right hand and the left hand. In particular, the concordance rate of B is 100% in all fingers and both hands. On the other hand, the left hand of A and the right hand of C show higher concordance rates. In terms of fingers, the little fingers show the highest consistent rates.

Tables 3 and 4 show the result of frequency analysis on the forearm EMG (after RMS). The column label shows the electrode number. The averages of the concordance rates are 89% and 84% for the right hand and the left hand, respectively. In terms of examinees, the concordance rate of B is the highest in both arms. In terms of channels, channel 3 and 7 show higher concordance rates in the right hand. In the case of the left hand, channel 1 and 5 show higher ones. On the other hand, channel 8 shows the lowest rates in both arms.

Tables 5 and 6 show the results of EEG (after RMS). The column label shows electrode names. The averages of the concordance rates are 28% and 26% for the right hand and the left hand, respectively. In terms of examinees, the concordance rate of the right hand is the highest for C and the concordance rate of the left hand is the highest for B. In terms of channels, when examinees move the left hand, the right hemisphere shows higher concordance rates, and vice versa.

4.2.2 Cross-correlation coefficient analysis

We analyze the fingertips acceleration and the forearm EMG correlation. We have 480 sets of data (3 examinees × 2 stimulus × 2 hands ×5 accelerations × 8 EMGs). The cross-correlation coefficient average is 0.51. Then, we normalize the cross-correlation coefficients by trial to average them. Figure 5 shows averages of five fingers of all trials calculated for each channel. The upper figure shows the left arm results and the lower figure shows the right arm results. The more they are highly correlated, the darker the color is. The right has high correlations of channel 3 and 7 while the left has high correlations of channel 1 and 5. In addition, the right has low correlations of channel 2 and 8 while the left has low correlations of channel 6 and 8.

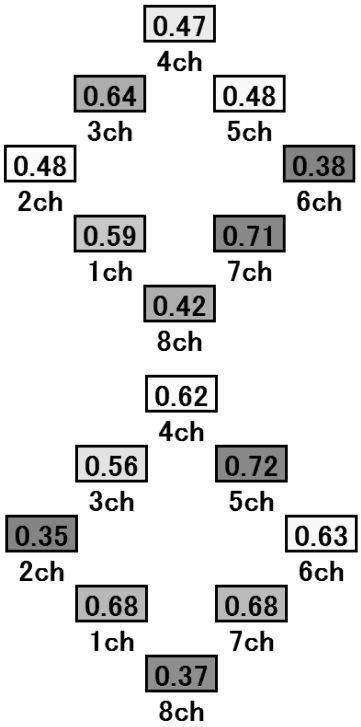


Fig.5 Acceleration and EMG correlation

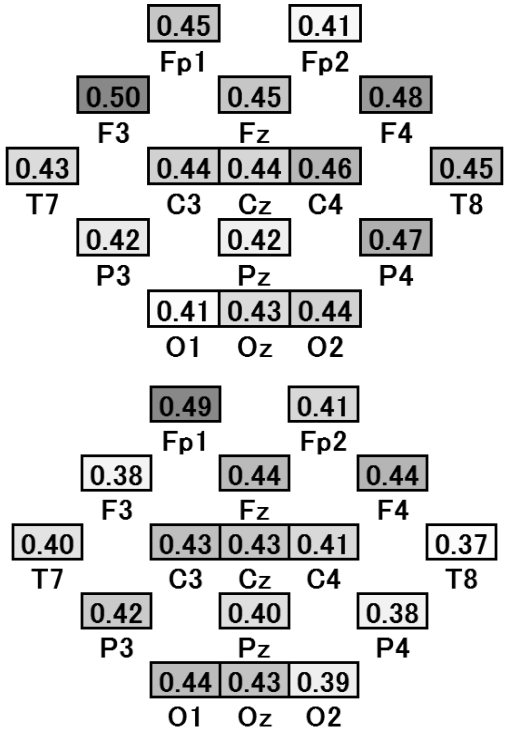


Fig.6 Acceleration and EEG correlation

Next, we analyze the fingertips acceleration and the EEG correlation. We have 960 sets of data (3 examinees × 2 stimulus × 2 hands × 5 accelerations ×16 EEGs). The cross-correlation coefficient average is 0.17. As in the fingertips acceleration and the forearm EMG correlation, they are normalized to calculate averages. Figure 6 shows averages of

Tab.7 Difference between pairs of symmetrical channels along the midline (Right)

	Fp1-Fp2	F3-F4	T7-T8	C3-C4	P3-P4	O1-O2
R	0.079	-0.063	0.026	0.026	0.039	0.050
L	0.039	0.015	-0.020	-0.021	-0.046	-0.033

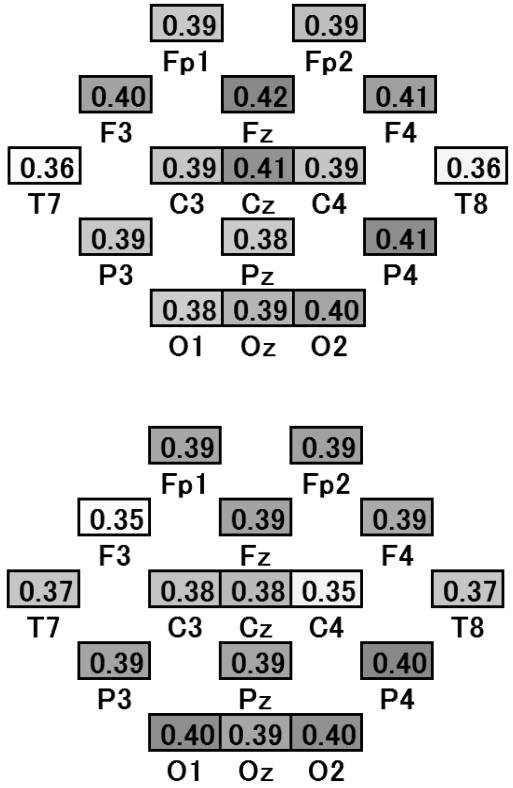


Figure 7 EMG and EEG correlation

Table 8 Difference between pairs of symmetrical channels along the midline (Left)

	Fp1-Fp2	F3-F4	T7-T8	C3-C4	P3-P4	O1-O2
R	-0.002	-0.038	0.000	0.022	-0.013	0.002
L	-0.005	-0.001	-0.001	-0.004	-0.028	-0.019

five fingers of all trials calculated for each channel. The upper figure shows the left arm results and the lower figure shows the right hand results. The more they are highly correlated, the darker the color is. Table 7 shows the differences between pairs of symmetrical channels along the midline. The right hand is advantageous in the case of positive values while the left hand is advantageous in the case of negative values.

Finally, we show the EMG – EEG correlation. We get 1, 536 sets of data (3Examinees × 2Stimulus × 2Hands × 8EMGs×16EEGs). The cross-correlation coefficient average is 0.16. Figure 7 are EMG 8 channels all trials means

calculated for each channel. The above figure is the left result, the lower figure is the right result. The more highly correlated color is dark. Table 8 is the difference between the symmetrical channels bordering the midline. Right is advantageous if a positive number, left is advantageous if a negative number.

4.3 Discussions

The results of frequency analysis on the fingertips acceleration show relatively good concordance rates. The acceleration sensor is suitable for capturing the motion of an object. However, there are individual differences in the concordance rates. Although the method of opening and closing a palm is the same, the momentum and the strength to hold the hand by examinee are different or the degree of holding the hand is different.

The results of frequency analysis on the forearm EMG show relatively good concordance rates, too. Especially, we find several channels with high concordance rates near the little finger in the palm side and the thumb in the back of the hand. It is explainable by the distributions of arm muscles and another reason is that the EMG sensors put at the places with more muscles on the forearm capture more muscular reaction. On the other hand, the concordance rates of channel 8 are not good because channel 8 is located on a bone and therefore the muscles are thin. In addition, the muscles corresponding to the channels in the left and the right hands show the same tendency if the channel numbers are symmetrically exchanged along the line between channel 4 and 8. The concordance rates in the left hand are a little bit lower than the right. This is because the left hand is not dominant so the amount of muscle is inferior.

The results of frequency analysis on the EEG show inferior concordance rates compared to the other two. We find the peak frequencies are almost half. It seems that some other activities are carried out at the same time. However it may be because of noises that are not completely removed at the preprocessing stage. We cannot explain the phenomena whether the frequency peak is involved in the palm opening and closing at this point. It requires more investigation.

5 Conclusions

In this paper, we study the correlation of proximity voluntary muscles and EEG. Specifically, we focus on the acceleration of fingertips and the EMG of forearms as the information of the proximity voluntary muscles. We simultaneously measured them as well as EEGs. The respective time-series data were corrected regarding to the positional deviation to be preprocessed. The analysis was performed as the frequency analysis and the cross-correlation coefficient analysis. As the results, we find that there are correlations between fingers, arms and a brain. In particular, the correlations of the acceleration/EMG and the acceleration/EEG seem effective. Also we find meaningful difference between the right and the left.

Our future work includes extensions of the experiment conditions to adopt other hand movements. We try to detect some EEG patterns using the information of fingertips acceleration and forearms EMG. It is expected that the efficient EGG pattern mining is applicable to prosthetic hands.

6 References

[1] Benabid AL, Pollak P, Gao D, Hoffmann D, Limousin P, Gay E, Payen I, Benazzouz A. "Chronic electrical stimulation of the ventralis intermedius nucleus of the thalamus as a treatment of movement disorders"; Journal of neurosurgery, Vol. 84, Issue 2, 203—214, Feb 1996.

[2] Choi, Kyuwan, Andrzej Cichocki. "Control of a wheelchair by motor imagery in real time"; In International Conference on Intelligent Data Engineering and Automated Learning, 330—337, Nov 2008.

[3] Penfield, Wilder; Rasmussen, Theodore. "The cerebral cortex of man; a clinical study of localization of function"; 1950.

[4] Guy Hotson, David P McMullen, Matthew S Fifer, Matthew S Johannes, Kapil D Katyal, Matthew P Para, Robert Armiger, William S Anderson, Nitish V Thakor, Brock A Wester, Nathan E Crone. "Individual finger control of a modular prosthetic limb using high-density electrocorticography in a human subject"; Journal of neural engineering, Vol. 13, Issue 2, 026017, Feb 2016.

[5] Shima Keisuke, Takata Daisuke, Bu Nan, Tsuji Toshio. "A Recurrent Probabilistic Neural Network with Dimensional Reduction and Its Application to Time Series EEG Discrmination"; Transactions of the Society of Instrument and Control Engineers, Vol. 48, Issue 4, 199—206, 2012.

[6] Quoc V Le. "Building high-level features using large scale unsupervised learning"; In 2013 IEEE international conference on acoustics, speech and signal processing, 8595—8598, May 2013.

[7] Giulia C Matrone, Christian Cipriani, Maria Chiara Carrozza, Giovanni Magenes. "Real-time myoelectric control of a multi-fingered hand prosthesis using principal components analysis"; Journal of neuroengineering and rehabilitation, Vol. 9.1, Jun 2012.

[8] Kasuya Masahiro, Ryu Kato, Hiroshi Yokoi. "Development of a Novel Post-Processing Algorithm for Myoelectric Pattern Classification"; Transactions of Japanese Society for Medical and Biological Engineering, Vol. 53, Issue 4, 217—224, Dec 2015.

[9] G Pfurtscheller. "Central beta rhythm during sensorimotor activities in man"; Electroencephalography and clinical neurophysiology, Vol.51, Issue 3, 253—264, Mar 1981.

[10] G Pfurtscheller, F.H. Lopes da Silva. "Event-related EEG/MEG synchronization and desynchronization:Basic principles"; Clinical Neurophysiology, Vol. 110, Issue 11, 1842—1857, Nov 1999

[11] Kato Tech Co., Ltd. "HapLog";
Available (accessed 2016-08-20),
from < http://www.keskato.co.jp/products/haplog.html >

[12] Thalmic Labs. "Myo"; Available (accessed 2016-08-20),
from < https://www.myo.com/ >

[13] Biosemi. "Biosemi EEG ECG EMG BSPM NEURO amplifier electrodes" , Available (accessed 2016-08-20)
from < http://www.biosemi.com>

Comparison of Feature Extraction Methods for Early-Modern Japanese Printed Character Recognition

Kazumi Kosaka, Kaori Fujimoto, Yu Ishikawa, Masami Takata, and Kazuki Joe,
Dept. of Advanced Information & Computer Sciences, Nara Women's University, Nara, Japan

Abstract *– In this paper, we compare feature extraction methods for early-modern Japanese printed character recognition. The national diet library in Japan provides a lot of early-modern (AD1868-1945) Japanese printed books in the web, but full-text search is essentially impossible. In order to perform advanced search in historical literatures, the extraction of texts from images is required. To solve this problem, we have already proposed a multi-font Kanji character recognition method using the PDC feature. For achieving more performance, we compare three feature extraction methods to analyze the recognition results.*

Keywords: early-modern Japanese printed books, multi-font Kanji character recognition method, PDC feature, weighted direction index histogram feature, the cellular feature

1. Introduction

The National Diet Library (NDL) [1][4][5][6] in Japan keeps about 350,000 books as well as 8,000 journals/magazines dating from the Meiji era to the first half of Showa era (AD1868-1945). The books cover a broad range including philosophies, industry, literatures, technologies, natural sciences, art, etc. Most of them have gone out of print and scholarly valuable materials in a study of early-modern. The NDL started a project called "The Digital Library from the Meiji Era" in 2002. In the project, early-modern Japanese printed books are optically 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 in good condition against loss or damage. Users can view the digital images whenever or wherever with Internet connection for free. "The Digital Library from the Meiji Era" will be closed at the end of May in 2016 to be integrated into "The National Diet Library Digital Collections"[3][1]. After integration, the books in the Digital Library will be available on "The National Diet Library Digital Collections" Web site. At the NDL Web site, user can search the materials by giving just book information, such as title, author, publisher, and publication year. Since the text of early-modern Japanese printed books is exhibited as picture images, full-text search is essentially impossible. In order to perform the full-text search, it is required to extract texts from images. As described above, the number of the target books is so enormous that text extraction by hand is impossible in cost. Since there was no existing research on early-modern Japanese printed character recognition, we decided to collaborate with the NDL to start the research project of automatic text extraction for "Digital Library from the Meiji Era". Indeed, when commercially available OCRs are applied to the image data, the recognition rates are too low to be practical. This is because any standard character font sets for early-modern Japanese printed books did not exist in those days. Thus, we have presented that a method of hand written Kanji character recognition [4][5][6] can be used for the recognition of early-modern Japanese printed characters. In early-modern Japanese printed books, it is reasonably inferred that a different typography is adopted by publisher. Even if within the same publisher, we have reported that typography differs by age [7]. For these reasons, we decided to use a method of hand written 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 [4][5][6]. In this method, the PDC feature is extracted from early-modern Japanese printed character images. The recognition rate using the PDC features is approximately 92% for 2634 types of Kanji characters [8]. In general the miss-recognitions depend on the number and the types of learning character samples. In the meanwhile we have never used any feature other than the PDC as the feature

[1] In June 2016, "The Digital Library from the Meiji Era" has ended and we can use "The National Diet Library Digital Collection".

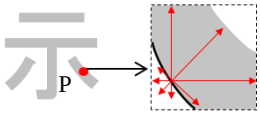


Fig. 1 : direction contributivity of dot P

Original 3-rd peripheral form

Fig. 2 : example of original image and 3-rd peripheral form

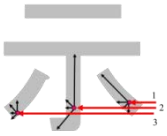


Fig.3 : scanning state toward a direction when n is 3

extraction methods for early-modern Japanese printed character recognition. Thus, in this paper, we try to use some other features. As high recognition accuracy methods for low quality printed characters, the weighted direction index histogram feature and the cellular feature as well as the PDC are well known [9]. They are useful features for hand written character recognition. In this paper, we compare recognition rates of the three feature extraction methods. Then, we analyze the miss-recognition results of each feature and exploit their tendency. The rest of the paper is organized as follows. In section 2, we briefly explain the three feature extraction methods. In section 3, we perform experiments using each feature described in section 2 and compare the results. Also, we analyze character types of miss-recognitions and exploit the tendency.

2. Features used for recongition

For our multi-fonts Japanese character recognition method for early-modern printed books we adopt the PDC feature as the feature extraction method. In this section, we briefly explain the weighted direction index histogram feature and the cellular feature in addition to the PDC feature.

2.1 The PDC Feature

The Peripheral Direction Contributivity (PDC) feature [4][5][6][10] is a feature focused on the direction of character strokes. The PDC feature, superior to straight strokes, is one of very efficient features for the recognition of Japanese characters, containing many straight strokes, written carefully in the standard style by hand.

The PDC feature reflects four statuses of character strokes: complexity, direction, connectivity and relative position. The complexity of character strokes is represented by their density. The relative position is represented by n-th peripheral form. The direction and the connectivity are represented by direction contributivity. The direction contributivity is

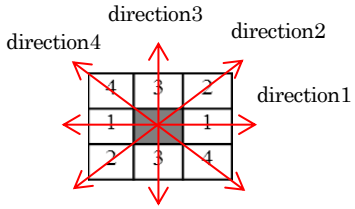


Fig.4 : target pixel and exact values for 4 directions

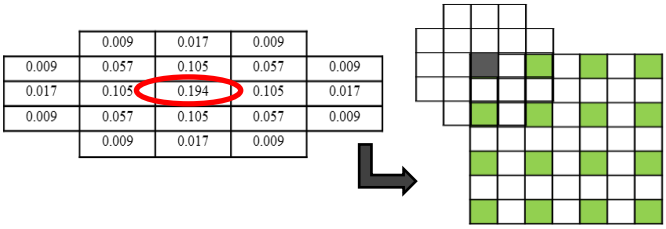


Fig.5 : weight coefficients of 2-dimensional Gaussian filter, the scope and the center position for integrating into 4 × 4 regions

given as a four dimensional vector, which is calculated by dots of character bit-map images. d_P, the direction contributivity of the dot P shown in Fig.1, 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

$$d_{mP} = \frac{l_m + l_{m+4}}{\sqrt{\sum_{j=1}^{4}(l_j + l_{j+4})^2}}$$

where l_j (j= 1,2,…,8) is the length of connected dots scanned in eight directions as shown in Fig.1. The n-th peripheral form is calculated as follows. A character bit-map image is scanned in eight directions (each 45 degree). A scan finds and passes over several strokes, the 1-st to the n-th strokes, of the character to plot the dots representing each crossing. The form drawn by the crossing dots is called the n-th peripheral form, where n is the number of crossing within a scan. Fig. 2 shows an example of an original image and the 3-rd peripheral form.

Direction contributivity values for each dot, where each eight direction of vertical, horizontal and diagonal scans meets the contour of the target character, are projected to the corresponding axis. Fig. 3 shows a scanning state toward a direction when n is 3. The range of projected axis is divided by 16 to calculate average values. Thus, the average values are used as a PDC feature vector. A PDC feature vector P_n consists of 512*n dimensions. For example, when n is 3, we get $P_n = 512 \times 3 = 1{,}536$ dimensions.

2.2 The Weighted Direction Index Histogram Feature

The weighted direction index histogram feature [11] is based on the directions along the contour lines of character strokes.

First, an input character image is smoothed in order to extract contour lines. Then, the contour line image is divided into 35 × 35 regions, where a 4 direction histogram is calculated in each region. The contour lines of the target pixel (S_i) are tracked counterclockwise so that the next 4 directions are determined at the pixel (S_{i+1}) next to the target pixel. Fig. 4 shows the target pixel (S_i) and the exact values for 4 directions.

The direction index of 35 × 35 regions is a histogram of these direction values.

Then, the direction index of 35 × 35 regions is integrated into 18 × 18regions using a 2-dimensional Gaussian filter. As the result, a 4 direction index is obtained in each 18 × 18 region after integration and the feature vector consists of 18 × 18 × 4 = 1,296 dimensions. This is the weighted direction index histogram feature. Fig. 5 shows weight coefficients of 2-dimensional Gaussian filter, and the scope and the center position for integrating into 4 × 4 regions. The weight coefficients are applied to integrate the direction indices of the target pixel and 20 circumferential pixels.

2.3 The Cellular Feature

The cellular feature [12] is based on the edge directions that indicate rapid changes of brightness of an image.

First, microscopic feature is calculated by the edge directions and lengths in a 3 × 3 local region at the target pixel and its eight neighbor pixels. The obtained edge directions are divided into eight directions.

Next, the feature of a cell space is calculated with integrating the microscopic features around 4 neighbor pixels. This is represented by the linear sum of the edge lengths in each cell by direction.

Thereafter, a fan-shaped area within ±45 degrees by direction is defined in order to extend the local regions by direction. The fan-shaped area has two patterns with the different number of cells included in the area. Fig. 6 shows a fan-shaped area by direction and inside cells are presented in green. Integrating the fan-shaped area, the area is enlarged similarly and monotonically. The integration is repeated until it hits against a stroke in the original image. The decision of stopping integration is obtained by the following formula.

$$\mathrm{G}(i_c, j_c) = f(2 \cdot i_c, 2 \cdot j_c) + f(2 \cdot i_c + 1, 2 \cdot j_c) + f(2 \cdot i_c, 2 \cdot j_c + 1) + f(2 \cdot i_c + 1, 2 \cdot j_c + 1)$$

where i_c *and* j_c are the position of each cell. When $\mathrm{G}(i_c, j_c)$ is greater than 2, (i_c, j_c) hits against a stroke of the original image and the integration is stopped.

The integration is carried out by direction using the following formula.

In the case of $\theta_c = 1,3,5,7$:

t=1:

$$\mathrm{D}(\theta_c, 1) = A(\theta_c, 0) + E(\theta_c, 0) + C(\theta_c, 0) + D(\theta_c, 0)$$

$t \geq 2$:

$$\mathrm{D}(\theta_c, 1) = \begin{cases} \mathrm{D}(\theta_c, t-1) \; if \; \mathrm{G}(i_c, j_c) \geq 2 \\ Min(17.0, Max(\mathrm{D}(\theta_c, t-1), \mathrm{A}(\theta_c, t-1) + \mathrm{C}(\theta_c, t-1) \\ \quad -\mathrm{B}(\theta_c, t-2) + \mathrm{E}(\theta_c, 0) + \mathrm{D}(\theta_c, 0))) \; : \; otherwise \end{cases}$$

In the case of $\theta_c = 2,4,6,8$:

t=1:

$$\mathrm{D}(\theta_c, 1) = A(\theta_c, 0) + C(\theta_c, 0) + D(\theta_c, 0)$$

$t \geq 2$:

a) $\boldsymbol{\theta = 1}$ b) $\boldsymbol{\theta = 2}$

Fig.6 : fan-shaped area by direction

$$\mathrm{D}(\theta_c, 1) = \begin{cases} \mathrm{D}(\theta_c, t-1) \; if \; \mathrm{G}(i_c, j_c) \geq 2 \\ Min(17.0, Max(\mathrm{D}(\theta_c, t-1), \mathrm{A}(\theta_c, t-1) + \mathrm{C}(\theta_c, t-1) \\ \quad -\mathrm{B}(\theta_c, t-2) + \mathrm{D}(\theta_c, 0))) \; : \; otherwise \end{cases}$$

Let the neighbor pixels of $\mathrm{D}(i_c, j_c)$ be (A), (B), (C), and (E). The internal information is represented as $A(\theta_c, t)$ where θ_c is a direction represented by a number from 1 to 8 and t represents the number of integrations to be executed. The Cellular Feature is determined using the features of the target cell and the cells in the fan-shaped area. This feature is updated when integration is executed. When the size of the cell space is 60×60, the feature of a Kanji is represented with 13 times integrations. When the size of the cell space is 148×148, the feature consists of 1,152 dimensions.

3 Experiments

In this section, we show the experimental results to compare the recognition rates with three different features, the PDC feature, the weighted direction index histogram feature and the cellular feature, as described in section 2.

3.1 Recognition Experiments

As the method of recognition process, we adopt the Support Vector Machine (SVM), which is one of the most efficient supervised machine learning methods [13]. We use 2,678 types of JIS level-1 Kanji, JIS level-2 Kanji and Hiragana in the experiments [6][14]. Each type consists of six different font, namely different publisher's images.

Five images are used as training data while one image data is used as test data. Six recognition experiments are performed by changing the test data.

In order to perform the experiments, we use a high-end computer with 96 Xeon (8core, 2.0GHz, L3 18MB, QPI6.4GT/sec) and 769GB (8GB × 96) memory and a PC with Core i7-4770K and 16GB memory. In this section, we refer the high-end computer and the PC to I and II, respectively.

Tab. 1 shows the recognition rates for the three features. Tab. 2 shows the platforms for the experiments and processing times for the learning.

In Tab.1, the recognition rate of the PDC feature achieves the highest accuracy, 86.8%. The recognition rates of the weighted direction index histogram feature and the cellular feature are

85.5% and 81.1%, respectively. In the meanwhile, Tab.2 indicates that the PDC feature takes the longest learning time.

Tab. 1 : recognition rates for the three features

Feature	Recognition rate (%, average of 6 times)
PDC feature	86.8
Weighted direction index histogram feature	85.5
Cellular feature	81.1

Tab.2 : platforms and processing times for the learning

Environment	Feature	Processing time(hour)
I	PDC feature	486
I	Weighted direction index histogram feature	431
II	PDC feature	428
II	Weighted direction index histogram feature	183
II	Cellular feature	174

The reason of the heavy computational cost is that the PDC feature requires the largest number of dimensions for its feature vector.

3.2 Analysis of miss-recognitions

We focus on miss-recognized characters in the experiments, and analyze the tendency of miss-recognitions. The character types of miss-recognitions are classified using the following conditions. In classifying miss-recognitions, we use radicals described in Appendix of this paper. Radicals used in this analysis are "hen", "tsukuri", "kanmuri", "ashi", "kamae", and "tare".

Also, the classifying conditions include "old-form" of Kanji when it is miss-recognized. The old-form refers to Kanji characters previously used. In 1946, the government determined a common set of Kanji, and the Kanji characters not-included in the common set are old-forms.

a) Hiragana
b) Characters with different hen and the same tsukuri
c) Characters with different tsukuri and the same hen
d) Characters with different contents (components inside tare, kanmuri, ashi and kamae) and the same tare
e) Characters with different contents and the same kanmuri
f) Characters with different contents and the same ashi
g) Characters with different contents and the same kamae
h) Strokes with different thickness
i) Blurred strokes
j) Characters with hardly understandable contents
k) old-forms

The difference of stroke thickness is shown in Fig.7, which frequently causes miss-recognitions. Similarly, Fig.8 shows examples of two Kanji characters with blurred strokes. Kanji characters in Fig.9 are very difficult to recognize because of inkblots and complexity of the Kanji structures.

Tab.3 : Miss-recognition Rates

Feature that incorrectly recognize	Rate(%)
A	26.8
B	9.5
C	14.0
D	17.8
E	6.1
F	7.1
G	18.7

Fig.7: difference of stroke thickness

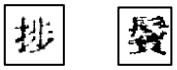


Fig. 8: examples of Kanji characters with blurred strokes

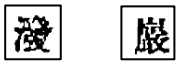


Fig.9: examples of inkbots and complexity of the Kanji structures

We also classify the miss-recognitions by which feature is used as shown below.

A) cellular feature
B) weighted direction index histogram feature
C) PDC feature
D) weighted direction index histogram feature and cellular feature
E) PDC feature and cellular feature
F) PDC feature and weighted direction index histogram feature
G) PDC feature, weighted direction index histogram feature and cellular feature

For each of A) to G), we analyze miss-recognitions based on the conditions a) to k). A) means miss-recognitions only with the cellular feature.

Tab. 3 indicates the miss-recognition rates of A) to G).

The cellular feature, which has the lowest recognition rate from Tab.2, often miss-recognizes in the case of A) from Tab.3. In the case of the weighted direction index histogram feature, the miss-recognition rate of D) is larger than B). In the case of the PDC feature, the rates of E) and F) are small, and major miss-recognitions are divided into two types of C) and G). Furthermore, G) has the second largest rate. Then, we

investigate the rates of classified miss-recognitions based on a) to k).

Tab.4: A) Cellular Feature

Classification	Rate (%)	(ex)Correct answer :Miss-recognition
a	21.3	ろ :る
b	6.8	柏 :拍
c	27.1	提 : 揚
d	1.7	屈 :届
f	0.8	忍 :思
h	18.6	柏 :桶
i	16.1	頷 : 鋲
j	4.2	盥 :瑪
k	3.4	満 : 滿

Tab.5: B) weighted direction index histogram feature

Classification	Rate (%)	(ex)Correct answer :Miss-recognition
c	14.3	狼 : 狭
g	42.9	間 : 問
h	14.3	狼 : 幅
i	28.6	狼 : 柏

Tab.6 : C) PDC feature

Classification	Rate (%)	(ex)Correct answer :Miss-recognition
b	2.7	蛸 : 悄
c	10.8	蛸 : 蜻
d	8.1	麼 : 磨
g	8.1	輿 : 與
h	2.7	猿 : 嬪
i	29.7	鬢 : 滝
j	18.9	難 : 楠
k	18.9	繩 : 繩

Tab.4 shows the classified results of A). The rates of a) and c) are over 20%. Namely, Hiragana and characters with different hen and the same tsukuri tend to be incorrectly recognized. Then, h) and i) are the large rates because of image quality problem. The cellular feature is affected by the difference of stroke thickness and blur. In the case of Hiragana, it is also affected by the connection of curve strokes. This is because the cellular feature focuses on edge directions.

Tab.5 shows the classified results of B). From Tab.5, the largest rate is g) followed by i). The weighted direction index histogram feature is apt to incorrectly recognize characters with different contents and the same kamae in addition to characters with blurred strokes. This is because the weighted direction index histogram feature divides the target region in four directions to integrate each direction feature.

The classified results of C) are shown in Tab.6. From i), j), and k) the miss-recognitions are caused by characters with blurred strokes and complicated contents or old-forms. The PDC feature is affected by blurred characters but unaffected by characters with different thickness of strokes. In the cause of miss-recognition by old-form, it is because old-forms are partly similar to the current versions of Kanji in general.

Tab.7:D)weighted direction index histogram feature and cellular feature

Classification	Rate (%)	(ex)Correct answer :Miss-recognition
a	5.7	ほ :は
b	5.7	拱 : 供
c	23.1	喧 : 噴
h	19.2	謝 : 諦
i	30.8	喧 : 瞳
j	9.6	潑 : 滝
k	5.7	福 : 福

Tab. 8 : E) PDC feature and cellular feature

Classification	Rate (%)	(ex)Correct answer :Miss-recognition
c	33.3	穂 :種
i	66.7	穂 :椽

Next, we analyze the results incorrectly recognized by two features to be classified in D) to F). Tab..7 shows the classified result of D). From the rates of h) and i) the difference of thickness and blur of strokes have great influence. The influence by image quality is considerable because the cellular feature and the weighted direction index histogram feature focus on the contour of character strokes or the edge directions. Tab..8 shows the result of E). The number of miss-recognized character types is small to be classified into two types of c) and i). Tab..9 shows the result of F). As well as E), the number of miss-recognized character types is small to be classified into three types of b), c) and k).

Finally, we analyze the results incorrectly recognized by all three features, namely, G). The result of G) is shown in Tab.10. It indicates that image quality problems such as h) and i), and the similar structures of k) are the major reason of miss-recognitions.

Thus, we observe that when the image quality is poor such as different thickness and blurs of strokes, the miss-recognition easily goes to any features. Especially, the weighted direction index histogram feature and the cellular feature are apt to be affected. Also, we observe that similar characters but partly different structures easily cause miss-recognition.

3.3 Character types and miss-recognitions

We analyze the character types that all three features incorrectly recognize namely G) in sub-section 3.2. Tab. 11 lists up the three times miss-recognized characters by all three features out of six recognition experiments. The columns of

Tab.11 include character ID number, correct characters, most frequently miss-characters, and the numbers of correct answers for each feature. Tab.12 lists up the four times miss-recognized characters by all three features. In this experiment, we do not have any characters that are miss-recognized more than four times by all three features.

Tab.9 : F) PDC feature and weighted direction index histogram feature

Classification	Rate (%)	(ex)Correct answer :Miss-recognition
b	16.7	緒 : 紺
c	33.3	緒 : 諸
k	50.0	戦 : 戦

Tab.10: G) PDC feature, weighted direction index histogram feature and cellular feature

Classification	Rate (%)	(ex)Correct answer :Miss-recognition
c	18.9	綱 : 網
e	10.1	塞 : 寒
g	5.8	圍 : 園
h	24.6	怜 : 拾
i	11.6	堇 : 憩
k	28.9	瀬 : 瀬

We try to classify the miss-recognized characters from Tab.11 and 12. The character ID 2, 3, 7, 8, 15 and 16 are incorrectly recognized with different tsukuri and the same hen. The character ID 5 is incorrectly recognized with different contents and the same tare. The characters ID 1, 6 and 13 are incorrectly recognized with different contents and the same kanmuri. The characters ID 4 and 9 are incorrectly recognized with different contents and the same kamae. In the causes of character ID 18, 19 and 20, and character ID 10, 11, 12, 14, 17 and 21, blurred strokes and old-forms are the main reason, respectively. On the other hand, the number of correct answers for each feature evenly varies for any characters.

Tab.13 shows the total numbers of correct answers and character types in the recognition result of all experiments performed 6 times using three features. The numbers of correct answers are more than 3. In other words, there are no characters that are not correctly recognized in experiments performed 6 times using three features.

We conclude that the recognition experiment proves a high recognition rate using three features. Indeed, no characters are incorrectly recognized using any of the three features. We will investigate a new recognition method to compensate three features to get more effective recognition results.

4. Conclusions

In this paper, we describe the comparison of three feature extraction methods for early-modern Japanese printed character recognition. In order to empirically compare them,

Tab.11 : three times miss-recognized characters by all three features out of six recognition experiments

Character ID number	Correct character	Miss-recognized character	PDC feature	Weighted direction index histogram feature	Cellular feature
1	塞	寒	3	3	3
2	縞	編	3	3	3
3	怜	恰	3	3	3
4	圍	園	3	3	2
5	瘤	痛	1	3	2
6	筧	算	3	3	1
7	爛	燗 / 烟	2	1	2
8	繻	紺	3	3	3
9	與	輿	1	2	3
10	諸	諸	1	2	2
11	諸	諸	1	0	3
12	嘱	囑	1	2	2
13	崇	祟	3	3	3
14	蔵	藏	2	3	3
15	網	綱	2	3	3
16	綱	網	2	1	1

Tab.12: four times miss-recognized characters by all three features

Character ID number	Correct character	Miss-recognized character	PDC feature	Weighted direction index histogram feature	Cellular feature
17	囑	嘱	1	2	1
18	懲	遮	1	2	1
19	歡	默	1	2	1
20	堇	掠	1	2	2
21	瀨	瀬	2	0	2

Tab.13: total numbers of correct answers and character types

The number of correct answers	The number of character types
18	579
16~17	883
11~15	1072
6~10	130
5	9
4	5
0~3	0

we use the PDC feature, the weighted direction index histogram feature and the cellular feature. It takes the longest computing time for the PDC feature that has the longest feature vector and the highest recognition rate.

Next, we analyze the miss-recognitions to be classified by feature and group of features. The highest rate of classified miss-recognitions is 26.8% in the case of cellular feature. Then,

the second highest rate is 18.7% in the case of all three features.

From the result of classified miss-recognitions, we find that the weighted direction index histogram feature and the cellular feature are apt to be affected by image quality such as the different thickness and the blur of strokes. The cause of miss-recognitions in all three features is invoked by characters that are similar and with partly different structures as well as the different thickness and the blurs of strokes. Furthermore, there are no characters incorrectly recognized in the experiments performed 6 times with three features.

We believe that the recognition rate will increase when we use the majority method where a character appearing most frequently in a recognition result is assumed as the final recognition result.

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 Prof. Ryuichi Oka of Aizu University for his guidance and Prof. Shinji Tsuruoka and Prof. Fumitaka Kimura of Mie University for his programs.

References

[1] National Diet Library :http://www.ndl.go.jp/

[2] Digital Library From the Meiji Era:http://kindai.ndl.go.jp/

[3] National Diet Library Digital Collections: http://dl.ndl.go.jp/

[4] Ishikawa.C,Ashida,N,Enomoto.Y,Takata,,Kimesawa,T.a nd Joe,K: Recognition of Multi-Fonts Character in Early-ModernPrintedBooks,PDPTA2009,Vol. Ⅱ,pp.728-7 34(2009).

[5] Fukuo,M.,Enomoto,Y,Yoshii,N.,Takata,M.,Kimesawa,T. and Joe,K.: Evaluation of the SVM based Multi-Fonts Kanji Character Recognition Method for Early-Modern Japanese Printed Books, PDPTA2011, Vol. Ⅱ, pp.727-732(2011).

[6] Kosaka,K., Awazu,T., Ishikawa,Y., Takata,M., Joe,K : An Effective and Interactive Training Data Collection Method for Early-Modern Japanese Printed Character Recognition, PDPTA2015, Vol. Ⅱ, pp.263-269 (2015. 07).

[7] Fukuo,M. , Takata, M., and Joe, K. :"The Kanji character recognition evalution for the modern book of the same publisher"(in Japanese), MPS, 26:1–6 (2012).

[8] Awazu,T., Kazumi,K., Takata,M., Joe,K. : A Multi-Fonts Kanji Character Recognition Method for Early-Modern Japanese Printed Books.(in Japanese), IPSJTOM, (in press)

[9] Miyamoto, K., Kumano, S., Sugimoto, K., Tamagawa,M., Eiho,S : A Recognition Method of Low Quality Printed Characters by Multiple Features (in Japanese), IEICE, Vol.J82-D,No4,pp.771-779(1999).

[10] Hagita,N.,Naito, S. and Masuda, I : Handprinted Chinese Characters Recognition by Peripheral Direction Contributivity Feature (in Japanese), IEICE, Vol.J66-D, 10, pp.1185-1192, 1983.

[11] Tsuruoka.S,Kurita,M., Harada,T., Kimura,F., Miyake,Y : Handwritten"KANJI"and"HIRAGANA"Character Recognition Using Weighted Direction Index Histogram Method, IEICE, Vol.J70-D,No.7,pp.1390–1397 (1987).

[12] Oka,R. : "Handwritten Chinese-Japanese Characters Recognition Using Celllular Features"(in Japanese), IEICE, Vol.J66-D, No.1, pp.17–24 (1983).

[13] Cristianini,N.andShawe-Taylor,J.: Support vector machine Introduction, Kyoritsu Publisher(2005).

[14] National Institute of Informatics : https ://www.jisc.go.jp/

[15] Soga,M.,Yusa,M.: BASIC KANJI, Taisyukan Publisher(1989).

[16] Matsuura,M. , Kamiduma,N., Handa, K. : Build Up Your KANJI SENSE , ASK Publisher(2009)

Appendix

Fig. : Structures of Kanji

According to [15][16], we introduce that Kanji are made up of sevral phonetic and semantic components.

In the kanji dictionary, Kanji are classified into seven types such as left, right, up and down components. These seven components are called radicals. The seven radicals are ***hen, tsukuri, kanmuri, ashi, tare, kamae,*** *and* ***nyou***.

As shown Fig.a, a radical (a component) on the left side of a Kanji is called ***hen***. Fig. b shows a right component called ***tsukuri.*** In the same manner, Fig.c and Fig.d show a top and a bottom components called ***kanmuri*** and ***ashi***, respectively. Some radicals which enclose a Kanji, either totally or partially are called ***kamae*** as shown in Fig.e. Fig. f and Fig.g show ***tare*** and ***nyou***. When a radical is composed to genenrate a complicated Kanji, the radical itself alos gener ates a simple and basic Kanji. For example, 亻(hen, person) 口(kamae, mouth), 土(hen or tare, earth), etc. are basic Kanji as well as composing difficult Kanji characters. 金a basic kanji meaning metal or gold is a radical for various metals, such as 銀"silver," 鉄 "iron", and 鉛 "lead."

Real-Time Super Resolution: FPGA Implementation for the ICBI Algorithm

Takashi Matsumoto, Arisa Yamamoto and Kazuki Joe
Dept. of Advanced Information & Computer Sciences,
Nara Women's University, Nara, Japan

Abstruct – *In recent years, display devices have been improved in resolution and data transmission speed through the Internet allows the use of FHD contents. In the meanwhile, we have huge amount of low resolution contents in both data and devices. To make efficient use of such low resolution contents on FHD, the super resolution imaging is one of the hottest research topics. In this paper, we present an FPGA implementation of ICBI in real time. We show how we reduce the repetition time of the ICBI smoothing phase for the FPGA implementation keeping the hardware resource amount reasonable small and real-time processing ability.*

Keywords: Super resolution imaging, FPGA, Real-time processing, FHD

1. Introduction

Recently, display devices have been improved in resolution and Full High Definition (FHD) of 1,920 by 1,080 becomes quite popular. In the meanwhile, now we have an enormous amount of low resolution contents across the reaches of the world. For example, we have small and existing inexpensive video such as surveillance camera or robot built-in camera, broadcast contents in a good old days, subminiature camera such as endscope or other inspection cameras. Of course, these low resolution contents do not make full use of FHD displays. These old contents and devices should be kept and if possible improved for the current use. To make full use of Toshiba and Sharp FHD displays, the old contents should be reconstructed as if they are high resolution. The question is how we improve old contents to high resolution.

Super resolution imaging is a technique that enhances the resolution of an imaging system. Namely, it improves low resolution contents to high resolution contents. In super resolution imaging, reconstruction process super resolution imaging [1], learning based super resolution imaging [2] and new edge directed interpolation [3] are widely known.

Reconstruction process super resolution imaging, known as Toshiba's Regza, requires multiple low resolution images taken in different conditions. Learning based super resolution imaging requires a lot of degraded images for learning to generate a database for image improving. New edge directed interpolation (NEDI) is superior to the others but needs large computational cost against real time process.

Recently, Iterative curvature-based interpolation (ICBI) is reported [4], which is as effective as NEDI with small amount of computational cost. In [4], it is also reported that an ICBI implementation using a CUDA based GPU (240cores) achieves a real-time up-scaling of 128x128 color images to 256-256 images in 12.3ms per frame. It means that real-time ICBI software implementations are possible using high-end platforms.

We are interested in implementing real-time ICBI with reasonable cost, say an FPGA implementation instead of the CUDA implementation.

The rest of the paper is organized as follows. In section 2, we briefly explain the ICBI. In section 3, the FPGA implementation issue is presented. In section 4, we report experiment results using a prototype FPGA implementation with hardware resource cost.

2. ICBI

The ICBI is an iterative algorithm to repeat two phases: the Fast Curvature-Based Interpolation (FCBI) algorithm [4] and a smoothing algorithm. We briefly explain the original algorithm to make the difference with our implementation version clear.

2.1 FCBI

The FCBI algorithm is the first process of ICBI. It interpolates a pixel by the pixel brightness values in the neighbor. First, it compares the curvatures toward both diagonal directions. The curvature is calculated by the second order derivatives. In this paper, when we want to generate the red pixel by interpolation in Fig.1, we first use the following expression [4].

ISBN: 1-60132-444-8, CSREA Press ©

$(2i-2, 2j+2)+(2i, 2j)+(2i+2, 2j-2)-3(2i, 2j+2)$
$-3(2i+2, 2j)+(2i, 2j+4)+(2i+2, 2j+2)+(2i+4, 2j)$

Namely, we sum the blue pixels and subtract weighted yellow pixels. After calculating another curvature, we compare the curvatures of both diagonal directions and assign to (2i+1, 2j+1) the average of the two neighbors in the diagonal direction where the curvature is lower. For example, when the curvature on the upper right diagonal is smaller, we calculate the average of the two blue pixels surrounded by a yellow ellipse to interpolate the red pixel in Fig.1. The other pixels are interpolated in the same way. When all the pixel interpolation toward both diagonal directions is completed, the next phase of a smoothing method described in the next subsection is applied. Then, we interpolate the pixels toward the horizontal and vertical directions as in the same manner as shown in Fig.2.

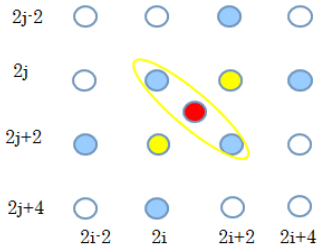


Figure 1 Curvature calculation for diagonal directions

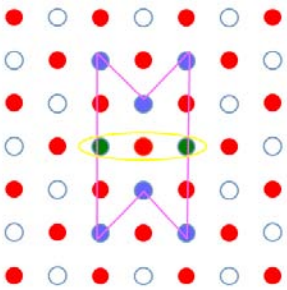


Figure 2 Curvature calculation for horizontal and vertical directions

2.2 Smoothing

ICBI consists of two parts: FCBI and a smoothing process. I explain the latter part, namely, smoothing. Since FCBI interpolate the red pixels with two neighbor pixels, the red pixels themselves are already smoothed. But the curvature is different from other curvatures and they are not smoothed from the wide range view. To smooth them, a sophisticated smoothing is needed as the past process of FCBI. The following expression represents the total summation of each curvature and it is to be minimized using a hill-climb method, where *a*~*e* and the red star are shown in Fig.3.

$|a-b|+|a-c|+|a-d|+|a-e|-\bigstar$

The star means that the previous curvature value is subtracted from the total summation so that the original image is not degraded. After the smoothing process toward diagonal directions, FCBI is applied for horisontal and vertical directions. Then another smoothing is applied toward horizontal and vertical directions.

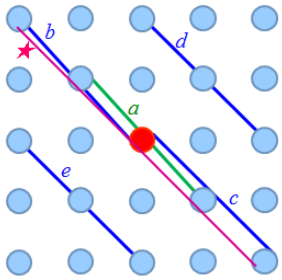


Figure 3 Smoothing process

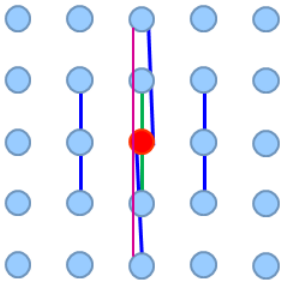


Figure 4 Smoothing for vertical direction

3 Hardware Implementation

As presented in [4], the ICBI algorithm has a problem of computation cost if it is processed just by software. Actually, it is impossible to use the ICBI with the processing speed of 30 frames per second. So we implement it by hardware for the real time process. Here we propose a hardware implementation method for ICBI. We use an FPGA for the hardware implementation. As the concrete platform, we use a ZedBoard [5].

3.1 Data

We explain input data. Since FCBI interpolates pixels based on their brightness values, we adopt YUV422 as the color information for input data. It is known that human's visual perception has better ability for brightness resolution rather than color information. Taking this ability into account, the format of

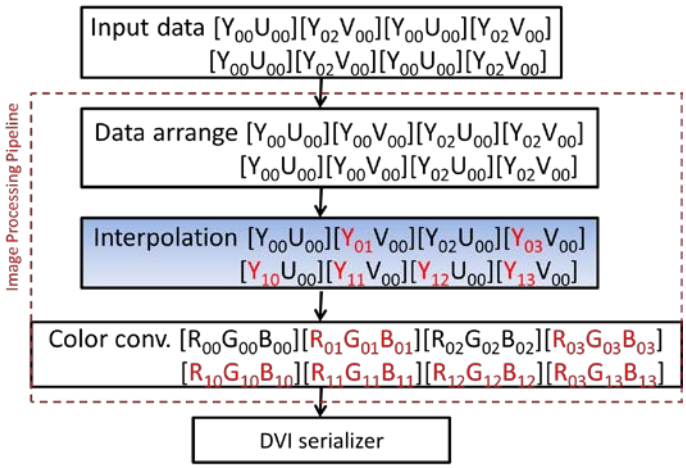


Figure 5

Table 1 Pipeline stage

1	pixel interpolation diagonal directions			
2	pixel interpolation horizontal-vertical directions	pixel interpolation diagonal directions		
3	diagonal and horizontal-vertical smoothing	pixel interpolation horizontal-vertical directions	pixel interpolation diagonal directions	
4	diagonal and horizontal-vertical smoothing	diagonal and horizontal-vertical smoothing	pixel interpolation horizontal-vertical directions	pixel interpolation diagonal directions

YUV422 is to use half of information bits for the brightness with reducing the color information, and the rest of half to put U, color difference for blue, and V, color difference for red, alternately.

In our proposed implementation, the flow of data is as follows. First, we take video data by a camera module OV7670 [6], and send the data to the frame buffer to synchronize the camera and the display. The camera module OV7670 is very cheap to generate 16bit VGA color information per pixel. The resolution of the display is 1,280 by 768. Since we develop a prototype first, we do not adopt FHD yet. Then, the pixel coordinate data is sent to the image processing pipeline that includes the interpolation pipeline. In the interpolation pipeline, we support nearest neighbor, bilinear, bicubic, FCBI and ICBI in this implementation. The processed data is sent to the display in real time. Figure 5 shows the flow of YUV422 data.

3.2 Pipeline

We do not explain the hardware implementation in detail in this paper except interpolation pipeline. The ICBI algorithm has the flow of pixel interpolation followed by smoothing toward diagonal directions and pixel interpolation followed by smoothing toward horizontal-vertical directions. Since the horizontal-vertical pixel interpolation cannot be executed until the diagonal pixel interpolation is completed, they must be executed in sequential.

In our interpolation implementation, diagonal interpolation is executed at the first clock. At the next clock, horizontal-vertical interpolation is executed with the next diagonal interpolation. At the third clock, diagonal and horizontal-vertical smoothing are executed with the next diagonal and horizontal-vertical interpolation. The smoothing process may be repeated to get more smoothed images. Table 1 shows the pipeline stages.

In our proposed ICBI implementation, the smoothing process occupies one stage pipeline, so it takes too much computation to be completed in a clock. So we use two clocks. The repeat of smoothing may degrade the resultant images. Namely, too many repeat times of smoothing generate noises on the resultant images. For example, when we repeat the smoothing four times, we find unexpected pixels close to edges. That means FCBI with four times smoothing may degrade the resultant images. So we try to enhance the effect by one smoothing instead of repeating it many times. In the smoothing process, we use a hill-climb method to minimize the summation of curvatures. We enhance the amount of change more than twice. We confirm this enhancement by software implementation to get enough results. We call this enhancement boost version of ICBI. In our hardware implementation, we use the boost version of ICBI.

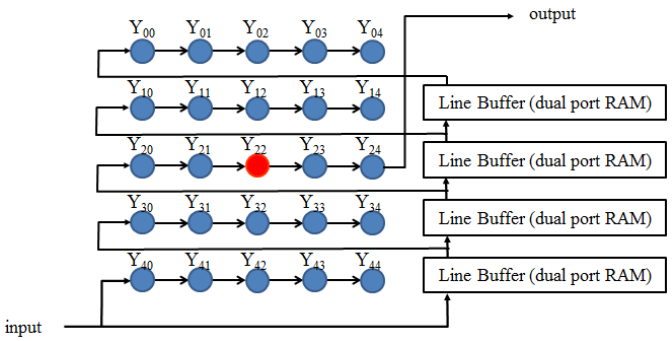


Figure 6 Basic structure of 5-line x 5-pixel pipeline

Figure 6 show the basic structure of 5-line x 5-pixel pipeline flow. Using this structure, the interpolation pipeline is implemented for ICBI, FCBI and other basic methods.

3.3 Validation of boost version

In the previous subsection, we propose the boost version of ICBI that enhances the step length for the hill-climb method to get enough results compared with repeating the smoothing phase. In this subsection, we show some preliminary experiment result to validate our boost version.

Figure 7 shows several images generated by software implementation to investigate the reduction of smoothing repeat times and the effect of boost version of ICBI. Comparing the normal FCBI with the boost version, we observe that the boost version generates a better image. As for the comparison between twenty times and four times smoothing, although twenty times smoothing generate higher resolution, we think we cannot detect the resolution difference by eye.

So we conclude that the boost version ICBI is efficient if the up-scaled images are used for human's watch. Quantitative evaluation is required for analytical use.

4. Experiment

Given the improvement and investigation, we implement the ICBI algorithm on an FPGA. In this implementation, we use 640x400x16bit BRAM in ZedBoard as frame buffer to achieve real-time 2x2 upscaling, namely 1,280x768 display format. As for color information, since we focus on brightness information for the ICBI algorithm, we do not translate the U and V information of YUV422 but just use Y (brightness) information. So the prototype implementation generates gray scale images.

Figure 8 shows the resultant images generated by our implementation. We take a movie by the camera module, and it is sent to the frame buffer, processed for interpolation including the boost version of ICBI pipeline, and displayed in real time (30 frames per second). The interpolation results are nearest neighbor, bilinear, FCBI and boost version of ICBI for up-left, up-right, down-left and down-right, respectively. Since the result image is optically taken by an ordinal camera, we cannot evaluate the results in detail. This is because we did not have a way to capture the resultant frame buffer data to external data storage in real-time yet.

(a) Original image 256×256pixel

(b) FCBI

(c) twenty times smoothing

(d) four times smoothing

(e) four times smoothing boost version

Figure 7 Validation of boost version ICBI by software interpolation

Table 2 Total HW resources

name	LUTs	FFs	BRAM (36kb)
XC7Z020 Total	53,200	106,400	140
Nearest Neighbor	1,197 (1,161)	814 (812)	120 (120)
Bilinear	1,892 (1,865)	971 (973)	121 (121)
Bicubic	5,667 (5,871)	1,506 (1,508)	120 (120)
FCBI	6,975 (7,007)	1,520 (1,525)	122 (122)
ICBI 4 times smoothing	22,716 (24,386)	4,529 (4,551)	126 (126)
ICBI 7 times smoothing	34,510 (41,825)	7487 (6716)	130 (129)
ICBI 9 times smoothing	---- (46,165)	---- (8,212)	---- (131)

We explain the resource issue of the prototype. We enumerate look up tables, Flip Flops and block RAM for the total amount of FPGA as shown in Tab.2. XC7Z020 is the FPGA installed in ZedBoard. In each row below XC7Z020, each number represents the amount of HW resource and each number in parentheses represents the expected amount of HW resource at compile time.

Table 3 HW resources for interpolation pipeline

name	LUTs	FFs	BRAM (36kb)
Nearest Neighbor	0 (0)	0 (0)	0 (0)
Bilinear	595 (604)	157 (161)	1 (1)
Bicubic	4,470 (4,710)	692 (696)	0 (0)
FCBI	5,778 (5,846)	706 (713)	2 (2)
ICBI 4 times smoothing	21,519(23,225)	3,715 (3,739)	6 (6)
ICBI 7 times smoothing	33,313 (40,664)	6,673 (5,904)	10 (9)
ICBI 9 times smoothing	---- (45,004)	---- (7,400)	---- (11)

As for the hardware resources for interpolation pipeline, Tab.3 shows the detail. In the ICBI, the resource FCBI uses is limited compared with ICBI smoothing multiple times. The more times of smoothing, the more amounts of hardware resources. Note that we plan to implement ICBI with 9 times smoothing but it turned out that the implementation exceeds the hardware limitation.

5. Conclusions

Since pixel interpolation by FCBI has relatively small computation cost, real time implementation is possible without huge amount of hardware resources. In the meanwhile, ICBI requires multiple times smoothing processes, so the pipeline must be designed with multiple stages to require huge amount of hardware resources. Improving the ICBI algorithm, we reduce the amount of resource to develop an FPGA implementation. The evaluation of real time process is confirmed by the prototype.

The current version of our ICBI prototype provides a standard enlargement factor of two. As explained in the introduction of the paper, one of the purposes of the super resolution imaging is to convert low resolution contents into FHD. However, the pixel number of low resolution existing cameras does not reach to FHD just by two times. The prototype provides just two times enlargement because of hardware resource. Our future work includes more enlargements with reducing the amount of hardware resource.

References

[1] Park, S.C., Park, M.K., Kang, M.G.: Super-resolution image reconstruction: a technical overview, IEEE Signal Processing Magazine. 20 (3): 21–36 (2003).

[2] Joshi, M.V., Chaudhuri S., Panuganti R. : A learning-based method for image super-resolution from zoomed observations, IEEE Trans Syst Man Cybern B Cybern. 35(3):527-37 (2005).

[3] Xin Li: New edge-directed interpolation, IEEE Trans on Image Processing. 10(10); 1521-1527 (2001).

[4] Giachetti, A., Asuni, N.: Real-Time Artifact-Free Image Upscaling, IEEE Trans on Image Processing. 20(10): 2760–2768 (2011).

[5] ZedBoard :<https://www.avnet.co.jp/kits/Xilinx/AES-Z7EV-7Z020-G-J.aspx> (2016/6/8 accessed)

[6] http://www.ovt.com/ (2016/6/8 accessed)

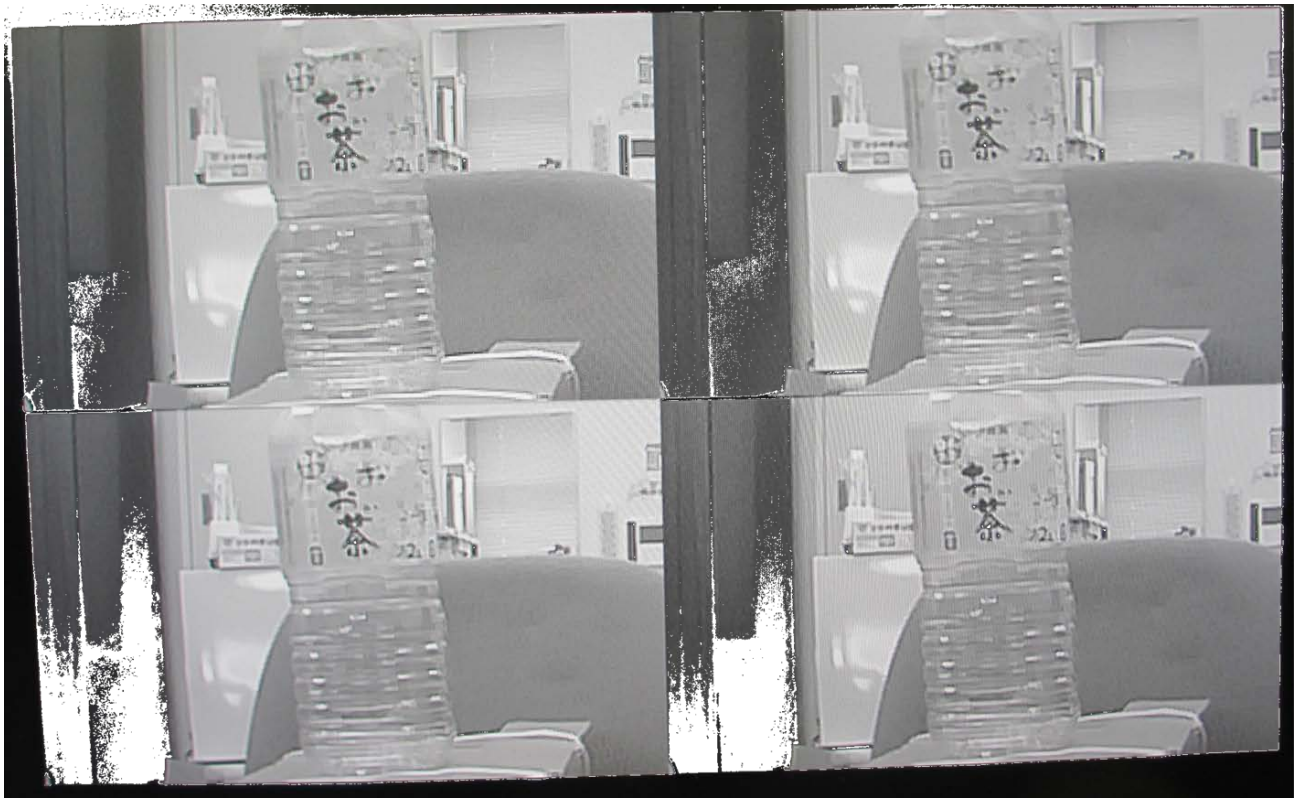


Figure 8 Validation of boost version

SESSION

LATE BREAKING PAPERS

Chair(s)

TBA

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©

Search Space Segmentation for Distributed Stochastic Algorithms

J. Mange[1], S. Pace[1], A. Dunn[1], and S. Enck[1]
[1]Computational Methods and System Behavior, US Army - TARDEC, Warren, MI, USA

Abstract— *For distributed algorithms, many different methods of problem- and search-space segmentation have been defined within various problem contexts. In particular, for distributed processing of certain types of stochastic algorithms, a natural question is whether this search space segmentation provides any practical advantages over running multiple instances of the algorithm over the entire space and combining the results appropriately. In this paper, we examine that question, first in the context of a simple algorithm for estimating the value of π, and then in the more complex context of a stochastic algorithm for calculating Ramsey number lower bounds. For each algorithm, we present comparison results of different levels of search space segmentation and the effect on performance.*

Keywords: Parallel processing, distributed computing, space segmentation, Ramsey theory

1. Introduction

Efficient segmentation of problem and search spaces for distributed processing is a complex issue that has been the subject of much research (e.g., [1], [5], [6]). For many optimization and search algorithms, this kind of space segmentation has been shown to provide enormous advantages in performance. However, for some types of stochastic algorithms, it is unclear whether these same advantages apply. Since these algorithms involve random searches and operations, it is not immediately apparent whether limiting each instance to operate within a defined sub-space will yield better results than allowing each to operate within the full space and combining the results in a way appropriate to the problem context (e.g., averaging).

Ultimately, we applied a search space segmentation method to a stochastic algorithm for the calculation of Ramsey number lower bounds, which will be explained in detail in a later section. However, to introduce the concepts more clearly, we first apply it to a basic algorithm for estimating the value of π.

2. Estimation of π

As a trivial example of the space segmentation concept, we use a simple algorithm for estimating the value of π. Since π is defined as the ratio of a circle's area to the square of its radius, one can use the Monte Carlo approach of randomly generating a number of points in a unit square to estimate π. As the number of generated points increases, the ratio of the number that fall within a circle inscribed in the unit square to the total number approaches the ratio of the area of the circle to the area of the square, which is:

$$\frac{\text{area of circle}}{\text{area of square}} = \frac{\pi \cdot \frac{1}{2}^2}{1}$$

This ratio can then be used to estimate the value of π. This algorithm is often used as an introduction to Monte Carlo algorithms, although its practical application is limited, because it converges much more slowly than many other algorithms for estimating π.

Figure 1 shows the average relative error for this algorithm as the number of generated points increases.

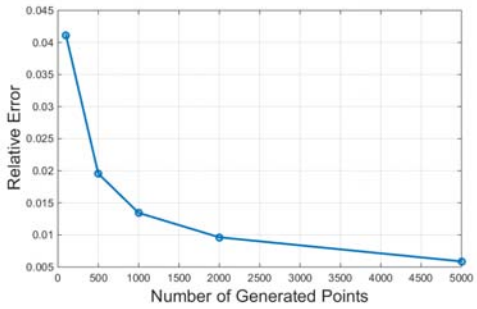


Fig. 1: Estimation of π.

2.1 Search Space Segmentation

Consider now that four processors are available for a distributed version of this algorithm. We considered three alternative methods of segmenting the search space (in this case, the unit square) for this problem:

- **Full segmentation** - All 4 processors operate on a separate quarter unit square, results added
- **Partial segmentation** - 2 processors each operate on a half unit square, results averaged between those 2, then added
- **No segmentation** - All 4 processors operate on the full unit square, results averaged

This has the effect of testing no segmentation of the search space, a partial segmentation, and full segmentation, respectively. See figures 4-2 for a representation of the search space for each.

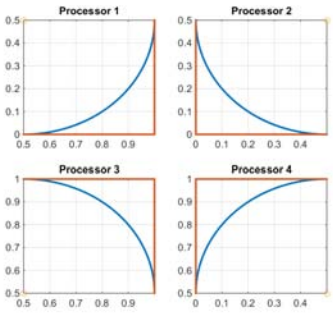


Fig. 2: Full Segmentation.

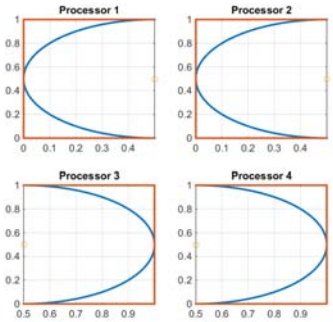


Fig. 3: Partial Segmentation.

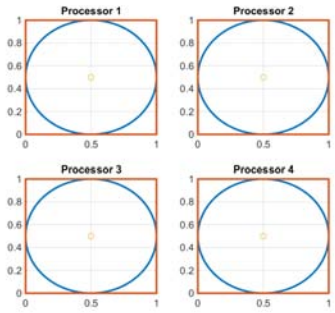


Fig. 4: No Segmentation.

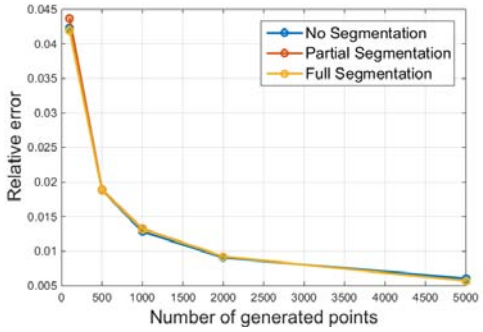


Fig. 5: Segmentation Results on Estimating π.

2.2 Analysis

As one might expect with this simple algorithm, there was essentially no difference between the three different methods of segmentation. Figure 5 shows a graph of the results of the methods, and one can see that all produced almost identical results. However, this is only true because of the simplicity of this example problem; with more complex algorithms, different methods of segmentation are not all equivalent, as will be seen in the follow sections.

3. Ramsey Numbers

The Ramsey numbers are a set of numbers related to combinatorics and graph theory [4]. These numbers have a fairly straightforward definition, yet their exact values are difficult to determine. Stated in graph theory terms, a Ramsey number $r = R(x_1, x_2, ..., x_k)$ is defined as the smallest number r such that any edge coloring of a complete graph of order r with k colors must contain a complete subgraph of color i on x_i vertices, for some $1 \leq i \leq k$.

Some general proofs exist for lower and upper bounds for the Ramsey numbers, but these bounds are often very loose. Establishing exact values for these numbers has proven extremely difficult, as evidenced by the fact that today, over 80 years after Ramsey theory was introduced, only a few non-trivial Ramsey number values are known [9]. These values are:

$$R(3,3) = 6 \quad R(3,4) = 9 \quad R(3,5) = 14$$
$$R(3,6) = 18 \quad R(3,7) = 23 \quad R(3,8) = 28$$
$$R(3,9) = 36 \quad R(4,4) = 18 \quad R(4,5) = 25$$
$$R(3,3,3) = 17$$

Other work has been done on tightening the general bounds for specific other Ramsey numbers or classes thereof, but currently these 10 values are the only known exact values.

3.1 Computation of Lower Bounds

While historically most of both the bounds and actual values for the Ramsey numbers have come from mathematical proofs or constructions, it is now increasingly feasible to use computer algorithms to attempt to generate colorings that establish new lower bounds. Since the definition of a Ramsey number is the smallest number such that a complete graph of that order must contain one of the subgraphs in question, lower bounds can be proven simply by providing an edge coloring of a complete graph of the desired order that does not contain any of these subgraphs. The stochastic algorithm we used in this paper employs this approach – it utilizes a randomized search algorithm to find a desired edge coloring, in order to provide a lower bound for the Ramsey number in question.

3.2 Stochastic Search Algorithm

The goal of the stochastic search algorithm described here is to find the lower bound for a specific Ramsey number by finding a valid edge coloring of the graph from the definition of a Ramsey number that does not violate the conditions specified. This is, in essence, a search problem with a very large search space for all but the smallest Ramsey numbers.

The basic function of the algorithm is to first construct a random graph coloring for a graph of the appropriate size with the appropriate number of colors, then iteratively attempt to "improve" the coloring by replacing edge colors with others that reduce the total number of subgraphs that violate the conditions. After a set interval, if the algorithm has failed to improve the coloring, it is replaced by a new random coloring and the process starts again.

The number of colorings for a graph of order n with k colors is given by:

$$k^{\frac{n\cdot(n-1)}{2}}$$

This number quickly grows large as n increases, and it becomes infeasible to search even a significant portion of the total search space on a single processor. For this reason, distributed computing is an attractive option. However, it is not clear whether segmenting the search space over multiple processors would yield better performance than simply allowing the same randomized algorithm to execute multiple times over the entire search space. In either case, different portions of the search space would be explored – in the first, by explicitly limiting the space each processor can investigate, and in the second, by allowing the randomness of the algorithm to determine which areas to explore. In the following section, we examine how to efficiently segment the search space for this problem.

3.3 Search Space Segmentation

In order to construct an edge coloring with k colors, each edge can be one of the k options. Therefore, a straightforward efficient search space segmentation involves assigning a specific color of k to each of n edges, using $p = k^n$ total processors. Some care must be given to ensure that the assigned edges do not themselves form one of the subgraphs that violate the constraints of the problem, but with a selection of specific edges that do not form any subgraphs of the order of any of the constraints, this problem can be avoided.

This is the segmentation approach we adopt for the testing of different search space segmentation methods. As in the example of estimating the value of π, we examine three different methods:

- **Full segmentation** - each processor is assigned a mutually exclusive area of the search space
- **Partial segmentation** - Half of the processors are assigned mutually exclusive areas of the search space, the other half are assigned the same spaces as the first half
- **No segmentation** - the same stochastic algorithm is run on each processor over the entire search space

4. Experimental Design

In order to test each of the segmentation methods defined previously, for several specific known Ramsey numbers, we run the stochastic search algorithm for a graph of order one less than the value of the Ramsey number – that is, the largest graph for which a valid coloring exists. As a meaningful measure of performance, we choose to count the number of times the most expensive operation involved is executed, namely the operation that counts the number of subgraphs that violate the constraints. This is roughly equivalent to an objective function for an optimization problem (and indeed has been formulated as such, see [2], [7]).

This count is then compared for different numbers of processors, for each of the three segmentation methods. In order to reduce the impact of the randomness involved with the algorithm on the performance results, the same tests were run 1000 times, and the average subgraph violation counts were averaged over these runs. The results of these tests are presented in the following section.

5. Results

Table 1 lists the results for each segmentation method on the stochastic algorithm for calculating the largest lower bound for the Ramsey numbers $R(3,3)$, $R(3,4)$, and $R(3,5)$, respectively. Figures 6, 7, and 8 show this same performance graphically for each of the Ramsey numbers examined. As can be seen, the choice of segmentation methods significantly affected the performance of this algorithm, with the full search space segmentation strategy exhibiting the best performance across all problem instances and all levels of distribution aside from a single processor.

Table 1: Ramsey Number algorithm segmentation results.

Ramsey Number	Number of Processors	No Segmentation	Partial Segmentation	Full Segmentation
$R(3,3)$	1	6.7318	6.8580	6.7576
$R(3,3)$	2	6.7972	5.8916	6.7190
$R(3,3)$	4	6.8032	5.0784	5.8745
$R(3,3)$	8	6.7604	4.3268	5.0368
$R(3,4)$	1	35.0736	35.1856	35.4264
$R(3,4)$	2	35.7344	33.2012	34.7824
$R(3,4)$	4	35.1464	31.4112	32.7638
$R(3,4)$	8	36.0648	28.1150	30.5584
$R(3,5)$	1	324.9200	322.1200	323.0000
$R(3,5)$	2	315.5600	296.1300	307.4480
$R(3,5)$	4	295.2800	284.7200	294.9960
$R(3,5)$	8	301.5200	280.7500	294.1440

As in the estimation of π example, there is little variation among the performance for the "no segmentation" method,

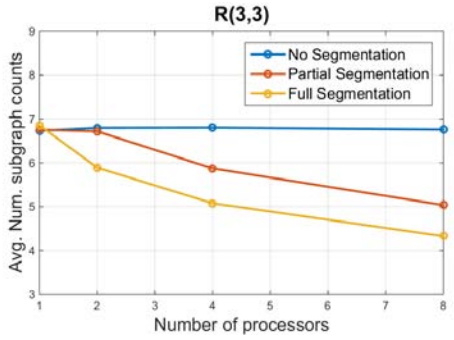


Fig. 6: Segmentation method comparison on $R(3,3)$.

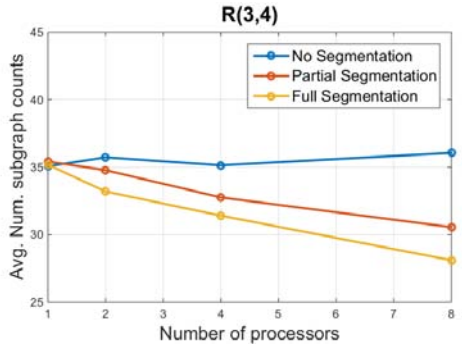


Fig. 7: Segmentation method comparison on $R(3,4)$.

since on average, the same number of subgraph violation counts is required regardless of the number of processors executing the same stochastic algorithm.

6. Conclusions

From these results, we can conclude that full segmentation of the search space is indeed the best strategy for the more complex stochastic algorithm to calculate Ramsey number lower bounds. These results are rather problem-specific, and the same choice may not hold for all such stochastic algorithms, since many elements of the problem context can affect the distributed performance of search and optimization algorithms.

It is also significant to note that there is something of a "diminishing returns" effect as the problem size increases. Since the search space for Ramsey numbers grows exponentially with the order of the graph being examined, the segmentation of this search space yields lower performance improvements as the order increases. Nevertheless, a full, efficient segmentation of the search space still yields the best performance of this algorithm.

Although we demonstrated the results of this approach on relatively small, known instances of Ramsey numbers, the same strategy is likely to yield valuable performance gains as we and other researchers push forward to find new bounds and new values for the as-yet-unknown Ramsey

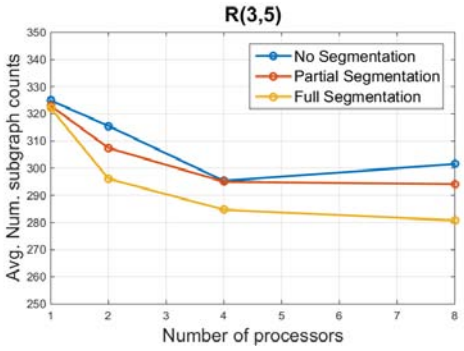


Fig. 8: Segmentation method comparison on $R(3,5)$.

numbers, using distributed computing resources to make possible searches of increasingly large search spaces.

References

[1] Bentley, Jon Louis. "Multidimensional divide-and-conquer." Communications of the ACM 23.4 (1980): 214-229.

[2] Exoo, Geoffrey. "On the Ramsey Number $R(4,6)$." the electronic journal of combinatorics 19.1 (2012): P66.

[3] Gourdain, N., et al. "High performance parallel computing of flows in complex geometries: I. methods." Computational Science and Discovery 2.1 (2009): 015003.

[4] Graham, Ronald L., Bruce L. Rothschild, and Joel H. Spencer. Ramsey theory. Vol. 20. John Wiley and Sons, 1990.

[5] Happ, P. N., et al. "Multiresolution segmentation: a parallel approach for high resolution image segmentation in multicore architectures." The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 38.4 (2010).

[6] Korting, Thales Sehn, Emiliano Ferreira Castejon, and Leila Maria Garcia Fonseca. "Divide And Segment - An Alternative For Parallel Segmentation." GeoInfo. 2011.

[7] Mange, Jeremy, and Andrew Dunn. "Luus-Jaakola Optimization Procedure for Ramsey Number Lower Bounds." International Journal of Mathematics and Computer Science 10.1 (2015): 57-68.

[8] Melanz, Daniel J. On the Validation and Applications of a Parallel Flexible Multi-body Dynamics Implementation. Diss. University of Wisconsin-Madison, 2012.

[9] Radziszowski, Stanislaw P. "Small ramsey numbers." Electron. J. Combin 1.7 (1994).

[10] Sammeth, Michael, Burkhard Morgenstern, and Jens Stoye. "Divide-and-conquer multiple alignment with segment-based constraints." Bioinformatics 19.suppl 2 (2003): 189-195.

[11] Seidl, Andrew A. Parallel Implementation of a Vehicle-Tire-Terrain Interaction Model. Diss. University of Wisconsin-Madison, 2012.

[12] Yang, Weitao, and Tai-Sung Lee. "A density-matrix divide-and-conquer approach for electronic structure calculations of large molecules." The Journal of chemical physics 103.13 (1995): 5674-5678.

An Optimized Approach for ETL in Real-Time Data Warehouses based on (m, k)-firm Constraints

Issam Hamdi[1], **Emna Bouazizi**[2], **and Jamel Feki**[2]
[1]Miracl Laboratory, University of Sfax, BP. 1088, Sfax 3018, Tunisia
[2]Faculty of Computing and IT, University of Jeddah, Jeddah, Saudi Arabia

Abstract - *Nowadays the update frequency for traditional data warehouses cannot meet the objectives of real-time data analysis relying on data freshness. To alleviate this problem, the real-time data warehouse (RTDW) technology has emerged. A RTDW allows decision makers to access and analyze very recent data as fast as possible in order to support real-time decision processes. The RTDW must often deal with transient usage charges, due to the unpredictability of access to data. In this paper, we propose an architecture called DETL-(m, k)-firm-RTDW architecture (Decentralized Extract-Transform-Load approach based on (m, k)-Firm constraint for Real-Time Data Warehouse). This architecture deals with diversity and disparities in data source systems to reduce the time for ETL and it has threefold objectives: i) guarantee the data freshness, and ii) enhance the deadline miss ratio even in the presence of conflicts and unpredictable workloads. DETL-(m, k)-firm-RTDW architecture is designed for complex analytical queries that are very costly due to several join operations. Despite the complexity of their queries, decision makers want their requests to be processed rapidly. Therefore, we focus on optimization techniques and more precisely data partitioning optimization and materialized views in order to guarantee the user access to the required information with reasonable response time, as short as possible. Finally, we evaluate our feedback control scheduling architecture which considers both materialized views and data fragmentation using the TPC-DS [18] benchmark; the preliminary results are quite promising.*

Keywords: RTDW; ETL; Quality of Service; Data sources partitioning; (m, k) -firm constraints.

1 Introduction

The traditional data warehouse uses historical data to provide strategic decision making and product management for corporate decision makers. However, enterprise hopes data warehouse to provide real-time strategic decision making, such as real-time marketing. For example, financial institutions require accurate and up-to-date analytical reports regarding available prices and stocks. Too much latency in the data may cause the organization to lose a large amount of money.

The real-time data warehouse is a new data warehouse architecture which is based on the traditional data warehouse development. Real-time means detect and capture the changed data from the data source in time, and load changed data in time when the data in the data source are changed by transaction processing, in order to meet user's real-time analysis and decision-making requirements.

To accumulate data in one place, we need an extract, transform and load (ETL) software. But, such software takes enormous time for this purpose. This is exactly the problem we are dealing with in this paper.

The remaining of the paper is organized as follows. In Section 2, we present pertinent works related to QoS management in RTDW, real-time ETL and the (m, k)-firm constraints in real-time systems. Section 3 describes our proposed approach called DETL-(m, k)-firm-RTDW to manage ETL process in RTDW with QoS guarantees for RTDW. Our work is evaluated according to a set of simulation results in Section 4. We conclude the paper, in Section 5, by briefly discussing our approach and by presenting our future work.

2 Related Works

In this section, we describe our real-time data warehouse model by presenting data and transaction models, and defining the basic performance metric we consider. Then, we mention some works related to real-time ETL. We finish by giving an overview of the previous work in which the (m,k)-firm approach is used for the QoS enhancement.

2.1 Management of QoS in real-time data warehouse

In traditional data warehouses, updates are typically applied during downtimes, e.g., every night. In contrast, a real-time data warehouse attempts to load new data as they arrive. In this paper, we limit ourselves to a RTDW that receives write-only update transactions in order to reflect the state of the real and read-only query transactions.

The data in this model are stored in relational databases called ROLAP (Relational Online Analytical Processes) using the star schema. Therefore, the RTDW maintains two types of tables: base tables that are sourced directly, and derived tables corresponding to materialized views that are recomputed results of SQL queries.

1) **Transaction Model.**

Transactions are classified into two classes: update transactions and user transactions as explained below:

- User OLAP (On-line Analytical Processing) transactions, representing user requests, arrive randomly (not periodically) and can only read data. In order to allow users to express their real needs, each OLAP transaction is provided with two parameters **[13]**, acceptable response time delay Δt_{rl} and acceptable result staleness Δs, when it is submitted so that it can satisfy the requirements of users well. We use the absolute deadline (AD) to determine the priorities of OLAP transactions and solve the scheduling problem between short and long queries with a deadline.

$$AD(q_i) = t_a(q_i) + \Delta t_{rl}(q_i) + t_e(q_i) \quad (1)$$

  Where $t_a(q_i)$ is the arrival time of the transaction q_i, $t_e(q_i)$ is the execution time of q_i and $\Delta t_{rl}(q_i)$ is the acceptable response time delay.
- Update transactions update the values of real-time data (sensor data) in order to reflect the real world status. They are executed periodically and have only to write new sensor data. Each update transaction loads new data into a table. So tables are updated repeatedly over time. If the period is unknown or unpredictable, we let the user choose a period when the warehouse should check for new data. For each update transaction T_i, the deadline of T_i is estimated to be (r_i+P_i) where r_i is its release time and P_i is its period **[13]**.

2) **The quality of service in RTDW.**

The QoS can be seen as a metric that permits to measure the overall system performance. Indeed QoS is a collective measure of the service level provided to the customer. It is characterized by different performance criteria that include basic availability, error rate, response time and the rate of successful transactions before deadlines.

In **[13]**, authors introduce the main performance metrics for QoS management in real-time data warehouses. Indeed, the QoS introduces two concepts that they are defined as follows:

- *Quality of Transactions* (QoT): denotes the ratio of the number of OLAP transactions that miss their deadlines to the number of OLAP transactions that have already been executed.

$$DMR = \frac{|Q_{dm}|}{|Q_e|} \quad (2)$$

  Where, Q_{dm} is the set of OLAP transactions that miss their deadlines, Q_e represents the set of OLAP transactions that have already been executed, $|Q_{dm}|$ is the number of Q_{dm} and $|Q_e|$ is the number of Q_e.

- *Quality of Data* (QoD): is the ratio of the number of queries for which the result staleness is unacceptable to the number of queries that have already been executed.

$$URSR = \frac{|Q_{urs}|}{|Q_e|} \quad (3)$$

  Where, Q_{urs} is the set of OLAP transactions of which the result staleness is unacceptable, Q_e represents the set of OLAP transactions that have already been executed, $|Q_{urs}|$ is the number of Q_{urs} and $|Q_e|$ is the number of Q_e.

The staleness is computed only from those unapplied updates that are already in the system at the time the user runs the query; this is because the user does not expect to see "future results" **[16]**. In RTDW, the data staleness is calculated according to the partition unit. The staleness of a partition p_i is defined as the difference between the maximum arrival timestamp of each unapplied update on p_i and the freshness of p_i **[13]**:

$$S(p_i) = \max(t_a(u_j), p_i) - F(p_i) \quad (4)$$

Where $F(p_i)$ is the freshness of the partition pi as the maximum timestamp of all records in p_i and $\max(t_a(u_j), p_i)$ is the maximum arrival timestamp of each unapplied update on p_i. Because a query is only concerned about the update tasks that arrive before it, the staleness of a partition p_i related to a query q is changed as follows:

$$S(p_i, q) = \max_{t_a(u_j) \le t_a(p_i)} (t_a(u_j), p_i) - F(p_i) \quad (5)$$

For a query task q, the staleness S(q) is the maximum staleness of all partitions p_q that affect query results:

$$S(q) = \max_{p_i \in p_q} (S(p_i, q)) \quad (6)$$

Nowadays, few researching works focus on the QoS management in RTDWs which allows users to express their real needs and to control the transient overshoot of RTDW.

The authors in **[16]** proposed a workload scheduling WINE in RTDW. The algorithm is based on partitions of data warehouses and allows users to specify the Quality of Service (QoS) and Quality of Data (QoD) of queries. But it does not monitor the system resources and the running status of the update and query queues.

In **[5]**, Issam et al. proposed an architecture using local feedback controller. This architecture, called FCSA-RTDW, on which we base our work, aims an efficient management of transactions workload fluctuations to guarantee the data freshness, enhance the deadline miss ratio even in the presence of conflicts and therefore it enhances the QoS. But, FCSA-RTDW architecture has a main lack: it does not deal with ETL process management.

2.2 Real-time ETL

The ETL process extracts the data from source systems, transforms the data according to business rules, and loads the results into the target data warehouse. In traditional ETL tools, loading is done periodically during the downtime and

during this time no one can access the data in the data warehouse. In today's fast-paced, companies must be able to quickly integrate vast amounts of data from disparate systems.

In this context, the authors in **[3]** proposed an E-LT (Extraction, Loading, Transformation) as a newer approach to populating data warehouses that moves the data transformation step to the target data warehouse, then it changes the order of operations to extract the data from the source tables, to load the tables into the destination server, and then to transform the data in the target data warehouse. However, this approach imposes a significant overhead in RTDW

In **[17], [19]** authors discuss another technique for data warehouse refreshment. This technique differs from conventional data warehouse refresh technology in which data warehouse is updated in an off-line fashion. But there is still a gap that is how to resolve bursts of streams due to the unpredictability of incoming data streams. Therefore, an efficient technique is desirable to balance the load of incoming data streams.

Another approach called External Real-time Data Cache (RTDC) **[12]** stores real time data outside the data warehouse and thus it does not affect traditional data warehouse resources. The function of RTDC is to load the real time data into database from source systems. With this approach, there is no additional load on the data warehouse as the real time data lie on separate cache database.

The downside of using a RTDC solution, with or without just-in-time data merging, is that it involves an additional database that needs to be installed and maintained. Also, there is additional work required to configure the applications that need to access the real-time data **[12].**

In **[9],** the authors propose a triggering and scheduling approach for the real-time ETL which supports triggering the ETL task according to the integration rules (cf. Fig.1), and balances the execution of updates and queries dynamically. But this approach needs to invade the business system by a set of triggers to capture real-time changed data that will bring a great load to business system.

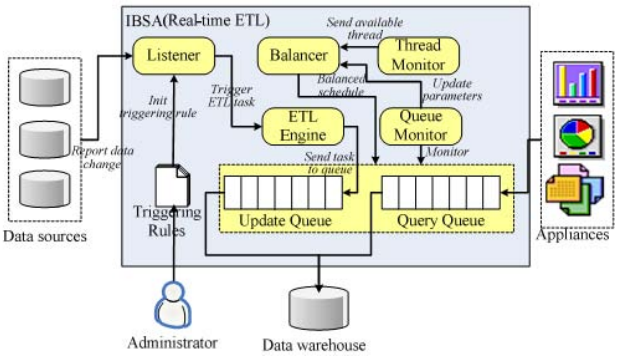


Figure 1**.** Integration based scheduling real-time ETL system.

2.3 The (m, k)-firm constraints in real-time systems

The (m, k)-firm approach was initially introduced for periodic tasks in real-time systems **[14]**, in order to relax strict real-time constraints. Indeed, the authors defined the (m, k)-firm deadline as $0<m \leq k$. The (m, k)-firm deadline expresses the acceptable quality of service, where at least m instances of a task in any window of k consecutive instances meet their deadlines **[14]**.

In addition, (m, k)-firm approach has also been adapted to transactions in real-time databases, aiming to decrease the number of missed deadlines **[1] [4].** In this paper, we adapt this method to the context of RTDWs.

3 The Proposed DETL-(m, k)-firm architecture for QoS Enhancement in RTDWs

In this section, we present the design of DETL-(m, k)-firm architecture that provides data services with QoS guarantees for RTDWs. Fig. 2 shows the DETL-(m, k) -firm architecture. It consists of operational data source systems, a change data capture (CDC) module that uses incremental extraction, i.e. only the changes made to the source systems will be extracted with respect to the previous extraction, a data transition to transform data to format required by the target system. The ETL engine transfers the ETL task to INSERT statements and send them to (m, k)-firm controller which is used to regulate the system workload in order to prevent its overloading by the reject of some optional update transactions.

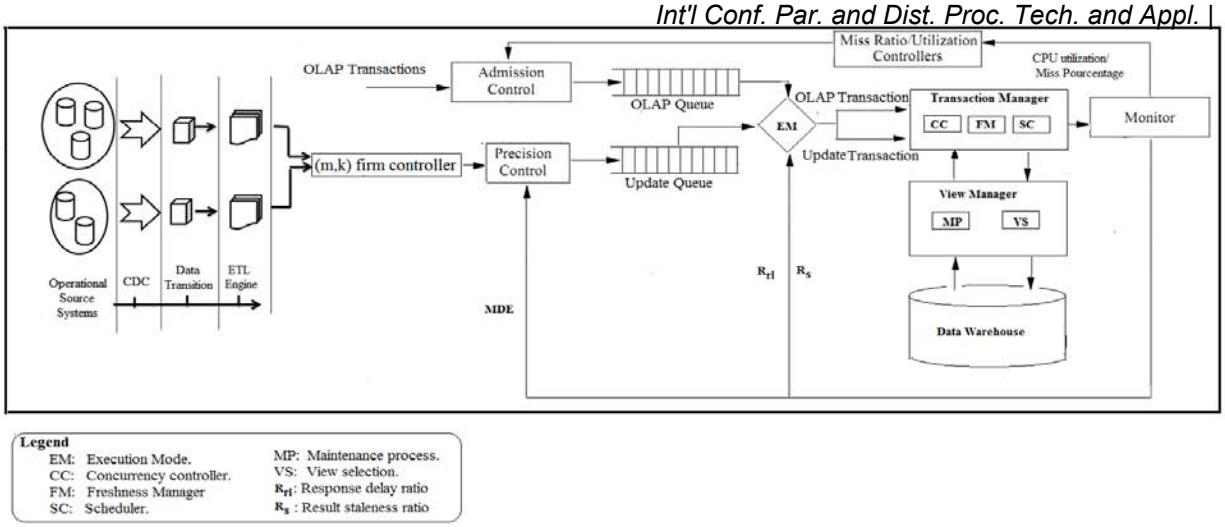


Figure 2. DETL-(m, k)-firm architecture.

The admission controller is used to regulate the system workload in order to prevent its overloading. The update transaction load is reduced by discarding update transactions according to an upper bound for the data error given by the maximum data error, denoted MDE. The transaction manager handles the transactions' execution. It consists of a concurrency controller (CC), a freshness manager (FM), a scheduler (SC). The CC solves accessing data conflicts appearing between transactions. The FM checks the data freshness. The SC is used to schedule transactions according to the EDF (Earliest Deadline First) algorithm. In addition, DETL-(m, k)-firm architecture contains an execution mode (EM) module that determines the execution mode of the present system according to two parameters **[13]:** (i) the response delay ratio (R_{rl}) and (ii) the result staleness ratio (R_s). If ($R_s \leq R_{rl}$), the query scheduler will run. Otherwise, the system mode is updated, the update scheduler will run. Moreover, R_s and R_{rl} are recalculated whenever an update or OLAP transaction is processed completely. The view manager (i) selects a set of materialized views according to a dynamic selection of materialized views algorithm (DynaSeV) which selects views from results of incoming queries under the execution time and the storage space constraints of the system and (ii) maintains materialized views according to their access frequency and the system workload **[4].**

The ETL process can involve a number of challenges. Since a data warehouse is assembled by integrating data from a number of heterogeneous sources, the ETL process has to bring all the data together in a homogeneous environment. Bellow, we present a real-time data extraction method for RTDWs and we propose a geographic data sources partitioning algorithm to deal with diversity and disparities in data source systems.

3.1 Real-time data extraction method

The first part of an ETL process is to extract the data from various source systems. Indeed, changed data capture is used to capture the data that is inserted, updated and deleted at the source side and insert the same at the target using incremental extraction.

In the traditional data warehouse, incremental data extraction technology is divided into trigger, whole table contrast and log analysis. The whole table contrast method has to copy the needed table from data source. But this method consumes lots of storage when the data table is large, also it will bring a great load.

Indeed, database management systems have a transaction log file that records all changes and modifications in a database. The log analysis method can scan and analyze the contents of the database transaction log. This method does not affect the RTDW and does not need additional storage space, which makes it suitable for RTDW.

3.2 Data sources partitioning

The major problem with the existing ETL models is the time taken by the process. In order to improve the performance of ETL, software parallel processing may be implemented to make ETL processes run faster because there is more CPU runs it. This has enabled the evolution of a number of methods to improve the overall performance of ETL processes when dealing with large volumes of data.

The requirement on the data propagation delay is highly related to the geographical distance between two data sources since the travel speed of physical disturbances is linearly proportional to the geographical distance. Let T_P be the traveling delay of physical disturbance between the two data sources

$$T_p = \frac{\text{distance between the two data sources}}{\text{propagation speed}} \quad (7)$$

To minimize T_p, data sources are partitioned into a set of clusters based on the geographical distance. Indeed, the traditional ETL task needs to handle large amount of data and process data with batch mode, which impacts performance of the data warehouse a lot.

One of the most famous partition algorithms is the K-means. The K-means is the simplest and most commonly used algorithm **[8]**. It starts with random initial centroids and keeps reassigning the patterns to clusters based on the similarity between the pattern and the cluster centroids until a convergence criterion is met after some number of iterations.

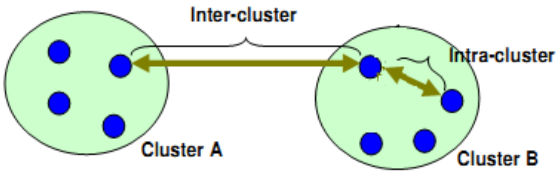


Figure 3. Data source clusters.

However, finding the right number of data source clusters is a difficult task. To alleviate this problem, we propose a new data source partitioning algorithm. The objective of our proposed algorithm is to partition data sources recursively until the intra-cluster distance (cf. Fig.3) in each cluster is less than a maximum acceptable intra-cluster distance called Max_{intra}.
As shown in Fig 4, the maximum acceptable intra-cluster distance represents the distance that across a circle through its center point and touches the two farthest data sources.

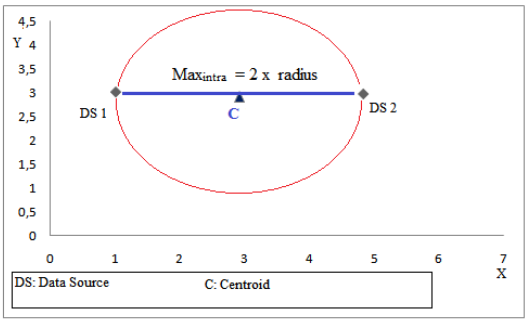


Figure 4. The maximum acceptable intra-cluster distance.

Our proposed algorithm works as follows:

Algorithm 1 Geographic data sources partitioning algorithm

Input: The set of data source DS.
Input: Acceptable data propagation delay called Max_{intra}
Begin
Let C be the initial center (usually C $\leftarrow \{\bar{x}\}$).
2: Let D_{ij} be the distance of a data source DS_i to the center C_j.
3: Let R be the radius of the circle

$$R = \frac{Max_{intra}}{2}$$

3: Let S= $\{DS_i \mid D_{ij} < R\}$ be the set of data sources assigned to center C_j.
4: Remove the data sources set S from the original data sources set DS.
5: **do**
{Repeat from step 1 to select next initial center from other set of data sources
}
While (|DS|>1 and (|S|>1).
End.

This algorithm takes as input the maximum acceptable intra-cluster distance value. Then, it divides the data sources set into clusters. Indeed, the geographic data sources partitioning algorithm starts by calculating the initial center. Then, it finds the Euclidean distance of all objects from the initial center. If this distance is less than Max_{intra}, the object is in that center of centroids. From other objects, it selects next initial centroid and we repeat from step 1. This algorithm repeats step 1 until the number of objects that are assigned to cluster is greater than 0 and there are other sources in the DS set (|DS|>1 and (|S|>1)).
Let us give an illustrative example: suppose that the RTDW has 6 data sources {DS_1, DS_2, DS_3,DS_4, DS_5, DS_6} that they were dispersed in various places and the DW administrator sets the acceptable intra-cluster distance value to 4.
The data sources are illustrated into two-dimensional coordinate system (X, Y). Indeed, each point in space in the Cartesian coordinate system locates a data source.

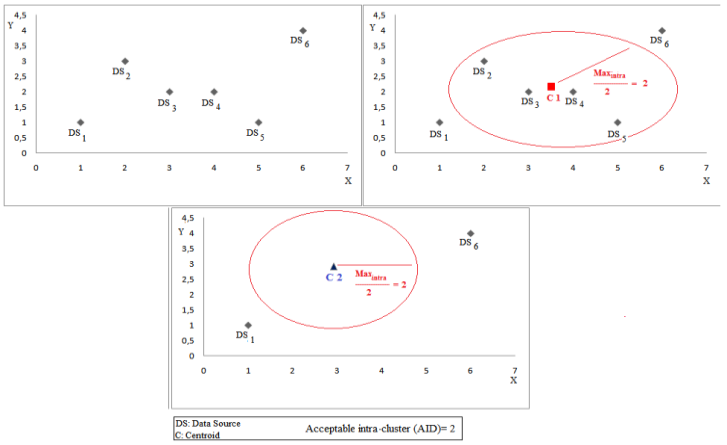


Figure 5. Geographic data sources partitioning algorithm test.

From the data source points, our proposed algorithm calculates a point whose attribute values are averages of data source points attribute values. So, the first initial centroid is average of data source points. Then, it finds distance of all data source points from the first initial centroid and it assigns each data source point in the training set to the first cluster until the intra-cluster distance in each cluster is less than an Max_{intra}. Our proposed algorithm repeats step 1 to select next initial centroids from the other data source points and it stops when the number of assigned data source points is less than 1. Finally, as shown in Fig. 5, we form the following cluster:
Cluster 1: {DS_2, DS_3, DS_4, DS_5}
Also, the distance between DS_1 and DS_5 is longer than Max_{intra}, so they cannot belong to the same cluster.
Cluster 2: {DS_1} and Cluster 3: {DS_5}

3.3 The (m, k) -firm Constraints Model for Update Transactions

The (m, k)-firm deadline expresses the acceptable quality of service, where at least m instances of a task in any window of k consecutive instances must meet their deadlines. For example, a (4, 5)-firm specification implies a 20 % maximum allowable loss rate ($\frac{k-m}{k} \times 100$). A (8, 10)-firm specification, which also implies a 20% maximum allowable loss rate, is less stringent than a (4, 5)-firm specification. A task that violates its own (m, k)-firm deadline, that is, when there are fewer than m deadline satisfactions occurring in a window of k consecutive instances, introduces a dynamic failure. Thus, the occurrence rate of dynamic failures can be used as a metric to measure how often the quality of service falls below the required level. For example, a control system can have a few occasional deadline misses without a significant degradation in control performance, provided that there are only a limited number of consecutive deadline misses.

3.4 Query Optimization in RTDW

To respect such a time constraint, the optimization techniques used with traditional DWs are continuing to be used in RTDWs. Among the most efficient optimization techniques are materialized views and data partitioning.

1) **Dynamic Management of Materialized Views in RTDW**

Roughly, a materialized view is a stored query result that may be re-built after changes have been made in its data source. The frequency of the re-reconstruction of the materialized view depends on the required freshness of data. Obviously, this frequency should be high in RTDWs so then it imposes new challenges, including efficient management of materialized views.

There have been works on view management for DWs. In **[11]**, the authors proposed a dynamic view management system for DWs called DynaMat that has two separate phases called On-line phase and Update phase. The goal of the On-line phase is to answer as many queries as possible. During the Update phase materialized views get refreshed; queries are prohibited during this phase. Therefore, DynaMat cannot well meet the requirements of the users in RTDW that should be refreshed periodically in order to prevent OLAP transactions reading extremely stale data.

The common materialized view management activities include identifying which materialized view to be created, and ensuring that all materialized views are refreshed properly each time the RTDW is updated.

To do so, the authors in **[6]** propose a dynamic materialized views selection algorithm called DynaSeV to select a proper set of materialized views based on the OLAP transactions importance and their execution times under the storage constraint. In order to maintain materialized views, the authors in **[6]** determine dynamically an update policy for them based on update adaptation threshold (UAT) for selecting a right update policy for a materialized view. Thus, this approach provides a new way to deal with materialized views management in RTDWs focusing on both data freshness and timing constraints.

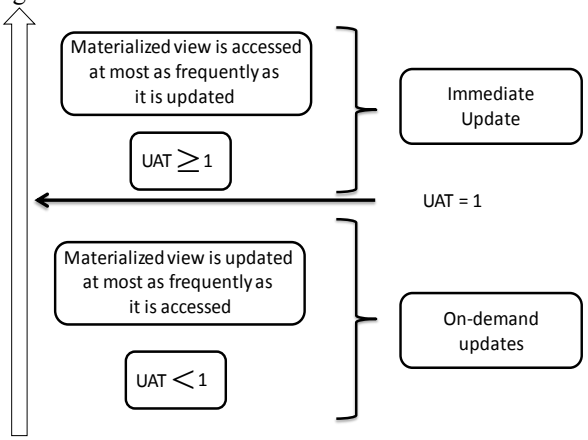


Figure 6. Dynamic Adaptation of Materialized Views Maintenance Policy **[6]**.

2) **A Data Partitioning Approach for Real-Time Data Warehouses**

Nowadays enterprises run RTDWs of hundreds of Gigabytes in size (or terabytes) stored in relational database tables, consequently data retrieval processes are time consuming. So, to speed up query processing, breaking a large table into several smaller ones is a must. In contrast, few papers in the literature have tried to address the issue of partitioning in the context of RTDW **[16]**.

To alleviate this problem, the authors in **[7]** propose a two levels efficient horizontal partitioning approach in the context of RTDW, 2LPA-RTDW. During the first-level, 2LPA-RTDW uses the G-means based fragmentation approach to set the initial partitions. Then, it adjusts the existing partitions when data amount increases by merging and splitting the existing partitions. The adjustment checks the data amount of each partition; adjust them by merging two small partitions or splitting large partitions. Indeed, each partition is processed one by one, if the current one is too bigger, it will be split; or if it is too small, try to merge into next, or such merging cannot be done because the next one is bigger, try to split the next. The data amount of each partition is controlled in an acceptable range. 2LPA-RTDW offers two advantages: firstly, we find automatically the initial number of partitions. Secondly, we keep the data amount of each partition more balanced by specifying a floating factor parameter **[7]**.

4 Experiments

In this section, we aim to evaluate the QoS performance provided by the proposed DETL-(m, k)-firm architecture in which we apply the two optimization techniques: materialized views and data partitioning techniques, based on simulation results.

4.1 Simulation Principle

We have evaluated our approach according to a set of simulation experiments, where a set of parameters have been

varied. Each simulation result represents the average of 10 simulations. The Oracle Data Integrator (ODI) is used to capture changed data from source data stores. It reduces the volume of processed data by extracting only the changed data; it is performed by journalizing models to implement the CDC mechanism. The system parameters for simulations are shown in Table 1.

Parameter	Meaning	Default
N_a	Number of queries	500 transactions
N_u	Number of updates	500 transactions
MDE[1]	Maximum Data Error ratio	0.2
$\frac{m}{k}$ ratio	The ratio of mandatory update transactions	[0.5, 0.7]

Table 1. Simulation Parameters.

We generate a set of 500 queries and 500 updates; SQL statements are from TPC-DS decision support benchmark **[18]**. ESCC (Extended Speculative Concurrency Control) **[4]** is used to address conflicts with the simultaneous accessing or altering of data that can occur. It uses both the duplication of reading transaction when a conflict is detected to resolve conflicts as read-write and write-read conflicts and the scheduling of real-time transaction based on their deadline.

The arrival numbers of queries in a second under high, medium and low loads are two, five and ten, respectively. The arrival numbers of updates in a second under high, medium and low loads are five, eight and twelve, respectively. Query execution time ranges from 100 to 1,000 milliseconds and update execution time ranges from 20 to 50 milliseconds. The acceptable response time delay ranges from 100 to 1500 milliseconds and the acceptable result staleness ranges from 10 to 100 milliseconds.

In the set of our experiments, we varied the ratio of mandatory update transactions value $\frac{m}{k}$ in order to show the effect of either increasing or decreasing the number of mandatory update transactions.

4.2 Results and Discussions

We performed a series of experiments to test the validity of our geographic data sources partitioning algorithm and to compare the performance of our DETL-(m, k)-firm approach with an existing work.

1) **Data sources partitioning Performance.**

In the next experiment, we consider a range of the number of data source clusters K to test the validity of our geographic data sources partitioning algorithm.

Theoretically, the ETL process speeding up S_{ETL} can be formulated as follows.

$$S_{ETL} = \frac{N * T_n}{(K + N - 1) * T_p} \quad (8)$$

Where, N is the number of tasks. Indeed, the ETL process in our model is divided into three sub-operations (change data capture (CDC) module, a data transition module and the ETL engine); T_n is the time to execute a task in the traditional ETL process (without data source partitioning); K is the number of the data source clusters and T_p is the time to execute a task using data source partitioning.

So, in this way all ETL process works together simultaneously and then we will be able to save a lot of clock cycles and will be able to speed up the ETL process. We can demonstrate it with the help of an example, let the time taken to process a sub operation in each module be equal to t_p= 1ms. We assume that the data sources are partitioned into K= 4 clusters and executes N= 3 process in sequence.

Clock Pulse	Change data capture (CDC) Process	Data transition process	ETL engine process
1	Data_Cluster 1		
2	Data_Cluster 2	Data_Cluster 1	
3	Data_Cluster 3	Data_Cluster 2	Data_Cluster 1
4	Data_Cluster 4	Data_Cluster 3	Data_Cluster 2
5		Data_Cluster 4	Data_Cluster 3
6			Data_Cluster 4

Table 2. The ETL process speeding up example.

In our example, the ETL process speeding up S_{ETL} is equal to

$$S_{ETL} = \frac{N * T_n}{(K + N - 1) * T_p} = \frac{3 * 4}{(4 + 3 - 1) * 1} = 2$$

So we have sped up the ETL process approximately 2 times.

In order to improve experimentally the performance of ETL when dealing with disparities in data source systems, ETL parallel processing is implemented.

The Oracle Data Integrator (ODI) captures changed data from source data stores and more precisely from .dat data file for TPC-DS benchmark **[18]**. Indeed, ODI's changed data capture identifies and captures data from the set of data sources in the same cluster and it makes the changed data available for integration processes. The simulation results show that our geographic data sources partitioning algorithm is effective in reducing the average response time in the order of milliseconds (ms) (see Fig. 7).

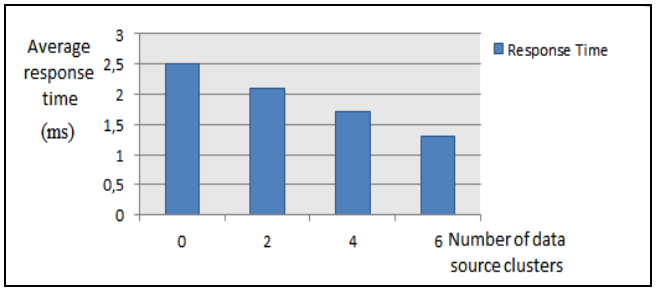


Figure 7. Average response time comparaisons.

[1] Maximum Data Error

2) **Performance Comparison**.

As shown in Fig. 8, we can assert that the DETL-(m, k)-firm architecture, with different ratios of mandatory update transactions, yields the lowest DMR of OLAP transactions, compared to the result provided by the conventional FCSA-RTDW **[5]**. Therefore, the applicability of our approach led to the average diminution of the DMR value by 10%. In addition, the simulation results shown in Fig. 9 affect the staleness result on the other hand. Indeed, less than the value of the ratio of mandatory update transactions implies a maximum allowable loss rate. So, the data freshness will be affected and thus a little degradation in QoD. In this case, URSR degradation is about 1%.

Therefore, a graceful QoS degradation is accepted in such a way that the RTDW continues to operate for providing an acceptable reduced level of service. Indeed, under the concept of the (m, k)-firm constraints, a real-time stream miss some deadlines without degrading drastically the QoS to optimize the resource use and to reduce significantly the RTDW overloads.

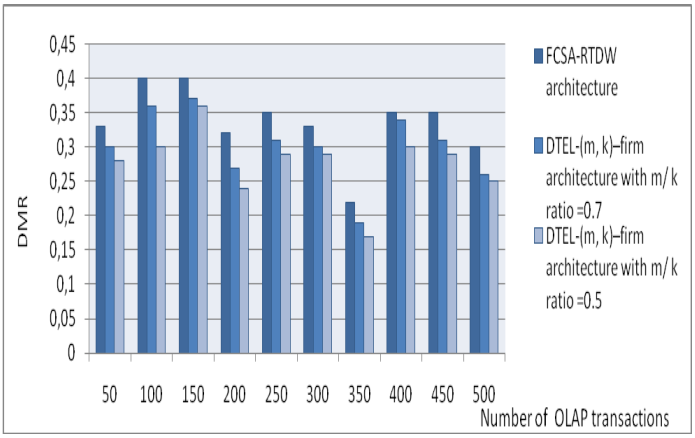


Figure 8. DMR performances.

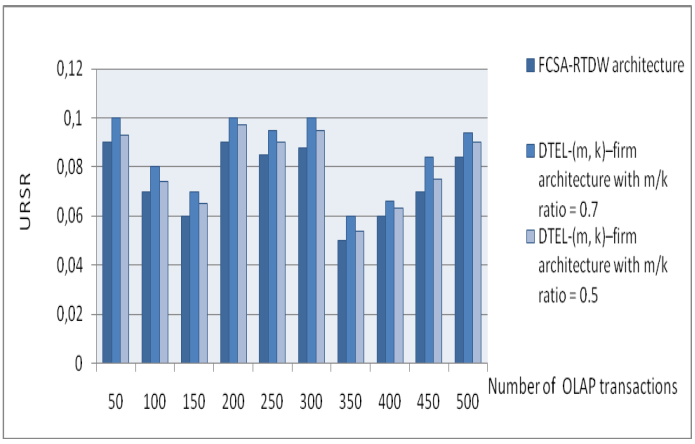


Figure 9**.** URSR performance.

5 Conclusion and Future Work.

In this paper, we have presented the DETL-(m,k)-Firm architecture as an optimized approach for ETL in RTDW which considers both materialized view and data fragmentation to improve query performance and, finally to ensure a high QoS for RTDW. In our work, we have applied the (m, k)-firm approach to update transactions. Experimental results confirmed the benefit of the proposed approach on increasing the number of transactions which meet their deadlines, even in the presence of unpredictable workload. To reduce the time for ETL process considerably, we have divided the data sources into clusters so that more than one ETL process works simultaneously to enhance the overall performance of ETL processes.

We plan to extend this work in several ways. We will propose to apply the (m, k)-firm approach on OLAP transactions, aiming to allow for more transactions to meet their deadlines without affecting the data freshness.

References

[1] Ben Salem, M., Achour F. , Bouazizi E., Bouaziz, R., and Duvallet C., Applicability of the (m, k) -firm Approach for the QoS Enhancement in Distributed RTDBMS, 13th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2013), pp. 166–175, 2013.

[2] Golab Lukasz, Johnson Theodore and Shkapenyuk Vladislav, Scalable Scheduling of Up-dates in Streaming Data Warehouses, IEEE Transaction on Knowledge and Data Eng., vol. 24, No. 06, 2012.

[3] G.X. Zhou, Q.S. Xie, Y. Hu, E-LT Integration to Heterogeneous Data Information for SMEs Networking based on E-HUB, Fourth International Conference on Natural Computation, pp. 212-216, 2008.

[4] Haubert, J., Amanton, L., Sadeg, B., Mammeri, Z., Admission Control for Relaxed Real-Time Transactions, IEEE International Computer Systems and Information Technology Conference, pp. 328–334, 2005.

[5] Issam Hamdi, Emna Bouazizi, Jamel Feki, Management of QoS and Data Freshness in Real-Time Data Warehouses using Feedback Control Scheduling, International Conference on Information Technology (ACIT), 2013.

[6] Issam Hamdi, Emna Bouazizi, Jamel Feki, Dynamic Management of Materialized Views in Real-Time Data Warehouses, 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR), pp 168-173, 2014.

[7] Issam Hamdi, Emna Bouazizi, Saleh Alshomrani, Jamel Feki, 2LPA-RTDW, A Two-Level Data Partitioning Approach for Real-time Data), IEEE/ACIS 14th International Conference on Warehouse, Computer and Information Science (ICIS), 632-638, 2015.

[8] J. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. In 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, USA, pages 281–297. University of California Press, 1967.

[9] Jie Song, Yubin Bao, Jingang Shi, A Triggering and Scheduling Approach for ETL in A Real-time Data Warehouse, 10th IEEE International Conference on Computer and Information Technology (CIT), 2010.

[10] Jingang Shi, Yubin Bao, Fangling Leng and Ge Yu: Priority-Based Balance Scheduling in Real-Time Data Warehouse, Hybrid Intelligent Systems (HIS), Vol. 3, 2009.

[11] Kotidis and N. Roussopoulus, DynaMat: A dynamic view management system for data warehouses, Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, 1999.

[12] Langseth, J., Real-Time Data Warehousing: Challenges and Solutions, DSSRe-sources.COM, 2008.

[13] Leng Fangling, Bao Yubin, Yu Ge, Shi Jingang and Cai Xiaoyan, Requirement-based Query and Update Scheduling in Real-time Data Warehouses, 12th international con-ference on Web-age information management (WAIM), 2011.

[14] Ramanathan, P., Hamdaoui, M.: A Dynamic Priority Assignment Technique for Streams with (m, k) -firm Deadlines. IEEE Trans. Comput. 44, 1443–1451, 1995.

[15] Ricardo Jorge Santos and Jorge Bernardino, Real-time data warehouse loading methodology, IDEAS, pages 49-58, 2008.

[16] Thiele Maik, Fischer Ulrike and Lehner Wolfgang: Partition-based workload scheduling in living data warehouse environments, Information Systems, vol. 34, No. 04, 2009.

[17] Thomsen, T. Pedersen, and W. Lehner, RiTE: Providing On-Demand Data for Right-Time Data Warehousing, 24th International Conference on Data Engineering (ICDE), 2008.

[18] TPC, Transaction processing performance council, http://www.tpc.org, 2014.

[19] Vassiliadis, P., Simitsis A., Near Real-Time ETL: Annals of Information Systems: New Trends in Data Warehousing and Data Analysis, Vol. 3, New York: Springer Publishing Company, 2008.

Building genomics foundation for precision medicine research: A portable multitask data management system

Yifan Zhang, Emre Ermisoglu, Dan Li

MidSouth Bioinformatics Center and Joint Bioinformatics Ph.D. Program of University of Arkansas at Little Rock and University of Arkansas for Medical Sciences, 2801 S. Univ. Ave, Little Rock, Arkansas 72204 U.S.A.

David Geisert, William Yang, Kenji Yoshigoe, Chad Haydan

Department of Computer Science, George Washington Donaghey College of Engineering & Information Technology, University of Arkansas at Little Rock, 2801 S. University Ave, Little Rock, Arkansas 72204 U.S.A.

Mary Yang

MidSouth Bioinformatics Center, Department of Information Science and Joint Bioinformatics Ph.D. Program of University of Arkansas at Little Rock and University of Arkansas for Medical Sciences, 2801 S. University Avenue, Little Rock, Arkansas 72204 U.S.A

mqyang@ualr.edu

I. Introduction

With the advent of high-throughput Next Generation Sequencing (NGS) technology, unprecedented amounts of sequence data have been generated. The exponential growth of sequence data not only poses challenges in data processing, transfer, storage and analysis, but also fosters the development of novel high-performance computing approaches for precision medicine research. To this end, we developed a portable data management system for automated process of large-scale multidivisional genomic big data.

Precision medicine is a relatively new term that is now gradually replacing the personalized medicine regarding to how genome-scale information can be used for accurate diagnose and treatment planning of diseases. While personalized medicine would eventually reform the current healthcare based on average patient, it generated a misunderstanding to the general populace beyond our current ability, in which the diagnosis, treatment and prevention of diseases are based on individual patient uniquely. Precision medicine on the other hand, focuses on the holistic consideration of genetics, environment, risk factors and life styles to provide an optimized treatment panning tailored toward individual patient rather uniquely individual treatment distinctively. While the goals between precision medicine and personalized medicine need to be further clarified, it is still common that these two terms are often used interchangeably.

ISBN: 1-60132-444-8, CSREA Press ©

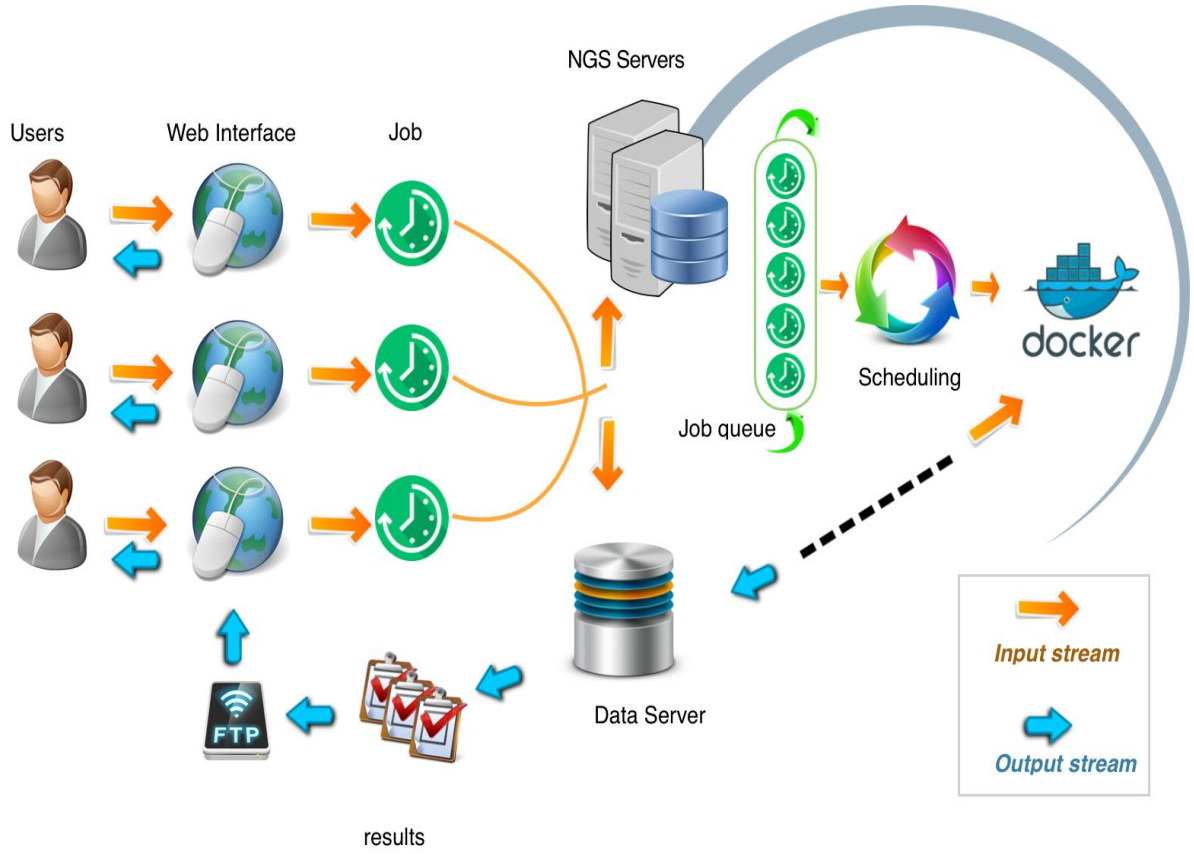


Figure 1: Overview of the NGS Data Management System

The advances of precision medicine clearly rely on the effective handling of big data to facilitate the direct clinical decision-making and outcome assessments for individual patients. To catalysis the process, developing big-data analytics infrastructures and approaches for data collection, quality control, storage, processing, sharing and applications are demanded. This will continuously foster a hybrid research – healthcare system that seamlessly integrates genomics research into practical healthcare.

Presently, NGS is becoming more accessible and affordable to many laboratories since sequencing cost continues to decrease. The Systems Genomics Laboratory of University of Arkansas at Little Rock aims to integrate multidimensional genomic data to facilitate the precision medicine research [1]. We have developed an online tool called **IDEAS** to **I**dentify **D**ifferential **E**xpression of genes for **A**pplications in genome-wide **S**tudies [2]. The online tool IDEAS has been tested to obtain genome-wide differential expression of genes for The Cancer Genome Atlas (TCGA) datasets such as the Kidney Rental Clear Cell Carcinoma data that we have studied [3]. IDEAS online tool provides an open access freeware for to facilitate precision medicine [4]. In this paper, we developed an NGS data management system in parallel to address more challenges in handling genomic big data. Our system not only automates data processing and analysis, but also allows efficient utilization of available resources. The system integrates the processing of data inputs and outputs, storage, sharing, utilization and applications.

Since NGS data are usually very big, high performance computing is often needed. We recently developed new high performance computing techniques for High-throughput state-machine replication using software transactional memory [5]. In this project, we consider that Docker [6] is a platform that can package an application and its dependencies so that the application is able to run in any Linux server, our NGS workflows were built using Docker container technology.

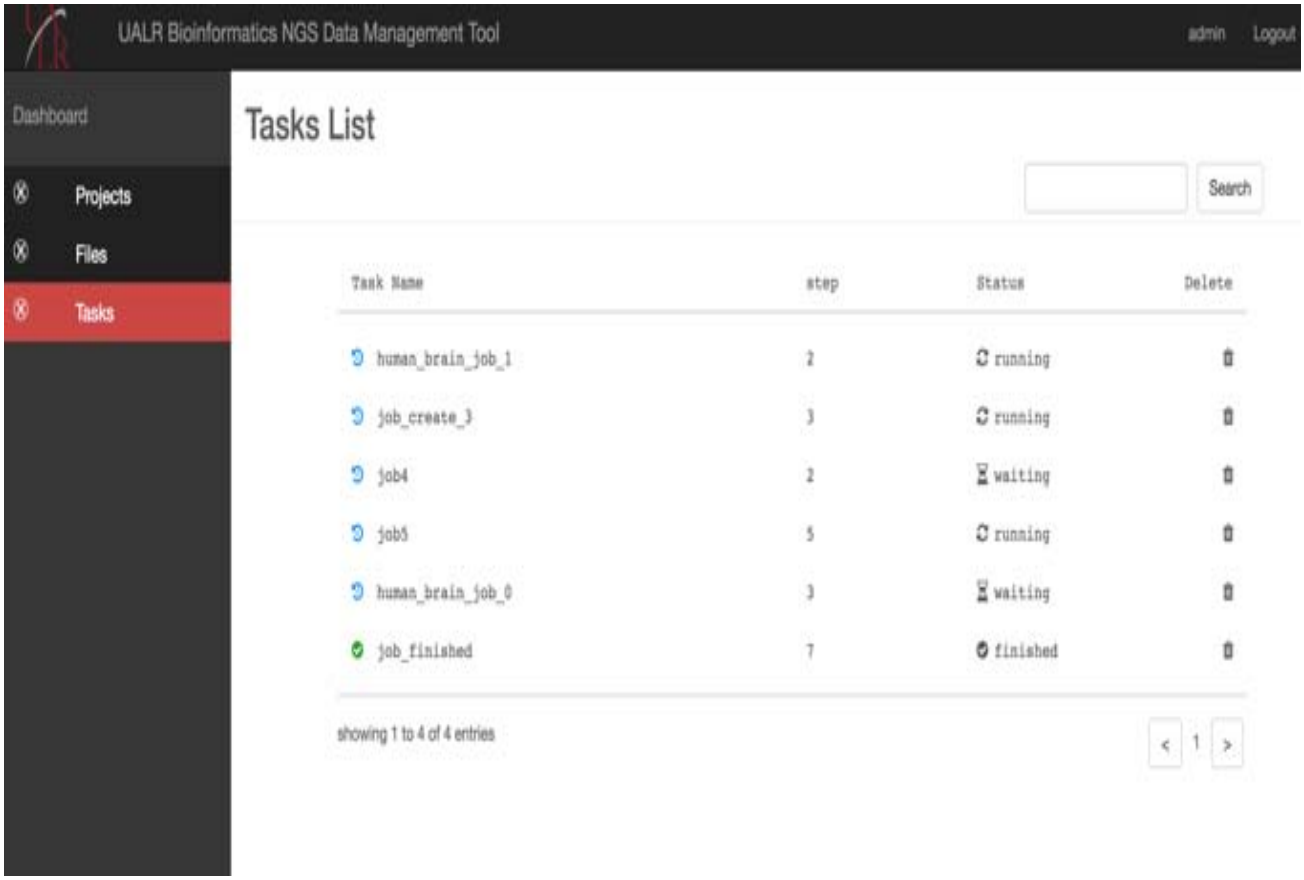


Figure 2: Web Interface for Tasks List

In our system, a Job Scheduler and a Docker service manager work together to ensure efficient allocation of resources, such as CPUs and memories, to the jobs in the queue. The Job Scheduler determines appropriate cores, memory, and order of execution of a job. Given physical memory limitation, we determine the optimal allocation of CPUs to accelerate job processing. In our system, the Docker Service Manager keeps track of available Docker servers, executes commands on each Docker server using the Docker remote APIs, and keeps track of the computational resources used by each running Docker container. Therefore the NGS management system is highly flexible and portable, and can run on local servers as well as in cloud services. This web tool was designed to facilitate biological discovery from large-scale sequence data with easy to operate features.

II. Materials and Methods

We developed the NGS Data Management System as a standalone, scalable, and easy to

use platform for managing NGS data efficiently. The system's user interface is a web application and it can be accessed with any device that has an internet connection. Users can track and analyze their data via the web interface similar to IDEAS that we developed. The system streamlines the quality control, read alignment, variant calling and functional annotation. The user interface was designed to be intuitive and easy to operate for users.

On the backend, the Docker engine is used to achieve scalability. New processing servers and storage units can be added to the system easily. We developed a Job Scheduler to organize tasks efficiently and maximize the usage of each processing unit. The overall architecture of the tool is shown in figure 1. Our system contains an easy to user web interface that allows users to create an account so that they can login and manage their projects. This provides users a functionality to upload their data files, choose available pipelines in our system and submit the job.

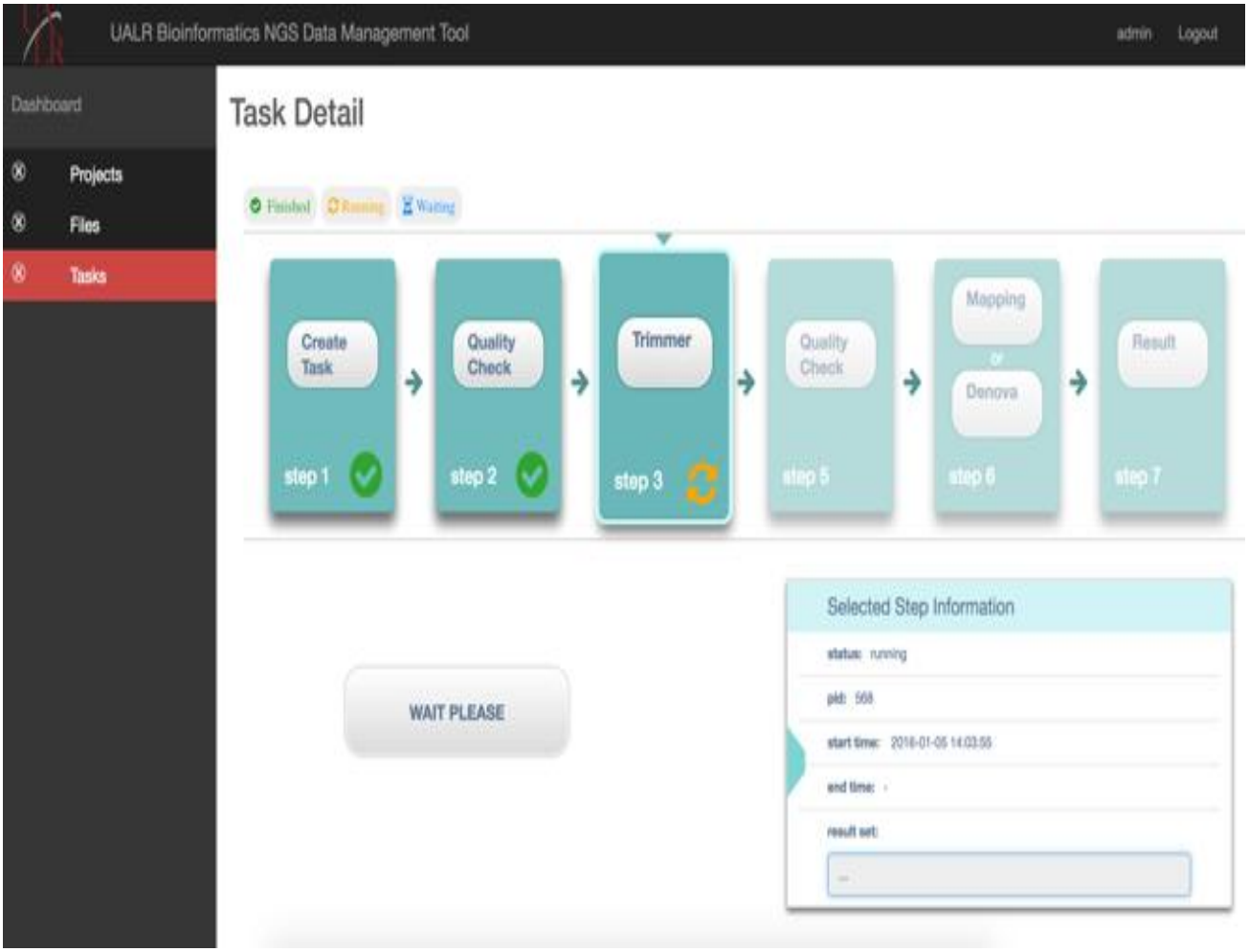


Figure 3 - Web User Interface for Task Detail

All jobs are executed on our server, and the results are stored in the FTP server of the Systems Genomics Laboratory of University of Arkansas at Little Rock. Users can see all the information about their jobs through the web interface including the status of their submitted jobs and the results of completed jobs. Figures 2 and 3 show the Tasks List and Task Detail pages of our websites. Since PHP [7] is particularly designed as a scripting language for server-side web development, we use a free open source PHP framework named Laravel in our server and use HTML and JaveScrpit to build our web pages. Our framework uses MVC model [8] to boost the features such as authentication, email and queue services.

We store the job and user information to share the information with other part of the system. For example, when a new job was submitted, the system initially set "waiting" status, so that the Job Scheduler starts to schedule it. The system then collects all the waiting jobs in the database and builds a queue base on the priority of each job. The system communicates with each component with the information in the database to enable low coupling. The Job Scheduler is responsible for handling a job queue that consists of the jobs submitted by the users from the web interface. As shown in Figure 4, this part collects the information such as free CPU cores and available memory in each server as well as the resource needs of every job.

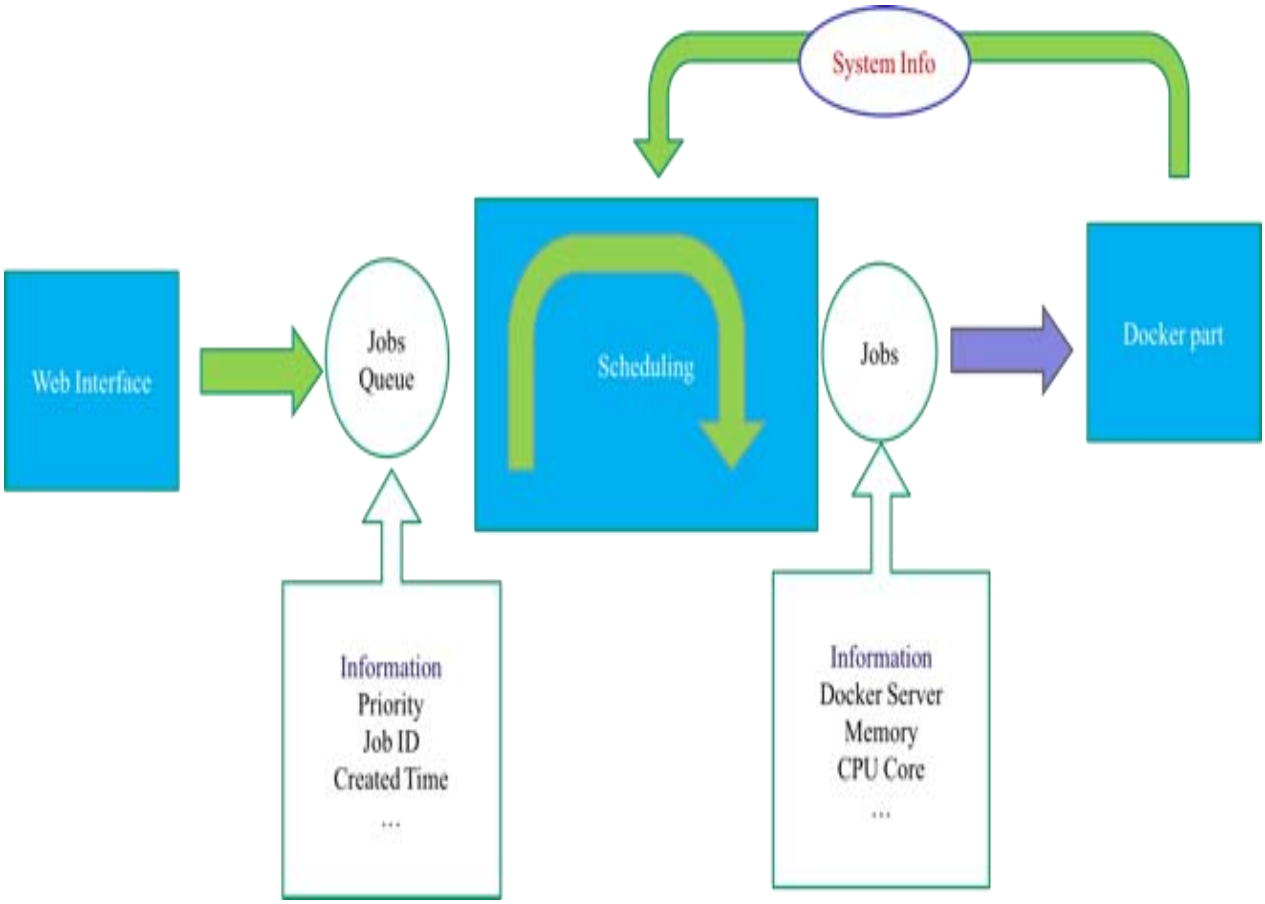


Figure 4: Job Scheduler infrastructure

Based on the information, the Job Scheduler determines which jobs shall run at given time and how many resources need to be allocated to each individual job.

The Job Scheduler was designed to maximize the usage of server resources and to ensure that the jobs are executed in the correct order efficiently. To optimize the scheduling, we assess the influences of memories and CPUs and runtime of jobs. For example, we measure the total running time for processing and mapping paired-end human RNA-seq data (~6.7GB) to the reference genome by assigning different number of cores and amount of memory (Figures 5 and 6). From both figures, we can see that the multicores boost the speed of processing a job up to about 6 cores given 4G physical memory. Using 7 or more cores without allocating more memory virtually slows down the running speed. This is because the more cores that a job occupy, much more amount of memory are needed for coordination. Therefore once the total memory exceeds physical capacity, the CPU cores spend more time to wait for IO, and the running time indeed increases accordingly. On another experiment, we tested that 12-20 cores with 8/16G physical memory, then we increased memory to 20G. In this experiment, more cores actually reduce the running time. To conclude, allocation of CPUs shall be appropriate to available physical memory and our system can handle this optimally to ensure that all jobs are executed in the correct order with appropriate resources.

The system also selects the best practiced RNA-seq data analysis algorithms and tools to build the processing pipeline. FastQC was used to assess the quality of input RNA-seq data. The reads failed to pass the test were then trimmed by Trimmomatic [9]. The users were allowed to check the quality of testing, adjust the parameters and redo the quality control. We apply Tophat2 [10] for the reads alignment and the toolkit Cufflinks [11] to assemble transcripts. Several alignment assessment tools such as Samtools [12], Picard [13] and QualiMap [14] were also added to the pipeline. The index data of model organisms such as human and mouse were stored in our storage unit. In our system, Docker uses Linux Containers to allow for a scalable and lightweight computing infrastructure that is easy to deploy to multiple environments and platforms. Each NGS pipeline has an image that creates a container for every initialized pipeline job.

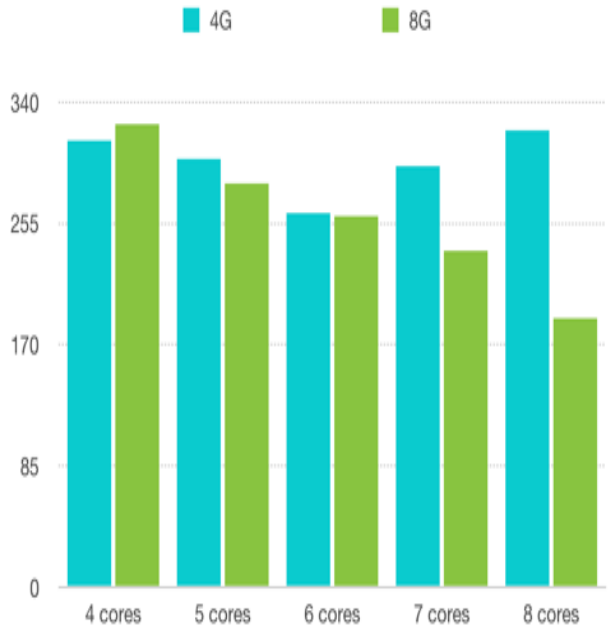


Figure 5 - The relationship between the running time and the allocated CPU cores & memory (smaller size)

Each Docker server is set up with a volume, for containers to temporarily store data, and a container, for transferring files between the pipeline containers and the data storage unit.

Each pipeline container attaches to its own directory with input files and output files. There is a data transfer container which transfers the input files for the execution of each pipeline container. After a job is complete, its associated pipeline container will be deleted, and the output files will be copied to the storage unit for users to view. We introduce a newly developed NGS Data Management System and present the development of a high performance computing based web tool to improve the acquisition, analysis, integration and utilization of NGS data as shown in Figure 7. The system is fully automated to process NGS data.

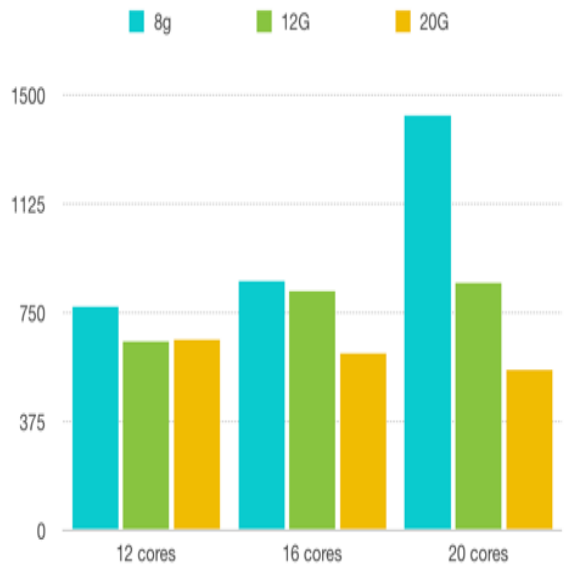


Figure 6 - The relationship between the running time and the allocated CPU cores & memory (larger size)

III. Discussions and Conclusion

The newly developed system consists of web user interface, data transfer component, and server management tool. The user friendly web interface facilitates users to control and process the data. The data transfer portion enables users to transfer data using a fast and secure protocol, while the server management tool manages, stores, and processes the data interactively. The newly developed NGS Data Management System can process genomic big data and facilitate precision medicine research.

The system also integrates Docker into computational pipelines. With Docker, we are able to package an application to an image and run multiple instances of the image in containers using a runtime environment that is unique to the container. This allows for a lightweight and viable solution for distributing computing environments in the cloud. In addition, the system allows us to manage and maintain multiple Docker servers through the usage of the remote APIs. Given available physical memory, the running time decreased when appropriate CPU cores are assigned to a job. A reference mapping based pipeline requires much less memory than a *de novo* assembly based pipeline. As far as the minimum required memory was allocated for a job, allocating additional memory did not reduce the running time significantly. While this project was tested with various data, further development of this tool includes parallel processing of multidimensional genomic big-data and the identification of disturbed gene networks and pathways utilizing this tool. Further utilization of this tool to facilitate precision medicine research will be reported in the forthcoming publications of the Systems Genomics Laboratory of University of Arkansas at Little Rock.

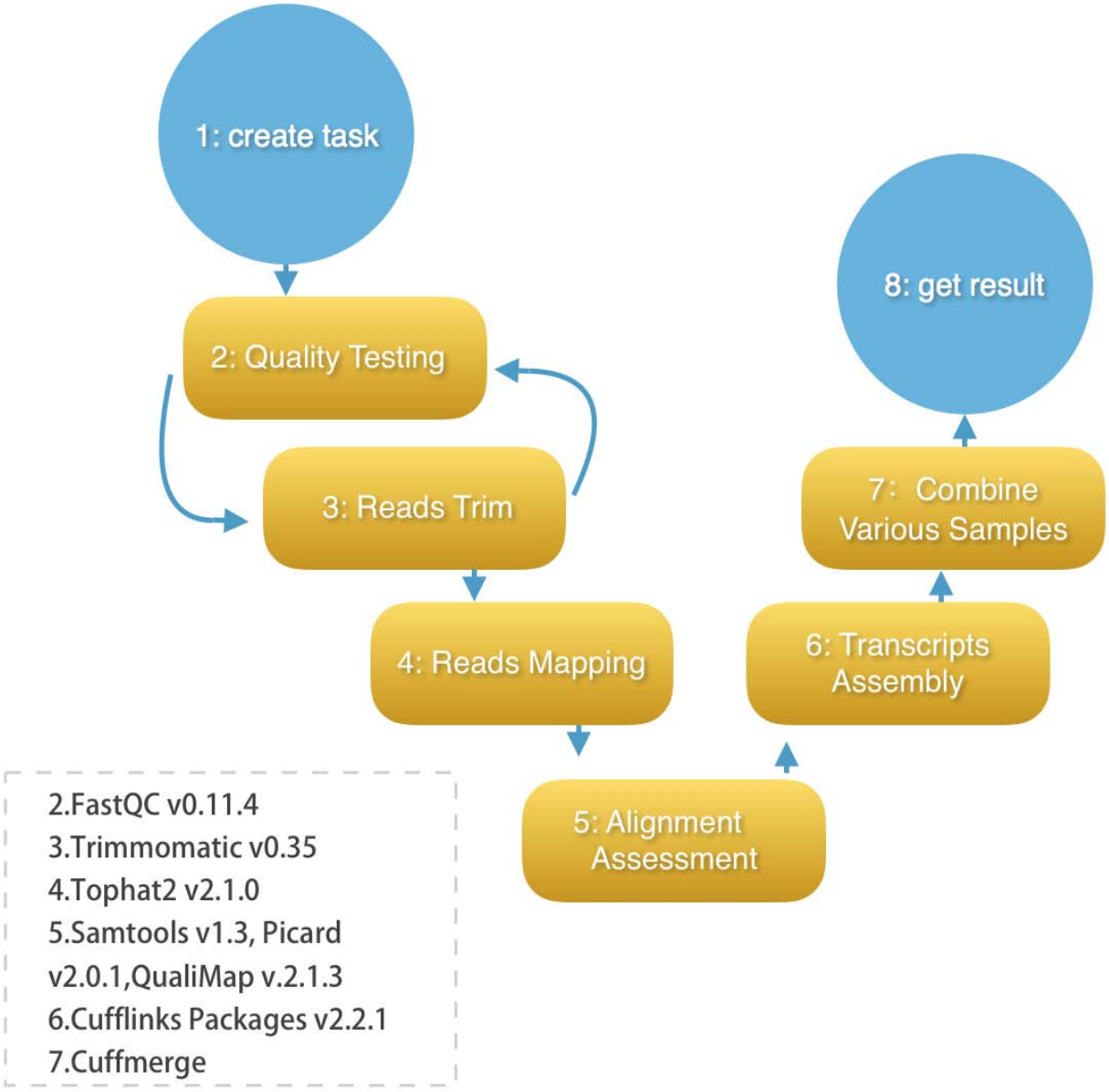


Figure 7 – The workflow of processing pipelines in the system.

IV. Acknowledgements

This project was supported by NIH 1R15GM114739, FDA BAA-15-00121 - HHSF223201510172C and ASTA 15-B-23 (PI: Mary Yang). The Systems Genomics Laboratory of University of Arkansas at Little Rock provided computing resources and supports of graduate and undergraduate students. The computational resources of William Yang and Kenji Yoshigoe were also supported by NSF MRI Award #1429160. Research activities at MidSouth Bioinformatics Center were also supported by NCRR P20RR016460 and NIGMS P20GM103429. Elizabeth Pierce, Chair of Information Science Department of University of Arkansas at Little Rock is acknowledged for providing generous academic supports to faculty and students.

V. References

[1]. Yang, M., (2016). Keynote Lecture: "Developing Synergistic Intelligent Computing and Big Data Analytics Approaches to Facilitate Precision Medicine Research". http://worldcomp.org/events/2016/keynotes/mary_yang_keynote

[2]. Yang, W., Yoshigoe K. et. al. (2016). "**IDEAS,** an online to **I**dentify **D**ifferential **E**xpression of genes for **A**pplications in genome-wide **S**tudies". http://worldcomp.ucmss.com/cr/main/papersNew/LFSCSREApapers/ABD7587.pdf

[3] Yang, W., Yoshigoe, K. et al. (2014). Identification of genes and pathways involved in kidney renal clear cell carcinoma. *BMC Bioinformatics,* Vol. 15 (Suppl 17), S2. http://doi.org/10.1186/1471-2105-15-S17-S2

[4]. Yang, W., (2016). **"**Tutorial Lecture: Resonance of big-data analytics and precision medicine research is producing a profound impact on optimized individual healthcare". http://worldcomp.org/events/2016/tutorials/william_yang_tutorial

[5]. Zhao, W, Yang, W., et. al. (2016). High-throughput state-machine replication using software transactional memory. *Journal of Supercomputing,* DOI: 10.1007/s11227-016-1747-2

- [6]. Docker source code (2015). https://github.com/docker/distribution
-
- [7]. PHP: http://www.php.net

[8]. What are the benefits of MVC? (2008) http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/

[9] Bolger, A. et. al. (2014). "Trimmomatic: a flexible trimmer for Illumina sequence data". *Bioinformatics*, 30(15): 2114–2120. doi: 10.1093/bioinformatics/btu170. PMCID: PMC4103590.

[10]. Kim, D. et. al. (2013) "TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions" *Genome Biology*, 201314:R36, DOI: 10.1186/gb-2013-14-4-r36, BioMed Central, 2013

[11]. Cufflinks: http://cole-trapnell-lab.github.io/cufflinks/

[12]. Samtools: http://www.htslib.org

[13]. Picard: https://broadinstitute.github.io/picard

[14]. García-Alcalde, F. et. al. "Qualimap: evaluating next-generation sequencing alignment data". *Bioinformatics.* 2012 Oct 15; 28(20):2678-9. doi: 10.1093/bioinformatics/bts503. Epub 2012 Aug 22.

Optimization and Parallelization of typical Polyhedron Program

Omar Ben Maaouia, Emna Hammami
University of Tunis El Manar, Faculty of Sciences of Tunis,
University Campus - 2092 Manar II, Tunis, Tunisia

Abstract - *Polyhedron programs (PP) i.e. DO nested loops with affine bounds are basic modules in large scientific applications. We address here a theoretical and experimental study of various techniques and transformations for the optimization of sequential PP and their parallelization. The first part deals with techniques for improving data locality and memory accesses through loop interchange and loop invariant. We then study the parallelization of PP's through dependency analysis and the use of specific transformations: one unimodular (loop interchange) and one general (partial loops partition). One specific PP corresponding to matrix computing kernels structured in 3-loop nests have been chosen to illustrate our contribution: the Gaussian Elimination. By applying the techniques mentioned above, we could derive various versions that were theoretically compared. In order to validate our contribution, we then achieved an experimental study covering the sequential and parallel versions where a quadcore bi-processor machine was targeted.*

Keywords. Loop interchange, Multicore platforms, Optimization , Parallelization, Partial Loop Partition, Polyhedron program.

1 Introduction

Parallel computing (PC) has become the dominant paradigm in computer architecture, mainly in the form of multicore processors in diverse scientific areas. The PC's basic idea is to simultaneously distribute different parts of complex application on multiple cores to be processed in a relatively short time. In the literature, the most used program structure in scientific computing is the polyhedron programs (PP) i.e. structured programs in nested loops with affine bounds translated by DO or FOR nested loop. Thus, starting from a sequential PP, there are several transformations which have been proposed in the literature in order to improve the exploitation of parallelism, e.g. loop interchange [1], loop unrolling [2] [3], skewing [4], partial loop partition (PLP) [5], strip mining [6], etc. All these techniques are designed to optimize the code parallelization depending on the intrinsic characteristics of the given code and the target platform. Besides, code parallelization requires an important preliminary step namely the dependency analysis [6][7]. In this work, we are interested in the theoretical and the experimental study of performance improvement techniques and parallelization of polyhedron programs. We targeted as a PP benchmark the Gaussian Elimination algorithm (EG) which is used in many scientific applications. In fact, such benchmark is composed of three nested loops, and flexible for generating semantically diverse equivalent versions, illustrating different intrinsic properties in sequential as well as in parallel case.

The remainder of the paper involves six sections, organized as follows: In the second section, we define our target PP which is the EG algorithm. Section 3 is devoted to the description of our theoretical study for both sequential and parallel cases of PP in general and EG in particular. There, we present various versions, analyze their characteristics, evaluate their performance and make comparisons according some specific criteria. Section 4 will illustrate our experimental study of the sequential versions and section 5 will show the experimental evaluation of the different parallel versions, which allows our theoretical contribution validation. We finally conclude and present some perspectives in section 6.

2 Our target benchmark

The Gaussian Elimination algorithm (EG) is a benchmark used for the resolution of non-singular dense linear equations. If we note this system (S) $Ax = b$, the EG consists in converting (S) into an equivalent system (S') $A'x=b'$ where A' is an upper triangular matrix. Then, the resolution of the system (S') is handled by the back-substitution [8].

We retain here only the triangulation algorithm of the matrix A without pivoting. We assume that A has the right properties which ensure the algorithm constructability [8]. For a given system (S) in order n, the studied algorithm has n-1 steps and consisting at the step k (1…n-1) in eliminating the unknown x_k in the equations k+1…n. This involves constructing, from the matrix A (denoted by $A^{(0)}$), matrixes sequence $A^{(k)}$, k=1…n-1, such that $A^{(n-1)}$ is upper triangular. The elements of $A^{(k)}$ are obtained from those of $A^{(k-1)}$ by the following recursive formulas:

$$a_i^{(k)} = a_{ij}^{(k-1)} - a_{i\,k-1}^{(k-1)} * a_{k-1\,j}^{(k-1)} / a_{k-1\,k-1}^{(k-1)}$$

where $k=1…n-1$; $i, j=k+1…n$

The standard version denoted KIJ is a perfect nest of three loops as follows:

```
DO k=1, n-1                                   / B1 /
  DO i=k+1,n                                  / B2 /
    DO j=k+1, n                               / B3 /
      A(i,j)=A(i,j)-A(i,k)*A(k,j)/A(k,k)      / S(k,i,j)/
    ENDDO
  ENDDO
ENDDO
```

3 Our Proposed approach

From the Gaussian Elimination (EG), we can derive two main versions as follows: the first is a perfect nested loop (NP) and the second one is a non-perfect nested loop (NNP). From each one, other sub-versions can be derived by applying the loop interchange technique.

3.1 Perfect Nest versions

3.1.1 Sequential versions

By examining the innermost loop of the standard version of EG, denoted KIJ, we notice that the elements A(i,j), j=k+1…n, are computed line per line. The access to A(i,k) is performed per column and that of to A(k,j) is made per row. The computation is as follows: *vector=vector+ scalar*vector*. It is a kernel whose type is AXPY. Its complexity is $C(n)=n^3 3n^2/2+n/2$. When we applied the loop interchange on the EG, five other valid versions can be derived. Thus, we obtain five NP permutations denoted KJI, JKI, IKJ, and JIK IJK. for each version, the complexity $C(n) = n^3-3n^2/2+n/2$. Table I illustrates more characteristics of the different permutations, where we note loops bounds and the access types (L for row and C for column). Note that in versions IJK and JIK, the upper bound of the third loop K is not affine (min(i-1,j-1)).

TABLE I. COMPARISON OF THE DIFFERENT PERMUTATIONS OF EG_NP

Versions		KIJ	KJI	IKJ	JKI	IJK	JIK
Loop bounds	B1	1, n-1	1,n-1	2, n	2, n	2, n	2, n
	B2	k+1,n	k+1, n	1, i-	1, j-1	2, n	2, n
	B3	k+1,n	k+1, n	k+1, n	k+1, n	1,min(i-1,j-1)	1,min(i-1,j-1)
Access Type	A(i,j	L	C	L	C	L	C
	A(i,k)	C	C	L	C	L	L
	A(k,j)	L	L	L	C	C	C
	Kernel	AXPY	AXPY	GAXPY	GAXPY	DOT	DOT

Notice that the best version is IKJ with the GAXPY-L kernel where the programming environment is C. In fact, it is related to the arrays storage which is performed line per line in C. This shows the effect of data reuse and highlights data locality.

3.1.2 Initial parallelization

In this section, we study the KIJ version in sequential and then parallel case, by performing the direct dependencies analysis [9] [5] [10]. Through the semantic of the algorithm based on recursion formulas mentioned above (see section 3.1.1), it is clear that the K loop is sequential. In other terms, we have S(i,j,k) δ* S(i,j,k+1).

For fixed k, we see that all updates of A (i, j), for i and j ranging from 1 to n + k are independent. Thus, loops i and j are parallel. We deduce that the vector of signs distances dependencies VSDD is $(+\ 0\ 0)^T$. Notice that the other five versions derived by loop interchange are all valid.
So, the KIJ parallel algorithm can be written as follows:

```
DOSER k=1, n-1          / B1 /
  DOPAR i=k+1, n        / B2 /
    DOPAR j=k+1, n      / B3 /
      S(k,i,j)
    ENDDOPAR
  ENDDOPAR
ENDDOSER
```

We denote $\mathbf{T_s}$ the sequential time of all versions whose the complexity i.e. $C(n)=n^3-3n^2/2+n/2$, We mention that the following notations are adopted in table II of the studied codes:
Type of loop: S for sequential, P for parallel
p_{max}: the maximum degree of parallelism (i.e. the maximum number of iterations that could be executed in parallel)
T_p: parallel time (for a processors number equal to p_{max})
S: speed-up = T_s / T_p
Sa: asymptotic speed-up (S for very high n)
E: Efficiency = 1 / S
Ea: asymptotic efficiency (E for very high n)

TABLE II. PARALLELIZATION OF THE DIFFERENT VERSIONS OF EG_NP

Version	KIJ	KJI	IKJ	JKI	IJK	JIK
Loop Type	SPP	SPP	SSP	SSP	SSS	SSS
T_p	3(n-1)	3(n-)	$3(n-)^2$	$3(n-)^2$	-	-
pmax	$(n-1)^2$	$(n-1)^2$	n-1	n-1	-	-
Sa	$n^2/3$	$n^2/3$	n/3	n/3	-	-
Ea	1/3	1/3	1/3	1/3	-	-

3.1.3 Parallel versions with PLP

When the number of available processors, denoted p, is strictly less than p_{max}, we may apply the PLP technique for rewriting the four versions in order to obtain at least one parallel loop. Note that for SPP, we set p=p1∗p2, knowing that p1 (resp. p2) processors will be allocated to the parallel loop whose depth is 2 (resp. 3). The KIJ_PLP version can be rewritten as follows:

```
DOSER k=1,n-1                 / B1/
  DOPAR s=1,p1                / B2/
    DOSER i=k+s,n,p1          / B3/
      DOPAR q=1,p2            / B4/
        DOSER j=k+s,n,p2      / B5/
          S(k,i,j)
        ENDDOSER
      ENDDOPAR
    ENDDOSER
  ENDDOPAR
ENDDOSER
```

We obtain a nest of 5 loops SPSPS. The KJI version can also turn into five-nested loops as the same type SPSPS. For the IKJ and JKI versions, they could only switch to four-nested loops SSPS. Table III summarizes the new results rewriting the four versions KIJ, KJI, IKJ and JKI.

TABLE III. PARALLELIZATION OF THE EG_NP VERSIONS WITH PLP

Versions		KIJ (KsIqJ)	KJI (KsJqI)	IKJ (IKsJ)	JKI (JKsI)
Loop Bounds	B1	1, n-1	1, n-1	2, n	2, n
	B2	1, p1	1, p1	1, i-1	1, j-1
	B3	k+ s, n, p1	k+ s, n, p1	1, p	1, p
	B4	1,p2	1,p2	k+ s, n, p	k+ s, n, p
	B5	k+q, n, p2	k+q, n, p2	-	-
Loop Types		SPSPS	SPSPS	SSPS	SSPS

3.2 Imperfect Nest versions

3.2.1 Sequential versions

By applying a movement of the loop invariant, it is possible to derive from the perfect nest KIJ (KIJ-NP), a non-perfect nest (KIJ-NNP) as follows::

KIJ-NNP

```
DO k=1,n- 1                          / B1 /
    coef =1/A(k,k)                   / S1 /
    DO i= k+1,n                      / B2 /
        T(i)=coef*A(i,k)             / S2 /
        DO j= k+1,n                  / B3 /
            A(i,j)=A(i,j)-T(i)*A(k,j)  / S3 /
        ENDDO
    ENDDO
ENDDO
```

This transformation allows reducing the complexity to: $C(n)=2n3/3 - n2/2+5n/6-1$. Then, we derive the other 5 NNP versions corresponding to these permutations, KJI, JKI, IKJ, JIK and IJK. The complexity is still $2n^3/3+O(n^2)$ with a bit difference in order 2, 1 and 0 (see Table IV).

TABLE IV. COMPARISON OF THE PERMUTATIONS OF EG_NNP

Versions		KIJ	KJI	IKJ	JKI	JIK : J(IK-IK)	IJK : I(JK-
Complexity		$2n^3/3-n^2/2+5n/6-1$	$2n^3/3-n^2/2+5n/6-1$	$2n^3/3-n^2/2-n/6-$	$2n^3/3-n^2/2-n/6-2$	$2n^3/3-n^2+n/3-2$	$2n^3/3-n^2/2-7n/6-4$
Loop bounds	B1	1, n-1	1,n-1	2, n	2, n	2, n	2, n
	B2	k+1, n	k+1, n	1, i-1	1, j-1	2, j-1	2, i
	B3	k+1, n	k+1, n	j+1, n	k+1, n	1, i-1	1, j-1
	B4	-	-	-	-	j, n	i+1, n
	B5	-	-	-	-	1, j-1	1, i-1
Access Type	A(i,j)	L	C	L	C	C	L
	A(i,k)	C	C	L	C	L	L
	A(k,j)	L	L	L	C	C	C
	kernel	AXPY	AXPY	GAXPY	GAXPY	DOT	DOT
Nest Scheme		[■[■[■]]]	[■[■[■]]]	[[■[■]]]	[[■[■]]]	[[■]][■[■]]]	[[■]][■[■]]]

Moreover, since the initial version KIJ is a non-perfect nest, we rely on various well-known works in the literature [8] to derivate the five other versions. This was quite hard compared to the previous case (EG-NP).

Table IV summarizes the major features of all the studied versions. The IKJ version is theoretically the best because it has a GAXPY line as kernel if we assume a line storage method (C environment). Notice that for perfect nests, the IKJ permutation was also a GAXPY-L kernel.

3.2.2 Initial parallelization

The dependencies analysis carried for the different versions have determined a type for each loop as mentioned in Table V below.

TABLE V. PARALLELIZATION OF THE EG_NNP PERMUTATIONS

Versions	KIJ	KJI	JKI	IKJ	J(IK-IK)	I(JK-JK)
Loop Type	SPP	SPP	SSP	SSP	S (SS-SS)	S (SS-SS)
T_p	3(n-1)	3(n-1)	$3(n-1)^2$	$3(n-1)^2$	-	-
pmax	$(n-1)^2$	$(n-1)^2$	n-1	n-1	-	-
Sa	$2n^2/9$	$2n^2/9$	2n/9	2n/9	-	-
Ea	2/9	2/9	2/9	2/9	-	-

3.2.3 Parallel versions with PLP

As studied in the NP case, where the number of available processors is less than p_{max}, we may use the PLP technique consisting in replacing each parallel loop by a two nested loop, the first one is parallel and the second is sequential.

In the other nest, the first loop is sequential and the second is parallel. Table VI shows the diverse characteristics of the new obtained nests.

TABLE VI. COMPARISON OF THE PERMUTATIONS OF EG_NNP_PLP

Versions		KIJ :KsIqJ	KJI :KsJqI	IKJ :IKsJ	JKI :JKsI
Loop bounds	B1	1, n-1	1, n-1	2, n	2, n
	B2	1, p1	1, p1	1, i-1	1, j-1
	B3	k+ s, n, p1	k+ s, n, p1	1, p	1, p
	B4	1,p2	1,p2	k+ s, n, p	k+ s, n, p
	B5	k+q, n, p2	k+q, n, p2	-	-
Loop Type		SPSPS	SPSPS	SSPS	SSPS

A comparison between the NP and NNP versions shows that, although the sequential NNP versions (for large n) are faster 1.5 times than the NP -($2n^3/3+O(n^2)$) for the NNP against ($n^3+O(n^2)$) for NP- NNP still less efficient 1.5 times than NP especially for parallel versions (whose asymptotic speed-up is $2n^2/9$ and the asymptotic efficiency is 2/9).

4 Experimental study of sequential algorithms

This section is devoted to the experimental computation of the several EG sequential versions.

To ensure the constructability of the EG algorithm, we chose 12 different values of N in the range [250, 3000] with a step equal to 250. For each N, the execution time is the mean of five runs. The matrixes are chosen according to the condition: the absolute value of any diagonal coefficient is equal to the sum of all the absolute values of the coefficients belonging to the corresponding row.

We therefore achieved 72 tests for the sequential computation illustrated by Table VII. We mention that the following notations are adopted:

- N : matrix size
- T : execution time (s)

We present excerpts from the different results through table VII, VIII.

4.1 Perfect Nest permutations

On one hand, Fig.1 highlights the cubic complexity for the 6 permutations. On the other hand, we notice from Table VII and Fig.1 that the best versions are KIJ (AXPY-LCC kernel) and IKJ (GAXPY- LLL kernel). IJK and JIK are the worst (DOT kernel). The relative gap between the two extreme versions increases with n, and ranges between 35% and 56%. These results evince satisfactorily that the row access is better than the column one.

TABLE VII. EXECUTION TIME (S) OF THE DIFFERENT EG_NP PERMUTATIONS

N	KIJ	IKJ	JKI	KJI	JIK	IJK
50	**0.084**	0.086	0.097	0.137	0.128	0.129
500	**0.676**	0.704	1.056	1.302	1.232	1.236
750	**2.288**	2.336	3.818	3.847	4.467	4.521
1000	**5.435**	5.560	9.312	9.380	10.960	11.291
1250	**10.629**	10.737	18.256	18.386	21.850	22.739
1500	**18.361**	18.518	31.970	32.513	38.268	39.931
1750	**29.151**	29.448	51.143	51.140	61.370	64.081
2000	**43.516**	44.059	76.292	76.887	92.304	96.408
2250	**61.919**	62.090	109.425	111.801	132.345	138.133
2500	**84.848**	87.000	151.369	155.322	182.849	190.503
2750	**112.891**	114.332	203.576	207.992	245.494	254.966
3000	**146.654**	147.083	267.596	274.396	321.845	332.936

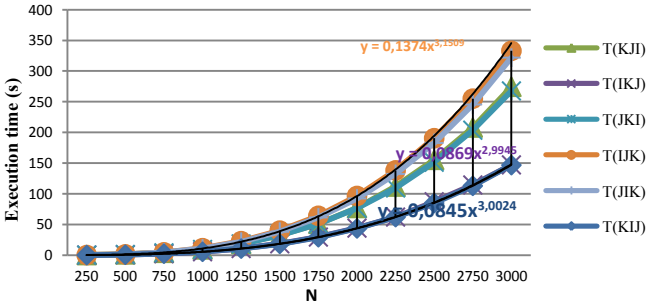


Fig. 1. *Execution time (s) of the EG_NP sequential permutations*

4.2 Imperfect Nest permutations

Table VIII and Fig.2 show that the best permutations are IKJ (GAXPY-LLL kernel) and KIJ (AXPY- LCC kernel). KJI is the worst one (AXPY-CCL kernel). The relative gap between the two extreme permutations increases with n and ranges between 21% and 57%. Thus, this also confirms the efficiency of the row access for this case.

TABLE VIII. EXECUTION TIME (S) OF THE DIFFERENT EG_NNP PERMUTATIONS

N	KIJ	IKJ	JKI	KJI	JIK	IJK
50	**0.057**	0.058	0.063	0.065	0.062	0.072
500	**0.455**	0.463	0.626	0.723	0.671	0.742
750	**1.537**	1.56	2.330	2.581	2.652	2.702
1000	**3.665**	3.703	6.079	6.520	6.606	6.587
1250	**7.185**	7.228	12.209	13.140	13.244	13.200
1500	**12.411**	12.471	21.707	23.166	23.773	23.046
1750	**19.697**	19.798	34.429	37.148	38.231	36.990
2000	**29.375**	29.546	53.037	56.234	57.516	55.980
2250	**41.807**	42.048	77.454	80.937	82.299	80.543
2500	**57.321**	57.637	108.100	112.593	114.298	112.021
2750	**76.286**	76.693	143.394	151.483	153.437	156.626
3000	**99.010**	99.559	185.967	200.352	203.245	231.901

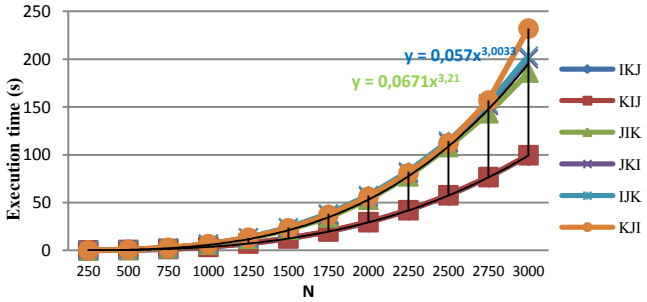


Fig. 2. *Execution time (s) of the EG_NNP sequential permutations*

4.3 Comparative study of EG-NP and EG-NNP

We summarize in Table IX and Fig.3 (resp. Table X and Fig.4) the execution time of both NP and NNP of KIJ version (resp. IKJ), and we define the following notation for the reducing time ratio R:

R(version)= T(version_NP)/ T(version_NNP)

where version ϵ {KIJ, KJI, IKJ, JKI, IJK, JIK}

TABLE IX. COMPARISON OF KIJ_NP AND KIJ_NNP

N	T(KIJ_NP)	T(KIJ_NNP)	R
250	0.084	0.059	**1.42**
500	0.676	0.463	**1.46**
750	2.288	1.560	**1.47**
1000	5.435	3.703	**1.47**
1250	10.629	7.228	**1.47**
1500	18.361	12.472	**1.47**
1750	29.151	19.799	**1.47**
2000	43.516	29.547	**1.47**
2250	61.919	42.049	**1.47**
2500	84.848	57.637	**1.47**
2750	112.891	76.693	**1.47**
3000	146.654	99.559	**1.47**

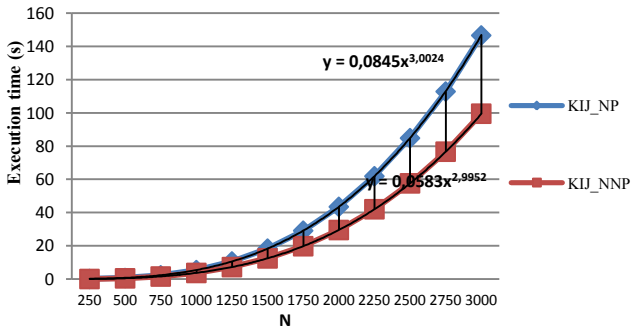


Fig. 3. *Comparison of KIJ_NP and KIJ_NNP*

From Fig.3, we notice that the non-perfect nest (NNP) for the KIJ permutation is faster than the perfect nest (NP) version with an important ratio ranging from 1.42 to 1.47.

TABLE X. COMPARISON OF IKJ_NP AND IKJ_NNP

N	T(IKJ_NP)	T(IKJ_NNP)	R
250	0.086	0.057	**1.51**
500	0.704	0.455	**1.55**
750	2.336	1.537	**1.52**
1000	5.560	3.665	**1.52**
1250	10.737	7.185	**1.49**
1500	18.518	12.411	**1.49**
1750	29.448	19.697	**1.50**
2000	44.059	29.375	**1.50**
2250	62.090	41.807	**1.49**
2500	87.000	57.321	**1.52**
2750	114.332	76.286	**1.50**
3000	147.083	99.010	**1.49**

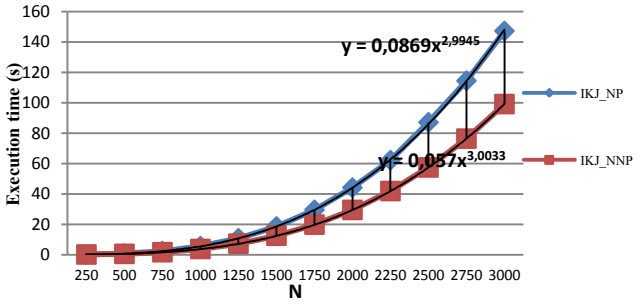


Fig. 4. *Comparison of IKJ_NP and IKJ_NNP*

A comparison between the IKJ-NP and IKJ-NNP versions shows that the Gaussian Elimination algorithm in non-perfect nest (EG_NNP) is almost 1.5 times (1.49-1.52) faster than the EG_NP versions.

The improvement of execution time for the best versions is about 33% (for large n). Thus, these experimental studies of the EG in all its sequential versions confirm the theoretical complexities i.e. $n^3+O(n^2)$ for NP and $2n^3/3+O(n^2)$ for the NNP.

5 Experimental study of parallel algorithms

In this section, we present our experimental study of the parallelization of the EG algorithm with its diverse permutations. Let us mention that our target machine (TM) is an Intel ® Xeon quad-processor 10-cores, CPU E E7-4850 @ 2.00GHz. Its forty cores have dedicated cache L1 with 32 KB size, dedicated cache L2 with 256 KB size. Each ten cores belonging to the same chip share the L3 cache level whose size is 24MB. The RAM size is 94.4 GB. To measure the performance evolution, we compute the sequential execution time (T_{ex}) then the parallel T_{ex} (on p cores). Note that each T_{ex} is the mean of 5 runs expressed in seconds (s). The computed matrixes were generated randomly. Twelve different sizes ranging from 500 to 6000 for several target cores (1, 2, 4, 6, 8, 12, 16, 24 and 32) were launched on our MA. The parallel experimentation was divided into two types: the initial parallelization and the PLP technique where the computation is adapted to the cores number and each parallel loop is split into two successive loops: a parallel then a sequential one.
Excerpts of the obtained results are depicted below (see Tables XI, XII for parallel EG-NP and Tables XIII, XIV for parallel EG-NNP).

5.1 Perfect Nest versions

5.1.1 Initial parallelization

Only the KIJ version of the EG algorithm is retained because it's the best in terms of completion time we obtained from the above sequential study (see section 4.1). This nested loop has an SPP (i.e. K is a sequential loop; I and J are both parallel loops).
Table XI and Fig.6 illustrate the obtained execution time T_{ex}(s) of the parallelization of KIJ-EG in its perfect nest.

TABLE XI. EXECUTION TIME (S) OF THE PARALLEL KIJ_EG_NP

P / N	1	2	4	6	8	12	16	24	32
	T_{ex}								
500	0.64	0.55	0.16	0.11	0.17	0.112	0.09	0.06	0.04
1000	4.74	3.69	1.32	0.89	1.33	0.86	0.68	0.45	0.36
1500	16.00	9.76	4.46	3.03	4.77	2.87	2.22	1.52	1.46
2000	37.90	24.58	10.46	7.19	10.20	6.84	5.60	3.75	3.73
2500	74.27	47.85	20.17	14.01	20.56	13.77	10.61	11.40	6.54
3000	128.7	84.11	41.84	24.07	25.12	24.25	18.46	17.51	11.82
3500	209.6	124.5	61.94	38.27	33.15	38.79	30.61	28.31	21.21
4000	306.5	185.7	81.67	57.17	52.06	56.27	44.73	38.76	33.45
4500	437.8	277.9	118.5	81.49	67.60	76.95	63.90	58.43	44.04
5000	607.8	417.5	162.9	112.1	92.81	109.26	86.87	67.67	56.87
5500	808.7	553.0	219.2	156.0	115.1	149.7	111.7	94.60	80.05
6000	1045	751.5	285.7	195.6	152.8	201.8	156.5	118.7	95.63

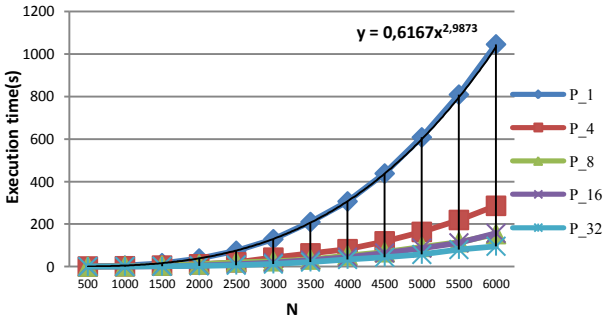


Fig. 5. *Execution time (s) of the parallel KIJ_EG_NP on TM*

From Fig.5, the time evolution curve is an almost cubic: $y=0.6167x^{2.9873}$. This confirms satisfactorily the theoretical complexity of the formula $C(n)=n^3+O(n^2)$. Note that the execution time decreases when p increases (from 1 to 32).

5.1.2 Parallelization with PLP

The PPS type is becoming a SPSPS nest.

TABLE XII. EXECUTION TIME (S) OF THE PARALLEL KIJ_EG_NP_PLP

P / N	1	2	4	6	8	12	16	24	32
	T_{ex}								
500	0.625	0.563	0.260	0.240	0.193	0.110	0.083	0.060	0.051
1000	5.004	4.141	2.330	1.823	1.439	0.816	0.684	0.499	0.421
1500	16.90	15.788	7.526	3.120	4.986	2.629	2.445	1.597	1.718
2000	38.29	35.411	17.728	11.425	12.315	6.475	5.501	3.872	4.134
2500	78.31	69.273	35.701	19.223	24.079	28.148	24.735	28.847	32.579
3000	132.1	137.39	59.861	43.260	41.767	38.118	34.797	44.117	45.437
3500	203.7	180.33	89.819	43.835	63.092	58.247	50.080	62.216	60.119
4000	307.0	243.26	124.39	82.395	95.003	86.042	74.876	83.061	88.080
4500	445.1	350.37	182.91	93.960	131.34	124.26	109.07	112.94	90.640
5000	650.5	412.68	233.41	129.00	185.10	156.62	131.60	138.02	111.53
5500	869.6	678.31	330.05	157.36	246.99	204.00	164.98	170.22	135.06
6000	1116	877.30	421.32	204.81	332.67	257.92	215.04	206.95	148.89

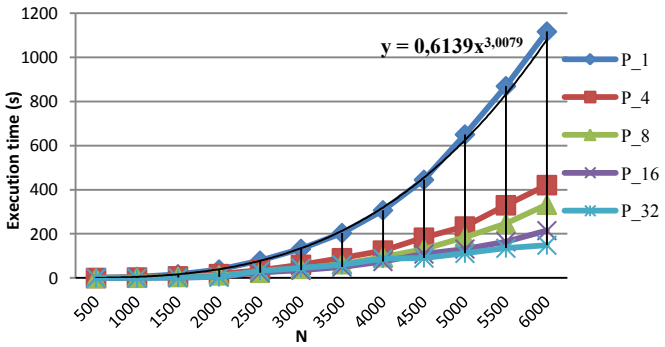


Fig. 6. *Execution time (s) of the parallel KIJ_EG_NP_PLP on TM*

Fig.6 shows the cubic evolution of the execution time of the Gaussian Elimination algorithm (EG) in perfect nest after undergoing the PLP parallelization: $y=0.6167x^{3.0079}$. This proves the theoretical complexity $C(n)=n^3+O(n^2)$. The execution time decreases when p increases.

We notice that the initial parallelization is better than the PLP parallelization.

5.2 Imperfect nest versions

5.2.1 Initial parallelization

We only retain the KIJ version because it was the best sequential permutation of the EG. Its loops are SPP (i.e. K is sequential however I and J are parallel).

We denote its execution time T_{ex} (s) whose variation depends on the matrix size. T_{ex} is illustrated through Fig.7.

We notice that T_{ex} decreases when p increases, except for the case p=12 and p=16 where it is bigger than the T_{ex} obtained for p=8. This overhead is due to the frequent use of the target machine by several users.

TABLE XIII. EXECUTION TIME (S) OF THE PARALLEL KIJ_EG_NNP

P / N	1	2	4	6	8	12	16	24	32
	T_{ex}								
500	0.447	0.253	0.127	0.089	0.074	0.090	0.095	0.050	0.060
1000	3.596	2.008	1.003	0.685	0.528	0.612	0.628	0.291	0.360
1500	12.124	6.865	3.381	2.293	1.737	1.965	1.666	1.375	0.916
2000	28.862	16.27	8.135	5.448	4.174	4.631	3.974	3.085	2.204
2500	56.647	31.854	15.972	10.945	8.478	10.876	8.802	6.112	4.376
3000	98.127	54.99	27.718	19.543	15.140	18.899	13.695	9.809	7.563
3500	168.432	87.712	43.754	30.801	25.343	29.842	20.731	15.880	11.875
4000	233.258	132.536	65.036	45.837	34.328	43.129	33.864	23.081	17.283
4500	343.697	186.634	92.758	62.463	49.190	56.602	50.095	34.113	23.914
5000	480.005	264.973	123.637	86.289	66.953	88.757	85.085	46.388	33.015
5500	627.724	365.722	161.186	120.177	89.589	110.410	98.326	61.882	45.045
6000	851.076	504.729	218.651	169.042	112.361	152.314	137.388	79.840	59.484

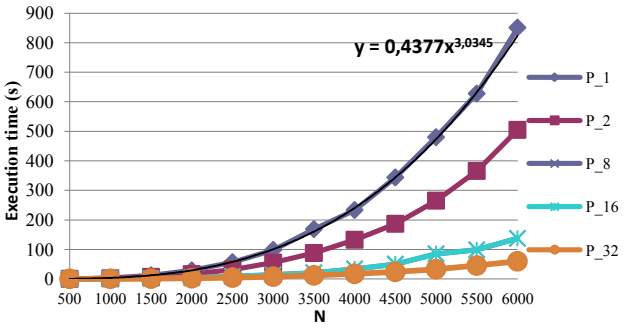


Fig. 7. *Execution time (s) of the parallel KIJ_EG_NNP on TM*

5.2.2 Parallelization with PLP

From Fig.8, the time evolution curve is an almost cubic: $y=0.6167x^{3.0358}$. This confirms satisfactorily the theoretical complexity of the formula $C(n)=n^3+O(n^2)$.

TABLE XIV. EXECUTION TIME (S) OF THE PARALLEL KIJ_EG_NNP_PLP

P / N	1	2	4	6	8	12	16	24	32
	T_{ex}								
500	0.493	0.367	0.253	0.230	0.218	0.139	0.133	0.094	0.054
1000	3.870	2.584	1.956	1.597	1.259	0.656	0.644	0.440	0.414
1500	13.315	11.299	6.083	3.640	4.825	1.861	1.708	1.322	1.179
2000	31.848	23.540	15.533	10.621	10.983	8.751	5.114	2.876	2.845
2500	64.460	48.396	29.359	15.715	19.444	18.146	24.860	7.161	7.778
3000	118.616	73.606	47.782	28.693	36.535	34.370	35.002	14.056	11.697
3500	175.217	113.200	74.444	44.973	52.312	41.790	45.580	19.356	18.594
4000	274.132	156.013	107.271	67.898	73.900	52.495	57.803	25.346	21.558
4500	389.966	226.062	149.905	109.148	103.126	78.191	82.042	39.096	34.168
5000	516.942	321.561	212.205	122.117	145.855	106.716	105.226	50.960	41.347
5500	688.107	436.047	291.428	151.822	183 .538	143.954	124.481	66.468	58.718
6000	899.742	587.527	397.157	204.816	205.275	188.750	167.882	88.545	75.267

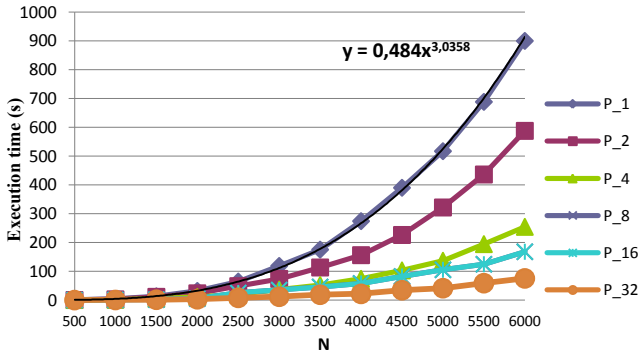


Fig. 8. *Execution time (s) of the parallel KIJ_EG_NNP_PLP on TM*

Note that T_{ex} decreases when p increases (p =1, 2, 4, 8, 16 and 32), except for the case p=8 where it is bigger than the T_{ex} obtained for p=6. This overhead is due to the frequent use of the target machine by parallel users.
A comparison of the initial and the PLP technique shows that the initial parallelization is better than the PLP.
Ultimately, we compare the initial parallelization of the perfect nested loop (NP) (resp. the PLP parallelization) and that of the imperfect nested loop (NNP) (resp. the PLP parallelization), then we conclude that the NNP is almost 1.5 times faster than the NP version.

6 Conclusion

In this work, we carried out a theoretical and experimental study of the Gaussian Elimination (EG) benchmark targeting two shared memory multicore machines: first of all a quad-core bi-processor architecture then a 10-cores quad-processors platform in order to ensure the validation of our contribution and to highlight the code performance improvement through our code optimization on the both versions: sequential and parallel. Notice that we have relied on several optimization techniques and polyhedral loop transformation in the parallelization.
In the first part, our purpose was essentially the improvement of data locality and memory access of the EG as our target polyhedron program and the display of the most efficient computing kernel through, invariant extraction and loop interchange. The second part was devoted to the parallelization, where we adopted the traditional technique relied on the dependencies analysis and some transformations e.g. loop interchange, then the partial partition loop (PLP), in order to improve the parallelism degree.

Then, from the Gaussian Elimination benchmark, we have derived diverse permutations, studied the performance of several sequential (resp. parallel) versions in a perfect (resp. imperfect) nested loop, and compared them.
The obtained results in the sequential case satisfactorily confirmed the theoretical temporal complexity and ensured the importance of array access type and the target computation kernel for all the studied permutations. In the parallel case, the obtained results highlight the interest of the parallelization of both EG-NP and EG-NNP algorithms.

However, we remark that from a certain value of the cores number, the performance degrades relatively.

Several interesting points remain to be seen, particularly: (i) Completion of other experiments on other parallel architectures e.g. GPUs in order to study the scalability, (ii) extension of the theoretical and experimental study to other benchmarks e.g. QR factorization and Cholesky, (iii) study of other PP transformations to extract more parallelism.

Acknowledgment

We would like to thank Pr. Zaher Mahjoub for his valuable help.

7 References

[1] J.R. Allen and K. Kennedy, "Optimizing compilers for modern architectures: a dependence-based approach", book, Morgan Kaufmann Publishers Inc. ISBN "1-55860-286-0", San Francisco, CA, USA, 2002.

[2] D.H. Kim, "ARM NEON Assembly Optimization", Journal of Multidisciplinary Engineering Science and Technology (JMEST), ISSN: 2458-9403, vol. 3(4), April 2016.

[3] H. Venturin, "Le débogage de code optimisé dans le contexte des systèmes embarqués", PhD thesis, University Joseph Fourier, Grenoble, France, October 2008.

[4] Y. Zhao and K. Kennedy, "Computer Scalarization Using Loop Alignment and Loop Skewing", Journal of Supercomputing, Kluwer Academic Publishers Hingham, MA, USA, vol. 31(1), pp. 5-46, January 2005.

[5] Z. Mahjoub and M. Jemni, "Restructuring and parallelizing a static conditional loop", Journal Parallel Computing, vol. 21(2), pp. 339-347, February 1995.

[6] A. Krall and S. Lelait, "Compilation Techniques for Multimedia Processors", International Journal of Parallel Programming, Kluwer Academic Publishers Norwell, MA, USA, vol. 28(4), pp. 347-361, August 2000.

[7] Y. Slama, "Etude de la parallélisation de nids de boucles par le modèle polyédrique ", DEA thesis, University of Tunis el Manar, Tunisia, October 1999.

[8] M. Cosnard and D. Trystram, "Algorithmes et Architectures Parallèles", Inters Editions, July 1993.

[9] P. Delisle, "Parallélisation d'un algorithme d'Optimisation par Colonies de Fourmis pour la résolution d'un problème d'ordonnancement industriel", Master thesis, University Québec in Chicoutimi, June 2002.

[10] V. Louvet, "Performances et Optimisations", Doctoral School MathIf, Institute Camille jordan-CNRS, September 2011.

Parallel Transcoding using the CA-Cloud Architecture

Prof Avinash Shankaranarayanan[1] **and Prof Christine Amaldas** [2]
[1]Royal Melbourne Institute of Technology (RMIT) International University
[2]Ritsumeikan University, Japan

Abstract - *These days, video processing and content delivery are being expended in more ways than the over assortment of systems available including streaming and storage of varying formats and codecs. Transcoding is the process of interpreting or changing over one coded signal representation to another. However, more often than not transcoding turns out to be computationally expensive due to the sheer volume of data generated in real time. Because of this computational complexity, one needs to sort for other distributed mechanisms including Clustering, Grids and the recently evolving Cloud computing platforms available for distributed transcoding. The rest of the paper makes a comparative analysis of the singular node processing x265/HEVC codec based transcoding to that of samples transcoded using the CA-Cloud architectural framework wherein additional computational assets accessible among numerous machines, multicore CPUs, and circulated processing assets which are evaluated here are compared to that of the CA-Cloud architecture previously researched by the authors.*

Keywords: *CA-Cloud Architecture, Distributed Computing, Transcoding, x265/HEVC.*

1 Introduction

In this milieu, video content is being produced, transported and consumed in more ways through a variety of devices including mobile phones, tablets, PCs, etc., than yester years. Meanwhile, a seamless interaction between video content production, transportation and consumption is taking place in majority of the devices. The difference in the device, the network and the video representation categories results in the necessity for a unified mechanism for video content adoption.

Transcoding is one such mechanism utilized here. Video transcoding is a process of converting an independent signal representation of a given video to another. At present, transcoding is being applied for such drives as: bit-rate reduction (in order to meet network bandwidth availability); resolution reduction (for display size adoption); temporal transcoding (for frame rate reduction); and error resilience (transcoding for insuring high QOS) [23][26].

Transcoding is a computationally heavy process and numerous methods has been proposed in order to increase its efficiency [18] [28]. Among them, many attempts have been made to decrease its computational complexity by reusing information like DCT coefficients and the motion vectors extracted from the original coded data instead of fully re-encoding the video content. To realize multiple transcoding and speed up, studies has been done to integrate multiple processors to fully decode and re-encode incoming video [21].

However, there has been limited research work done in the analysis of how distributed transcoding architecture could be utilized. In lieu of real-time transcoding where quality of users' experience: startup time and jitter free video experience, matters a lot. Furthermore, the advances in the utilization and economics of the distributed computing has widened such possibilities, namely, elastic computing in the cloud which requires a different type of distributed transcoding architecture [14][29].

In this paper, we will try to propose a possible scalable distributed architecture as previously researched by the authors: a transcoder designed with the cloud in mind and evaluate its implementation performance for x265/HEVC video transcoding.

2 Literature

In this section, we will commence with a brief overview of the video compression. This is crucial as it raises the knowledge of the fundamental concepts associated with it as deemed necessary especially when it comes to proper partitioning and scheduling of a video stream on a distributed platform for transcoding. It is also useful for understanding the underlying concepts that make up a video transcoder [12][25][13]. Moreover, a summary of video transcoding and its associated theories will be presented giving an overview of the current practices.

2.1 Video Compression

Row video contains a substantial amount of data. Communication and storage capabilities are inadequate and hence, it becomes more exorbitant. For example a given SD/HD video signal might have the following resolution rates as illustrated in Figure 1: 720x576/frame, 1280x720 and 1920∗1080 pixels/frame, respectively.

A playback speed of 40frames/sec with 3 colors namely RGB and 8bit color information is displayed on a standard HDTV depending on the technology utilized including LCD, LED,

OLED and other technologies. This produces an information flow of 398.13 MB/sec, 884.74 MB/sec and 1990.66 MB/sec increasing the bandwidth needed dramatically for higher pixel display as calculated in Figure 2.

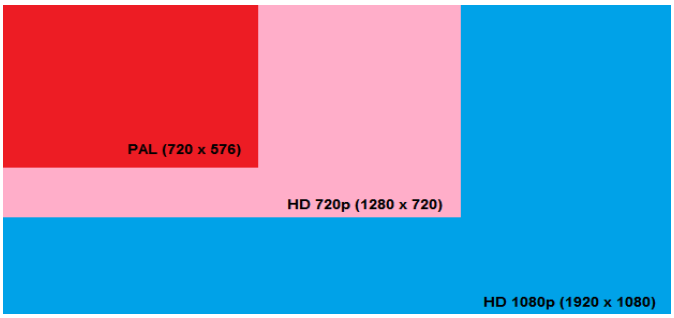


Figure 1: SD/HDTV Native Resolution & Transmission Rates

Pixel Frame Rates

$$\left(\frac{720\ X\ 576\ pixels}{Frame} + \frac{40\ Frames}{Sec} + \frac{3\ Colors}{Pixel} + \frac{8\ Bits}{Color}\right) = 398.13\ \text{MB/sec}$$

$$\left(\frac{1280\ X\ 720\ pixels}{Frame} + \frac{40\ Frames}{Sec} + \frac{3\ Colors}{Pixel} + \frac{8\ Bits}{Color}\right) = 884.74\ \text{MB/sec}$$

$$\left(\frac{1920\ X\ 1080\ pixels}{Frame} + \frac{40\ Frames}{Sec} + \frac{3\ Colors}{Pixel} + \frac{8\ Bits}{Color}\right) = 1990.66\ \text{MB/sec}$$

Figure 2: Pixel Frame Rates Calculated for a typical Video Source

Comparing these results for a channel with a bandwidth of 50Mb/sec will require the video to be compressed by a factor of about 18. The way this is achieved is through video compression which is a typical NP complete problem with relevance to distributed computing. Video compression is done through reduction of redundancy and irrelevancy [12][15][13].

Redundancy reduction involves typically the use of consecutive frames in a video signal which are highly correlated and exhibit temporal redundancy. It usually contains the same content with the only difference in placement (movement) of objects that makes up the content. Redundancy also exists inside a single frame due to the fact that neighboring pixels exhibit some sort of correlation and this type of redundancy is called spatial redundancy. Video compression utilizes these two sources of redundancies.

Irrelevancy reduction on the other hand has all the required information in a video signal which is not perceptually important. The human brain perceives the world in a certain way and this fact can be utilized to reduce some irrelevant information that is unimportant [12][15][13][10].

2.2 Transcoding and Codecs

On one hand, transcoding is the process of encoding or compressing specific frames with the chosen bit rate. Codecs on the other hand, enable the decoding process while taking care of performance versus quality that is specific to the hardware and software being deployed for viewing. Although codecs comprises of filters for both audio and video streams, our focus of the discussions includes both with relevance to HEVC standards.

The HEVC codec have been an adoption from its previous incarnation of the x264 format (very similar to MPEG/2 and H.264 standards with relevance to encoding perspectives) with variable bitrates and unique frame rates for processing. High level compression is achieved through the use of smart decision making algorithms when encoding with HEVC tweaks applied to I-frames and B-frames intervals, data rate selection, bitrate control technique (VBR, CBR) and most essentially, video resolution [31].

Most advanced codecs have various options for tweaking (spanning zillions of configurations made possible) which can lead to de-standardization of the adopted codec followed by issues pertaining to playback on many devices and playback environments. To avoid these pitfalls, several presets have been built around these codecs to enable simplified operations for end users (encoders). These include various software encoders including FFMpeg and x265 which has default presets created to avoid recreation of trade-offs between encoding time and output quality.

We have chosen the FFMpeg encoder for optimization within the CA-Cloud environment with storage and stripping of datasets occurring transparently within the cloud environment.

With FFMpeg specific preset were utilized and customized for utilizing the HEVC encoder by inserting command-line arguments like CRF selection, which ranges from 1 (Fastest Encode/Poorest Quality) to 30 (Slowest Encode/Best Quality) with FFMpeg 25 being the default setting. When using either codec(x265/HEVC), the most ideal setting was selected for determining which preset was relevant during single node encode and parallel encoding operations.

The next section focusses on how distributed encoding takes place leading to the CA-Cloud encoding methodology.

4.2 Distributed Transcoding and Codecs

As opposed to the classic distributed systems, in distributed transcoding systems, it is required that these systems should be able to deal with a stream of unbounded data and certain time constraints.

Data stream processing is a way to support real-time processing of continuous data streams. Typical scenarios includes multimedia processing and sensor data analysis from large sensor networks. These applications need low latencies for real-time responses. Usually stream data rates vary in time.

How to insure some real-time stream processing is one of the key technical challenges in the area of distributed stream processing systems. One of the simple solutions is to add new physical nodes to guarantee that the overall system performance is adequate to handle the largest possible burst of data.

However, various problems often prevent the use of this solution, for example, even if we can solve the problems and provide sufficient computational resources to handle the highest possible data rate, most of those resources may be wasted if the normal data rate is/becomes low. Fortunately, the emergence of cloud computing as an extension of distributed computing gives us the chance for an utility type of computing and essentially solves both problems of over provisioning and under provisioning [29, 31].

Cloud computing is a new paradigm in distributed computing in which computing is offered as a utility by third parties whereby the user is billed only as per their consumption in a service oriented manner. In the next section, we shall see what the cloud means and how it has a role in distributed computing and particularly, to video transcoding.

3 Transcoding on the CA-Cloud Arch

Communication over different types of networks becomes effective only when the overall latency of the data transmissions are effective. If the transmission or transfer rate is higher, then, process migration is ineffective and the process is best suitable for local processing. Most of the current Cloud infrastructures face more or less, a similar bottleneck [21].

Hence, it becomes cumbersome to build a global Cloud infrastructure that can effectively use distributed computing as a base to migrate processes from different parts of the globe for effective distributed computing and parallel executions of program threads. Instead of approaching a theoretic global infrastructure, the CA-Cloud utilizes a local approach that is more cost effective and would drastically bring down the idle time of systems operating at the selected geography. Another problem faced by most researchers, is the use of Cloud service and its related infrastructure.

The CA-Cloud [21] works on the principle of the power server model of computing. Each of the clients, runs the CA-Cloud server which is a simplistic http web server running services based on common scripting languages such as Perl, Hypertext Pre-Processor (PHP) or Common Gateway Interface (CGI). The client side coding model enables the developer to upgrade the services using the CGI framework which can use any of the languages that support CGI scripting.

Figure 3: Transcoding on the CA-Cloud Architecture

For the sake of simplicity and rapid development of services, we have used Perl as the default language of choice due to its availability and portability to most platforms. Different components are utilized for parallel/distributed video transcoding using the CA-Cloud Architecture as shown in Figure 3 with the results obtained for our sample video files (as discussed in the next section).

4 Results

For these series of tests, we have applied, five 10-second video files of varying and constant resolutions including 1080p content, ranging from a low-motion, low-detail to high-detail and high-motion shot videos with a High resolution Samsung camera. The varying content were applied to stress the codecs in different scenarios, while the short to medium duration of the video clips helped us to run numerous discrete encodes with both singular nodes and on our cloud platform with load balancing enabled by default.

A typical 1080p resolution test file was encoded using the FFMpeg transcoding software with a HEVC profile at 4Mbps per second, encoding to 1-pass CBR with a key frame interval of 40 frames per second as shown using our Figure 4 results. Each codec was tested using five standardized presets namely the: Ultra Fast, Very Fast, Medium, Slow and Placebo with CR rates ranging from CRF 23 to 28 on average for individual single core node performance to parallel processing of the same job over multiple nodes.

The x265/HEVC output files were produced via command line using the FFMpeg version 3.0.2 x265 video encoder over the selected nodes. With all encodes, the encoding rates were

verified to check if the encoder met the target rate, which in comparison to Handbrake was significant. With FFMpeg, we tested CRF values 18 (lowest quality / fastest encode), 22 (the default), 23, 25, and 28 (slowest encode / highest quality) while keeping other variables constant. The idea was to understand the qualitative versus performance trade-offs associated with the various presets with both single node and parallel settings initially undertaken during this research. The summary of the results can be viewed in Table 1 and Table 2 comparing single node vs cloud based transcoding using the x265/HEVC codecs used during the transcoding of the selected video clips.

Table 1 : FFMpeg x265/HEVC Results from Single Node

Settings	Avg Encoding Time	% Increase in Encoding Time	Avg CR Score
Ultra Fast	2.4	0%	1.3
Very Fast	4.3	76%	2.4
Medium	7.4	66%	4.1
Slow	92.4	1,390%	5.1
Placebo	321.6	287%	1.8
Total		1,819%	

Table 2 : FFMpeg x265/HEVC Results from CA-Cloud

Presets	Avg Encoding Time (CA-Cloud)	% Increase in Encoding Time (CA-Cloud)	Avg CR Score
Ultra Fast	2.1	0%	1.2
Very Fast	3.3	66%	1.8
Medium	4.4	56%	2.4
Slow	56.6	990%	3.1
Placebo	196.4	112%	1.1
Total		1,224%	

This was then compared to the scores received from encoding using the CA-Cloud Architecture based on two variables: duration and quality. The Average Encoding Time column enumerates the average encoding time for all the selected clips encoded using the presets as identified previously followed by the average percentage scores. Likewise, the percentage of Increase in Encoding Time Column shows the increase in time between the selected presets along with the differences representing the total change from CRF 23 to CRF 28.

By changing from CRF 28 to CRF 23, we were able to generate on average, a 22 percent improvements towards quality over five of the test video clips, while the encoding time rose to a whopping 73 percent. Likewise, encoding at CR 23 created a total quality difference of 55 percent between the two extreme presets while increasing the encoding time by 52 percent. This makes CRF 23 an interesting default preset for many encoders of this transcode.

With the new x265 codec, the following presets were tested as follows the: Ultra Fast, Very Fast, Medium (Default), Slower, and Placebo presets which resulted in a quality drop noticably from the Ultra Fast to Very Fast, and the Ultra Fast to slightly Medium in quality (as shown in Figure 4 and 5).

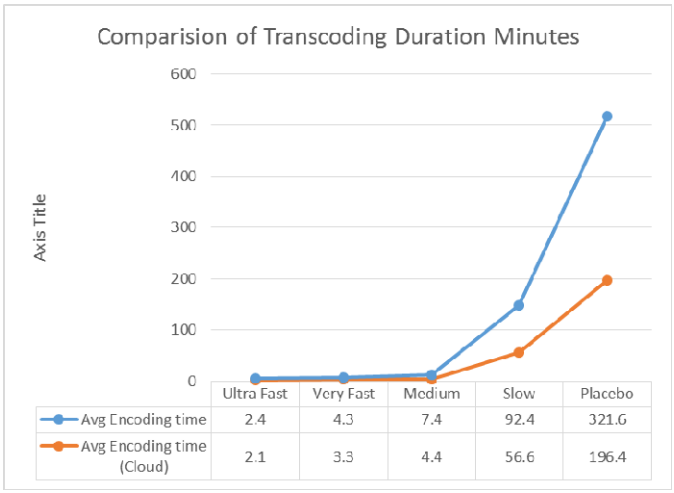


Figure 4: Transcoding time on CA-Cloud Virtual Machines.

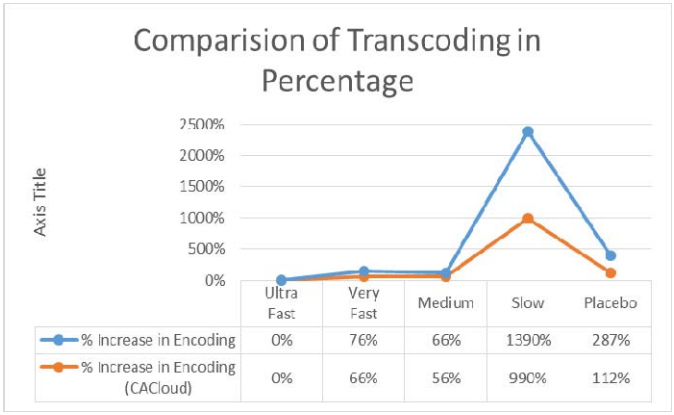


Figure 5: Percentage of Transcoding time on CA-Cloud Virtual Machines.

5 Conclusion

Currently, the advances in distributed computing has provided us with new ways of utilizing computational resources. Cloud computing which is one form of distributed computing offers special features such as delivering high grade distributed computing infrastructures as a utility. This means that any user can obtain an easy access to computational resources matching his demand on "pay as you use" basis. The idea behind distributed computing approaches for transcoding was due to the fact that resources were insufficient over singular nodes. However, proper utilization of resources without over provisioning is also vital, hence,

the cloud provides for the required elasticity needed by a distributed transcoder.

The distributed transcoder was made using the existing transcoder implementation tool, FFMpeg. This tool is used as a process running on several nodes that are distributed across a network. In addition, the CA-Cloud Architecture was used to enable both the communication and process management mechanisms providing most of the required communication protocols needed for proper scheduling and the management of distributed tasks.

6 Acknowledgments

I would like to take this opportunity to thank Royal Melbourne Institute of Technology International University for funding this research in the form of a Research Grant and Professional Development International Travel Grant which was received by the first author in June 2016 for outstanding research.

7 References

[1] Folding@home: A distributed computing project which studies protein folding. http://folding.stanford.edu/. Accessed: Jan 20, 2012.

[2] Fujitsu MB86H52: ASIC MPEG-2 to H.264 HD transcoder from fujitsu. http://www.fujitsu.com/downloads/MICRO/fme/documentation/m13.pdf. Accessed: Jan 20, 2016.

[3] The MPI standard: A specification for MPI implementations. http://www.mcs.anl.gov/research/projects/mpi/. Accessed: Jan 20, 2016.

[4] Starcluster: An open source cluster computing tool kit for amazon's EC2. http://web.mit.edu/stardev/cluster/index.html. Accessed: Jan 20, 2016.

[5] MPI: The complete reference. http://www.netlib.org/utk/papers/mpibook/mpi-book.html, 1996. Accessed: Jan 20, 2016.

[6] Ffmpeg: Cross platform solution to record and convert and stream audio and video. http://ffmpeg.org/, Dec 2000. Accessed: Jan 20, 2016.

[7] Big Buck Bunny: A short computer animated film by the blender institute. http://bigbuckbunny.org, Apr 2008. Accessed: Jan 20, 2016.

[8] Amazon elastic compute cloud: Delivers scalable pay-as-you-go compute capacity in the cloud. http://aws.amazon.com/ec2/, Nov 2010. Accessed: Jan 20, 2016.

[9] Cloud Software Finland: Finland national cloud software program. http://www.cloudsoftwareprogram.org/, 2010. Accessed: Jan 20, 2016.

[10] ISO/IEC 144 6. Information technology a coding of audio-visual objects.Technical report, 2003.

[11] Jokhio Fareed Ahmed, Lafond Sebastien, and Lilius Johan. Analysis of video segmentation for spatial resolution reduction video transcoding. In Proceedings of the International Symposium on Intelligent Signal Processing and Communication Systems, 2011.

[12] J.G. Apostolopoulos and S.J. Wee. Video Compression Standards. Wiley Encyclopedia of Electrical and Electronics Engineering, JohnWiley and Sons, Inc., New York, 1999.

[13] John G. Apostolopoulos, Wai tian Tan, and Susie J. Wee. Video streaming: Concepts, algorithms,and systems. Technical report, HP Laboratories Palo Alto, Sep 2002.

[14] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph,

[15] Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson,

[16] Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, Feb 2009.

[17] V. Bhaskaran and K. Konstantinides. Image and Video Compression Standards: Algorithms and Architectures. Kluwer Academic Publishers, Boston, Massachusetts, 1997.

[18] Hamann C.J., Roitzsch M., Reuther L., Wolter J., and Hartig H. Probabilistic admission control to govern real-time systems under overload. In 19th Euromicro Conference Real-Time Systems, pages 211–222, 2007.

[19] Andrey Khorlin. Scheduling in distributed stream processing systems. Master's thesis, 2006.

[20] G. Morrison. Video transcoders with low delay. In IEICE Trans.Communication, pages 963–969, 1997.

[21] Shankaranarayanan, Avinash. and Amaldas, Christine, (2013), The Art of Cloud Computing for Businesses, Aries Media Publishers, ISBN: 978-981-09-0152-3.

[22] Michael Roitzsch and Martin Pohlack. Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video. In Real-Time Systems Symposium,

2006. RTSS '06. 27th IEEE International, pages 271–280, 2006.

[23] Michael Roitzsch and Martin Pohlack. Video quality and system resources: Scheduling two opponents. In Journal of Visual Communication and Image Representation, pages 473–488, 2007.

[24] Yasuo Sambea, Shintaro Watanambe, Dong Yu, Taichi Nakamura, and Naoki Wakamiya. High-speed distributed video transcoding for multiple rates and formats. In IEICE TRANSACTIONS on Information and Systems, pages 1923–1931, 2005.

[25] Gary Shao. Adaptive scheduling of master/worker applications on distributed computational resources. Master's thesis, 2005.

[26] A. Vetro, C. Christopoulos, and H. Sun. Video transcoding architectures and techniques: An overview. In Signal Processing Magazine, IEEE, pages 18–29, 2003.

[27] Clemens C. W, Liesbeth Steffens, Reinder J. Bril, and Wim F.J. Verhaegh. Qos control strategies for high-quality video processing. In 15th Euromicro Conference on Real-Time Systems, 2007.

[28] J. Watkinson. The MPEG Handbook. Focal Press, Woburn, Massachusetts, 2001.

[29] Jeongnam Youn, Ming-Ting Sun, and Jun Xin. Video transcoder architectures for bit rate scaling of h.263 bit streams. In Proc. ACM Multimedia, pages 243–250, 1999.

[30] Andreopoulos Y. and Van der Schaar M. Adaptive linear prediction for resource estimation of video decoding. In Circuits and Systems for Video Technology, IEEE, pages 751–764, 2007.

[31] Encode to HEVC, https://www.viastream.com.au/, Accessed: Jul 1, 2016.

Prof Avinash Shankaranarayanan is currently working as an Academic/Consultant teaching Accounting and Finance, Business Communication and Computing, Internet of Things, ICT, International Human Resource Management and Financial Risk Management courses at RMIT International University. He is also an Adjunct Professor at VELS University, India; Trier University, Germany; and the Institute for Applied Sciences (IFaS), Germany. Avinash has been actively involved in publishing several research papers and periodicals in International Conferences and Journals of high standards such as the IEEE and ACM. His research and teaching interests includes: Applied Social and Cognitive Psychology in Education; Game Dynamics; High Performance Grid Computing; Bioinformatics; Material Flow Management; Renewable Energy Systems; Policy and Decision Making. Avinash comes from a diverse discipline of Engineering and Social Sciences and is fluent in a wide range of scholarly domains specializing in Higher Education and Engineering. He serves as an Editor and Steering Committee member on numerous research entities including, the Journal of the Environmentalist and Biologica.

Prof Christine Amaldas is currently teaching core Business and IT technologies at Ritsumeikan while serving as a visiting Professor and Consultant for numerous organizations and international tertiary institutions spanning the Asia Pacific region. As an expert researcher and academic, she has been the author of several Books, Chapters, Conferences and Journal periodicals. She has Chaired and presided over a number of Conferences and has given numerous Keynote speeches and Guest talks. She specializes in Asia Pacific Studies, Education, Information Security and Fraudulence, Ethics and Ethical Governance in ICT, Holistic Medicine and Energy Healing and Governance (Corporate, Public, IT and Dynamic). In 2010, she was the first person to receive the 2010 Teaching and Research Award from the Royal Melbourne Institute of Technology University, Australia for her excellence in teaching and research methodology. Moreover, she was the recipient of the prestigious Best Paper Award at the ANNA World Congress 2011, Chennai, India. Furthermore, she was the recipient of the prestigious '2012 Young Scientists Award' from ASTER, Japan and has received multitude of Grants and Awards since 2012. She is currently busy writing many articles and books in multidisciplinary areas of research.

Towards Efficient Mapping on Multicore processors according to Cache sharing

Emna Hammami, Yosr Slama, Wafa Benboubaker
University of Tunis El Manar, Faculty of Sciences of Tunis,
University Campus - 2092 Manar II, Tunis, Tunisia

Abstract - *The multicore architecture evolution requires updating the automatic parallelization, taking into consideration the intrinsic features of the target parallel architectures. In this context, we focus on parallel polyhedron programs (PP) i.e. programs structured in nested loops with affine bounds on which we aim at adapting the mapping to multicore architecture. In fact, thread placement is a technique widely used on parallel machines to reduce the overall communication time. For instance, two threads which communicate frequently are mapped close to each other. Finding the optimal mapping between threads and cores in a shared-memory environment is a complex task due to implicit communication. This paper presents a brief state-of-the art of the well-known mapping techniques based on hardware characteristics of the target platform and introduces a new theoretical model for predicting the cache sharing effect on the target architecture which can be a source of performance improvement. Besides, our proposal tries to adapt the PP mapping taking into account the cache memory sharing. Ultimately, we carry out a series of experiments targeting a quadcore bi-processor machine in order to evaluate the interest of our contribution.*

Keywords: Cache sharing, data locality, mapping, multicore architecture, nested loops, thread.

1 Introduction

Several studies have been done on program parallelization for multicores. To improve this trend, it is necessary to define a new model that allows adapting the parallelization to the variety of constraints imposed by multicore processors. Thus, exploiting such architectures becomes a main subject of lots of research. That revolves around studying the effects of hardware topology and intrinsic characteristics of contemporary machines on program implementation as well as adapting programming to the recent platforms. Indeed, there are new constraints in multicore architecture related to the different affinities that may exist between some cores sharing one or more than cache memory level.

It is in this context that we aim at studying cache effects on a multicore architecture when mapping tasks of a given parallel program to the available cores. In particularly, we take a special interest in polyhedron programs (PP); i.e. programs structured in nested loops with affine bounds, which are the most used codes in scientific computing. Thus, given a parallel PP, which does not take into consideration the target architecture, we propose to study the adaptation of tasks mapping with taking into account some hardware characteristics of the target multicore machine (TM), e.g. cache sharing, cache memory organization, etc.

The remainder of the paper involves four sections organized as follows. In section 2, we present a brief state-of-the art on the most known methods adapting the mapping to multicore architectures. Section 3 is devoted to the description of the proposed approach, where we detail an experimental study on a multicore parallel platform that allows our theoretical contribution validation and evaluation. Finally, we conclude and present some perspectives in section 4.

2 Related work

Studying the processors coordination, the cores position, the memory size and the cache levels organization, is essential to improve application performance and to optimize completion time. Indeed, when two tasks are accessing to the same data (D), it would be much better if they shared D on the same cache memory (L2 or L3) rather than fetching it through the main memory. Moreover, when tasks require intensive memory access simultaneously, it is preferred to place them on separate chips. Each one is linked to the memory and takes advantage of the maximal bandwidth. In the following, we present a summary of existing works based on architectural hardware features to find the best mapping.

In the literature, Thekkath and al. [1] proposed a placement algorithm fed by trace-driven simulators then evaluated the impact of thread mapping on multithreaded platforms. However, the obtained results don't show performance improvements due to the memory access patterns of the applications.

From the previous works, we notice that, in one hand, the problem of finding the optimal mapping between processes and processors for optimizing parallel programs in distributed memory environments is almost solved by constructing the task graph where vertices represent communication is straightforward. In fact, the obtained communication pattern which represents the exchanged messages gives enough

information about the sender and the receiver, so that is useful to calculate the amount of data exchanged between tasks to be considered in the optimization of the placement. In other hand, Klug and al. [2] proved that the behavior of SPEC OMP programs is very depending on the thread-core binding. In fact, they implemented a framework which uses the hardware counters to extract the best binding between threads of a running parallel application and processor cores in a shared memory system. In addition, the study of Rodrigues and al. [3] justified that optimizing thread placement on multicore machines improves well the codes performance.

However, although the communication is implicit in shared memory (SM) applications, the previous mapping methods in MPI environment could be adapted to be used in SM platforms in order to improve the quality of OpenMP codes. Indeed, Diener and al. [4] proposed a mechanism examining data sharing patterns between OpenMP threads in different workloads then used those patterns in a similar way as messages are used to map processes in cluster computers. In fact, such mechanism transforms memory accesses from different threads to communication patterns, regardless of cache parameters. It allows placing threads that share data on cores that share levels of cache. Targeting two different multicore architectures, such mapping approach achieved considerable improvements reducing execution time by up to 45%. Besides, Chen and al. [5] proposed to automatically detect the optimized mapping using a profile-guided method for SMP clusters and multiclusters which is able to minimize the cost of point-to-point communications for arbitrary MPI applications.

Moreover, many works depend on architectural topology to optimize data dependencies, i.e. residual inter-cores communications. In this context, by gathering information on the hardware distributed-memory architecture and the implementation of the communication model, J. Clet [6] proposed a match between the ranks of the MPI processes and the target cores. Such method is not dependent on any MPI application. It analyzes different mapping strategies applied to NAS tests. J. Clet noticed that increasing the communication rate on the same node is not enough to improve the application performance. However, a good cache management can optimize the target application. The proposed work chooses the amount of data (AD) exchanged between processes as discriminating criterion, computes the AD exchanged for each pair of processes and determines the communication scheme. The obtained pattern is represented by a matrix. Then after generating, by the Hwloc tool [7], a matrix representation of the target topology whose the values are the distance coefficients between the cores, the process mapping is deduced by performing a static correspondence between the two obtained matrixes. In other terms, each process is assigned to a close core of the cores executing the most interacting processes in order to enhance data locality [8] and to improve the overall performance of the MPI application communication.

To conclude, we notice that most of these related works, though effective in placing processes according to their communication patterns in a distributed-memory environment, the problem of detecting the best mapping of OpenMP threads in a shared-memory application is still a large subject of recent research [9] due to the architecture evolution. Subsequently, through this paper, we aim to present our approach for optimizing the quality of a parallel polyhedron program by taking into account some physical characteristics of the TM and choosing the best placement of OpenMP threads on the target cores. For this purpose, based on a selective study of hardware architecture detection tools, we have chosen the Hwloc package [7]. Indeed, it allows displaying a portable abstraction of the hierarchical topology of multicore architectures with gathering information about NUMA memory nodes, sockets, shared caches, cores and simultaneous multi-threading.

3 Proposed approach

In this section, we present our theoretical constructive contribution targeting a multicore architecture based on matching the parallel program's behavior to the cache memory sharing (SCM) in order to enhance data locality as well as data reuse. A series of experiments was performed on the chosen target architecture to evaluate the interest of our approach.

3.1 Theoretical study

In this work, given a program containing dependencies (see Nest 1.), we aim at defining a task mapping policy (tasks here are loop iterations) on the available cores, which takes into account the target multicore architecture, in particular the cache hierarchy. Indeed, when two iterations access the same data, the placement should consider the shared memory between cores.

In this paper, we try to affect dependent iterations which can't be run on the same core on cores sharing at least a cache level. Thus, we have taken into consideration some hardware features e.g. cache sharing and cores organization. In other words, we try to adapt the placement of the parallel program to the cache hierarchy, by imposing the physical choice of cores during the program compilation.

Indeed, the original placement (e.g. with OpenMP), launches threads that are randomly allocated on physical cores without taking into account the architectural topology. However, changing the threads distribution on the different cores may reduce data exchanges and thus improve data locality as well as data reuse. For these reasons, it is important to appropriately choose the allocation of parallel tasks on the cores and consider the SCM in order to reduce as much as possible cores communications (which represent dependencies between threads). Such optimization could be applied to any cache hierarchy. However, in this paper, we target the following multicore architecture: a multicore

machine with n cores (C1, C2, C3,.., Cn) where each pair shares a cache memory (see Fig 1.).

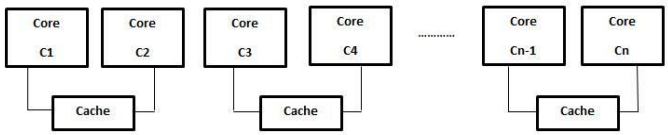


Fig. 1. *Target Multicore architecture (MA)*

In the following subsections, we study nested loops containing dependencies and we are going to choose the best task mapping depending on cache memory sharing (SCM) through a theoretical and an experimental study. In fact, our method aims to improve the quality of the parallel code and it is evaluated through a series of experiments.
Let us consider the following program (see Nest 1) which is a two nested loops with a single dependency distance (q, r).
We mention that DV denotes the dependency distance vector.

Nest1. (A, n)
FOR *i=1,n*
FOR *j=1,n*
*A[i , j]=A[i +**q** , j + **r**]*
ENDFOR
ENDFOR

We take a special interest in the following particular cases according to the values of n, q and r:

- **n = 4; q = 0; r != 0 :**

When (r = 1), we get the parallel code of the below nested loops N1.

N1. (A, n)
FORALL *i=1,n*
FOR *j=1,n*
*A[i , j]=A[i , j + **1**]*
ENDFOR
ENDFORALL

N1 contains a dependency by the j loop, with a dependency distance vector DV equal to (0.1). Thus, the i loop is parallel. We notice from Fig.2 that all the threads P1, P2, P3 and P4 are independent, so the placement of any one on any core does not generate inter-communications. Consequently, the mapping relying on cache hierarchy is useless in this case and can't be adapted to the physical core organization.

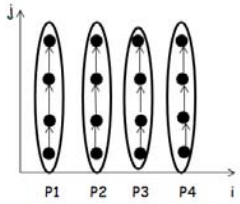


Fig. 2. *Iteration space of the nest N1 (q = 0; r = 1)*

- **n = 4; q != 0; r = 1 || r = -1 :**

From the nest (N2), the dependency distance vector DV is:

$$DV = (1,1)\ if \begin{cases} q > 0\ and\ r = 1 \\ or \\ q < 0\ and\ r = -1 \end{cases}$$

or

$$DV = (1,-1)\ if \begin{cases} q > 0\ and\ r = -1 \\ or \\ q < 0\ and\ r = 1 \end{cases}$$

N2. (A, n)
FOR *i=1,n*
FORALL *j=1,n*
*A[i , j]=A[i + **1** , j + **1**]*
ENDFORALL
ENDFOR

According to the iteration space of N2 (see Fig.3), it is hard to find a suitable thread mapping that influences the quality of a such parallel nested loops . Indeed, the Pi thread is still dependent on Pj (when i<j). Therefore, there is no adaptation to the cache hierarchy in this case because of the continued communication between cores even they share or not the same memory cache.

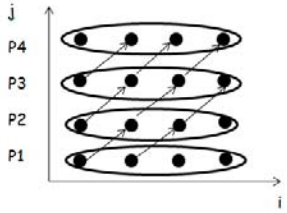


Fig. 3. *Iteration space of the nest N2 (q = 1; r = 1)*

- **n = 4; q != 0; r = 2 || r = -2 :**

The dependency direction vector when (q= 1) is the following:

$$DV = (1,2)\ if \begin{cases} q > 0\ and\ r = 2 \\ or \\ q < 0\ and\ r = -2 \end{cases}$$

or

$$DV = (1,-2)\ if \begin{cases} q > 0\ and\ r = -2 \\ or \\ q < 0\ and\ r = 2 \end{cases}$$

N3. (A, n)
FOR *i=1,n*
FORALL *j=1,n*
*A[i , j]=A[i + **1** , j + **2**]*
ENDFORALL
ENDFOR

From Fig.4, we notice that it is better to involve dependent threads on cores sharing a cache memory in order to reduce the communication rate. So, we propose to map the threads P1 and P3 respectively to the cores C1 and C2, then to map P2 and P4 respectively to C3 and C4.

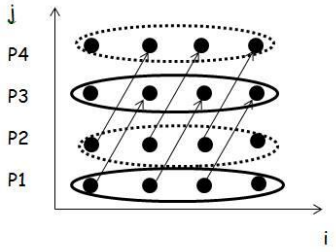


Fig. 4. *Iteration space of the nest N3 (q = 1; r = 2)*

As there is a dependency between P3 and P5, they shall be affected to the same core C2. Similarly for the thread P6 which depends on P4, so it will be allocate on C4.

P7 and P8 shall be mapped respectively to C1 and C3 in order to ensure data locality (see Fig.5).

Core C1	Core C2	Core C3	Core C4
P1	P3	P2	P4
P7	P5	P8	P6
P9	P11	P10	P12
P15	P13	P16	P12
P17	P19	P18	P20
.	.	.	.
.	.	.	.
.	.	.	.

Fig. 5. *Mapping threads of N3 on cores (n=4)*

In this case, we can divide the OpenMp threads into 4 groups as follows:

$Group\ 1 = \{Pi/\ i = 8k+1, k \geq 0\}\ U\ \{Pi/\ i = 8k+7, k \geq 0\}$
$Group\ 2 = \{Pi/\ i = 8k+3, k \geq 0\}\ U\ \{Pi/\ i = 8k+5, k \geq 0\}$
$Group\ 3 = \{Pi/\ i = 8k+2, k \geq 0\}\ U\ \{Pi/\ i = 8k, k > 0\}$
$Group\ 4 = \{Pi/\ i = 8k+4, k \geq 0\}\ U\ \{Pi/\ i = 8k+6, k \geq 0\}$

We precise that the threads of Group1 (resp. Group2, Group3, Group4) are allocated respectively on the core C1 (resp. C2, C3 and C4).

- **n = 4; q != 0; r = 3 || r = -3 :**

The dependency distance vector DV of the bellow N4 nested loops when (q= 1) is the following:

$$DV = (1,3)\ if \begin{cases} q > 0\ and\ r = 3 \\ or \\ q < 0\ and\ r = -3 \end{cases}$$

or

$$DV = (1,-3)\ if \begin{cases} q > 0\ and\ r = -3 \\ or \\ q < 0\ and\ r = 3 \end{cases}$$

N4. (A, n)
FOR *i=1,n*
FORALL *j=1,n*
A[i , j]=A[i + ***1*** *, j +* ***3*** *]*
ENDFORALL
ENDFOR

From Fig.6, we notice that when we intend to exploit the cache memory sharing, even if we map the P1 and P4 threads respectively to the cores C1 and C2, and the threads P2 and P5 to C3 and C4, it is still hard to find a suitable placement that minimizes communication between cores sharing cache memory.

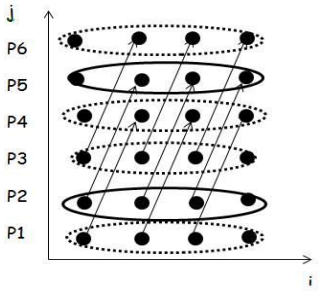


Fig. 6. *Iteration space of N4 (q = 1; r = 3)*

In fact, when repeating this process for P3, P6, P7 and P10 (see Fig.6), even if P4 and P7 are dependent, they will be allocated on cores non-sharing cache. Note that the dependency distance (r = 3) does not divide the cores number (n=4).

- **n = 4; q != 0; r = 4 || r = -4 :**

The DV of the obtained nested loops (N5) when (q= 1) is:

$$DV = (1,4)\ if \begin{cases} q > 0\ and\ r = 4 \\ or \\ q < 0\ and\ r = -4 \end{cases}$$

or

$$DV = (1,-4)\ if \begin{cases} q > 0\ and\ r = -4 \\ or \\ q < 0\ and\ r = 4 \end{cases}$$

In this case, since r = n (the cores number), the threads could be placed cyclically without respecting the hierarchy cache.

Core C1	Core C2	Core C3	Core C4
P1	P2	P3	P4
P5	P6	P7	P8
P9	P10	P11	P12
P13	P14	P15	P16
.	.	.	.
.	.	.	.
.	.	.	.

Fig. 7. *Mapping threads of N5 on cores (n=4)*

Following to the previous study of the different dependency cases targeting a quadcore machine, we notice that the only time in which the placement can be improved by adapting it to the cache memory sharing, is when the dependency distance vector is DV(q, r) where q! = 0 and r = 2 = n/2. Otherwise, in other cases, the obtained placement is: either that maximizes communications, or cannot be improved. Besides, we have studied the case of n = 6 and n = 8, and we also found that we can benefit from adapting the threads placement to a multicore architecture as mentioned in Fig.1 if and only if q != 0 and r = n/2. In the following, we are going

to show the proposed groups allocation and to summarize this study for any cores number (n).

- **n = 6; q != 0; r = 3 || r = -3 :**

The dependency distance vector of the bellow N6 nested loops when (q= 1) is the following:

$$DV = (1,3)\ if \begin{cases} q > 0\ and\ r = 3 \\ or \\ q < 0\ and\ r = -3 \end{cases}$$

or

$$DV = (1,-3)\ if \begin{cases} q > 0\ and\ r = -3 \\ or \\ q < 0\ and\ r = 3 \end{cases}$$

N6. (A, n)
FOR *i=1,n*
FORALL *j=1,n*
A[i , j]=A[i + ***1*** *, j +* ***3*** *]*
ENDFORALL
ENDFOR

This case is the only one where we could predict a mapping that minimizes communications and respects the cache sharing.

Core C1	Core C2	Core C3	Core C4	Core C5	Core C6
P1	P4	P2	P5	P3	P6
P10	P7	P11	P8	P12	P9
P13	P16	P17	P12	P15	P18
P22	P19	P23	P20	P24	P21
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Fig. 8. *Mapping threads of N6 on cores (n=6)*

Fig.8 shows that the threads could be divided into 6 groups:
$Group\ 1 = \{Pi/\ i = 12k + 1, k \geq 0\}\ U\ \{Pi/\ i = 12k + 10, k \geq 0\}$
$Group\ 2 = \{Pi/\ i = 12k + 4, k \geq 0\}\ U\ \{Pi/\ i = 12k + 7, k \geq 0\}$
$Group\ 3 = \{Pi/\ i = 12k + 2, k \geq 0\}\ U\ \{Pi/\ i = 12k + 11, k \geq 0\}$
$Group\ 4 = \{Pi/\ i = 12k + 5, k \geq 0\}\ U\ \{Pi/\ i = 12k + 8, k \geq 0\}$
$Group\ 5 = \{Pi/\ i = 12k + 3, k \geq 0\}\ U\ \{Pi/\ i = 12k, k > 0\}$
$Group\ 6 = \{Pi/\ i = 12k + 6, k \geq 0\}\ U\ \{Pi/\ i = 12k + 9, k \geq 0\}$

We mention that threads of Group1 (resp. Group2, Group3, Group4, Group5, Group6) are allocated respectively on the core C1 (resp. C2, C3, C4, C5, C6).

- **n = 8; q != 0; r = 4 || r = -4 :**

The dependency distance vector (DV) of the N8 nested loops when (q= 1) is the following:

$$DV = (1,4)\ if \begin{cases} q > 0\ and\ r = 4 \\ or \\ q < 0\ and\ r = -4 \end{cases}$$

or

$$DV = (1,-4)\ if \begin{cases} q > 0\ and\ r = -4 \\ or \\ q < 0\ and\ r = 4 \end{cases}$$

N7. (A, n)
FOR *i=1,n*
FORALL *j=1,n*
A[i , j]=A[i + ***1*** *, j +* ***4*** *]*
ENDFORALL
ENDFOR

Fig.9 shows the best placement of the OpenMP threads on an 8-cores architecture.

Core C1	Core C2	Core C3	Core C4	Core C5	Core C6	Core C7	Core C8
P1	P5	P2	P6	P3	P7	P4	P8
P13	P9	P14	P10	P15	P11	P16	P12
P11	P21	P18	P22	P19	P23	P20	P24
P29	P25	P30	P26	P31	P27	P32	P28
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Fig. 9. *Mapping threads of N7 on cores (n=8)*

In this case, the above threads could be divided into 8 groups as follows:
$Group\ 1 = \{Pi/\ i = 16k + 1, k \geq 0\}\ U\ \{Pi/\ i = 16k + 13, k \geq 0\}$
$Group\ 2 = \{Pi/\ i = 16k + 5, k \geq 0\}\ U\ \{Pi/\ i = 16k + 9, k \geq 0\}$
$Group\ 3 = \{Pi/\ i = 16k + 2, k \geq 0\}\ U\ \{Pi/\ i = 16k + 14, k \geq 0\}$
$Group\ 4 = \{Pi/\ i = 16k + 6, k \geq 0\}\ U\ \{Pi/\ i = 16k + 10, k \geq 0\}$
$Group\ 5 = \{Pi/\ i = 16k + 3, k \geq 0\}\ U\ \{Pi/\ i = 16k + 15, k \geq 0\}$
$Group\ 6 = \{Pi/\ i = 16k + 7, k \geq 0\}\ U\ \{Pi/\ i = 16k + 11, k \geq 0\}$
$Group\ 7 = \{Pi/\ i = 16k + 4, k \geq 0\}\ U\ \{Pi/\ i = 16k, k > 0\}$
$Group\ 8 = \{Pi/\ i = 16k + 8, k \geq 0\}\ U\ \{Pi/\ i = 16k + 12, k \geq 0\}$

Notice that threads of Group1 (resp. Group2, Group3, Group4, Group5, Group6, Group7, Group8) are allocated respectively on C1 (resp. C2, C3, C4, C5, C6, C7, C8).

- **Summary for any cores number n :**

The theoretical study can be generalized for any cores number (n) of a given target multicore machine (MA) respecting the hierarchy mentioned from Fig.1, where each pair of cores shares a memory cache.
We consider the case of a dependency distance vector DV(q,r) where q != 0 and r = n/2. Then, we define n groups as follows:

- $Group\ j = \{Pi;\ i = (2n)k + j\ \ for\ k \geq 0\}\ U\ \{Pi;\ i = (2n)k + 3r + j\ \ for\ k \geq 0\}$
  $where\ 1 \leq j \leq r - 1$
  The Groups j are mapped to the cores $C_{2(j-1)+1}$.

- $Group\ r + j = \{Pi;\ i = (2n)k + r + j\ \ for\ k \geq 0\}\ U\ \{Pi;\ i = (2n)k + 2r + j\ \ for\ k \geq 0\}$
  $where\ 1 \leq j \leq r - 1$
  The Groups r+j are mapped to the cores C_{2j}.

- $Group\ r = \{Pi;\ i = (2n)k + r\ \ for\ k \geq 0\}\ U\ \{Pi;\ i = (2n)k\ \ for\ k > 0\}$
  The Group r is mapped to the core C_{n-1}.

In the following section, we evaluate our theoretical contribution through a series of experiments.

3.2 Experimental study

First of all, let us mention that our multicore target machine (TM) is an Intel ® Xeon dual quadcore processor CPU E5420 @ 2.50GHz. Its eight cores have dedicated L1 cache with 32 KB size, and share in pairs the L2 cache level whose size is 6MB. The RAM size is 4 GB.

An experimental study covering the previous programs (N3, N6 and N7) was carried out on our TM in order to validate our theoretical proposal.

We compute the execution time of such codes in terms of the array size (N). Then we illustrate the speed-up (S), respectively for N3, N6 and N7. Our purpose is to ensure our general formula (see section 3.1) which maps dependent iterations on cores sharing the same cache in order to minimize the dependences and reduce the communication rate.

We mention that the following notations are adopted in this section to indicate the studied codes:
- N3_PC, N6_PC and N7_PC: are the parallel codes respectively of the nested loops N3, N6 and N7 undergoing the mapping depending on SCM.
- N3_PA, N6_PA and N7_PA: are the parallel codes obtained with the random placement (automatically generated by OpenMP) respectively of the nested loops N3, N6 and N7.

The following versions are derived from our target codes; the first is sequential and denoted N3(1th) (resp. N6(1th), resp. N7(1th)), the second version is N3_PC (resp. N6_PC, resp. N7_PC) and the third one is N3_PA (4 threads) (resp. N6_PC, resp. N7_PC).

They were coded in C under Linux. For the parallel experiments, we used the shared memory OpenMP environment. We point out that for each code; we chose 12 values of N in the range [500, 6000] with a step equal to 500. For each N, the execution time (T_{ex}) is the mean of five runs. We therefore achieved 180 tests in total (60 for each version). Excerpts of the results we obtained are depicted below.

TABLE I. EXECUTION TIME (S) AND SPEED-UP OF N3(1TH),N3_PC AND N3_PA ON TM

N	Execution time(s)			Speed-up	
	N3(1th)	**N3_PC**	N3_PA	**N3_PC**	N3_PA
500	0.048	**0.005**	0.021	**9.6**	2.28
1000	0.174	**0.010**	0.085	**17.4**	2.04
1500	0.254	**0.018**	0.150	**14.11**	1.69
2000	0.438	**0.031**	0.194	**14.12**	2.25
2500	0.659	**0.048**	0.223	**13.72**	2.95
3000	0.866	**0.068**	0.267	**12.73**	3.24
3500	1.078	**0.113**	0.316	**9.53**	3.41
4000	1.492	**0.152**	0.401	**9.81**	3.72
4500	1.990	**0.188**	0.465	**10.58**	4.27
5000	2.625	**0.231**	0.536	**11.36**	4.89
5500	3.268	**0.282**	0.658	**11.58**	4.96
6000	4.521	**0.335**	0.797	**13.49**	5.67

Table I and Fig.10 show that the obtained execution time (T_{ex}) for the N3_PC is less than the T_{ex} of N3_PA.

Moreover, they illustrate that the speed-up (S) does not vary uniformly with N, but S of the N3_PC is still better than that of N3_PA.

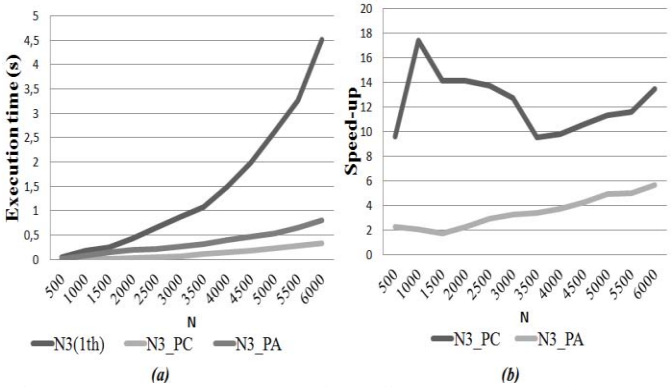


Fig. 10. *Experimental comparison of the different versions N3(1th), N3_PC and N3_PA on MA: (a) Execution time (s), (b) Speed-up*

We notice from Table II and Fig.11 that the N6_PC relying on the sharing cache memory (SCM) is the best. This is obvious because the mapping depending on the SCM takes advantage of data locality and minimizes cores communication. Fig.11 shows that the speed-up doesn't vary uniformly with N. This may be related to the OpenMP overheads.

TABLE II. EXECUTION TIME (S) AND SPEED-UP OF N6(1TH),N6_PC AND N6_PA ON TM

N	Execution time(s)			Speed-up	
	N6(1th)	**N6_PC**	N6_PA	**N6_PC**	N6_PA
500	0.296	**0.016**	0.046	**18.5**	6.43
1000	0.351	**0.025**	0.087	**14.04**	4.03
1500	0.792	**0.041**	0.122	**19.31**	6.49
2000	1.671	**0.056**	0.173	**29.83**	9.65
2500	2.618	**0.080**	0.244	**32.72**	10.72
3000	3.359	**0.113**	0.315	**29.72**	10.66
3500	4.793	**0.161**	0.382	**29.77**	12.54
4000	5.084	**0.208**	0.494	**24.44**	10.29
4500	6.433	**0.263**	0.589	**24.46**	10.92
5000	7.061	**0.329**	0.719	**21.46**	9.82
5500	8.334	**0.397**	0.806	**20.99**	10.33
6000	8.827	**0.475**	0.918	**18.58**	9.61

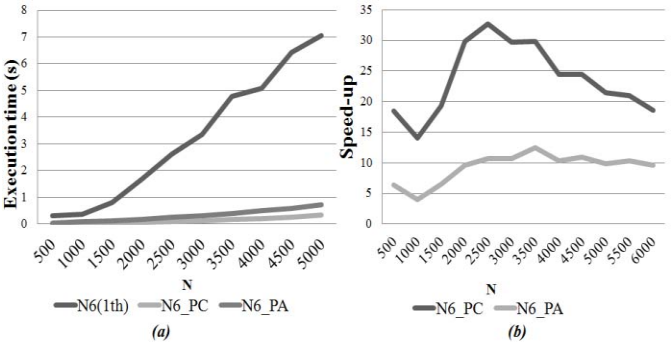


Fig. 11. *Experimental comparison of the different versions N6(1th),N6_PC and N6_PA on MA: (a) Execution time (s), (b) Speed-up*

Table III and Fig.12 display that N7_PC, which considers the SCM, is better than N7_PA.

TABLE III. EXECUTION TIME (S) AND SPEED-UP OF N7(1TH),N7_PC AND N7_PA ON TM

N	Execution time(s)			Speed-up	
	N7(1th)	**N7_PC**	N7_PA	**N7_PC**	N7_PA
500	0.252	**0.067**	0.115	**3.76**	2.19
1000	0.730	**0.070**	0.381	**10.42**	1.91
1500	1.359	**0.102**	0.455	**13.32**	2.98
2000	2.159	**0.117**	0.542	**18.45**	3.98
2500	3.279	**0.168**	0.619	**19.51**	5.29
3000	3.951	**0.181**	0.702	**21.82**	5.62
3500	5.123	**0.217**	0.853	**23.60**	6.00
4000	5.606	**0.297**	0.972	**18.87**	5.76
4500	6.471	**0.350**	1.089	**18.48**	5.94
5000	7.130	**0.432**	1.254	**16.50**	5.68
5500	8.518	**0.553**	1.387	**15.40**	6.14
6000	9.562	**0.694**	1.491	**13.77**	6.41

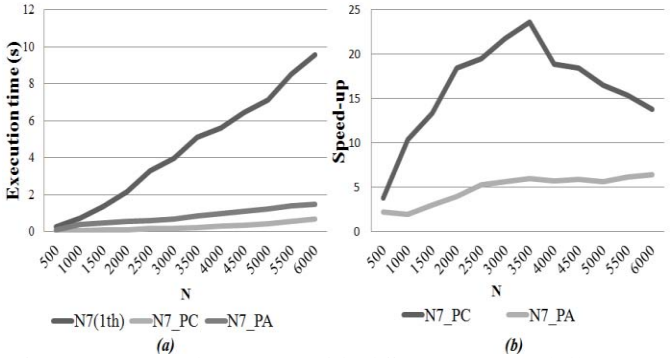


Fig. 12. *Experimental comparison of the different versions N7(1th),N7_PC and N7_PA on MA: (a) Execution time (s), (b) Speed-up*

4 Conclusion

The work reported in this paper aims to propose a solution for the problem of OpenMP threads mapping on multicore platform , taking into account the cache memory sharing since it has a very important impact on performance. In this context, we were interested in automatic parallelization of polyhedron programs. We first studied the most well-known methods focusing on adapting the mapping to the target architecture. Then, we presented our approach which consists in matching the program structure to the target hardware hierarchy relying on the residual dependencies. In fact, starting from a parallel program with a single level of dependency, we have proposed a grouping model of dependent threads to be mapped on the close cores sharing one or more than cache level. This took advantage of the data reuse and data locality and improves the performance of the target parallel code. Finally, experiments were carried in sequential and parallel environments on a quadcore bi-processor machine to validate our theoretical proposal.
However, several interesting points remain to be seen, in particularly: (i) extension of the theoretical study to multiple nested loops with more than one dependency, (ii) taking into account other architectural features of the target platform, (iii) extension of the experimental study to other programs and other parallel architectures.

5 References

[1] R. Thekkath and S.J. Eggers, "Impact of sharing-based thread placement on multithreaded architectures", In International Symposium on Computer Architecture (ISCA), Chicago, IL, pp. 176–186, April 1994.

[2] T. Klug, M. Ott, J. Weidendorfer, and T. Carsten, "autopin: automated optimization of thread-to-core pinning on multicore systems", Transactions on high-performance embedded architectures and compilers III, Springer-Verlag, Berlin, Heidelberg, pp. 219-235, January 2011.

[3] E.R. Rodrigues, F.L. Madruga, P.O.A. Navaux, and J. Panetta, "Multi-core aware process mapping and its impact on communication overhead of parallel applications", In IEEE Symposium on Computers and Communications (ISCC), Sousse, Tunisia, pp. 811–817, July 2009.

[4] M. Diener, F. Madruga, E. R. M. Alves, J. Schneider, P. Navaux, and H.U. Heiss, "Evaluating thread placement based on memory access patterns for multi-core processors", 12th IEEE International Conference on High Performance Computing and Communications (HPCC), Melbourne, VIC, pp. 491–496, September 2010.

[5] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuh, "MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters", In the Proceedings of the 20th annual international conference on Supercomputing (ICS), Cairns, Queensland, Australia, pp.353-360, June 2006.

[6] J. Clet-Ortega, "Une stratégie efficace pour le placement de processus en environnement multicoeur", 19ème Rencontres Francophones du Parallélisme (RenPar'19), SympA'13, CFSE'7, University Bordeaux 1,Toulouse, France, September 2009.

[7] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and N. Namyst, "hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications", Proceedings of the 18th Euromicro International Conference PDP 2010: Parallel, Distributed and Network-Based Processing, IEEE Computer Society Press, Pisa, Italia, pp 180-186, February 2010.

[8] C. Bastoul, "Improving Data Locality in Static Control Programs", PhD thesis, University Paris 6, Pierre et Marie Curie, France, December 2004.

[9] M. Kandemir, T. Yemliha, S. Muralidhara, S. Srikantaiah, M. Irwin, and Y. Zhnag, "Cache topology aware computation mapping for multicores", In Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), vol. 45(6), pp. 74–85, June 2010.

Parallel machine scheduling problems with machine and job correlations

Yang-Kuei Lin

Department of Industrial Engineering and Systems Management, Feng Chia University, P.O. Box 25-097, Taichung, 40724, Taiwan, ROC

Abstract - We study the problem of scheduling parallel machines with release time to minimize total weighted tardiness. We consider different levels of machine correlation and job correlation in the processing times. The problem is NP-hard in the strong sense since the single machine case is already NP-hard in the strong sense. A mathematical model is applied to evaluate the influence of machine correlation and job correlation on the computation results and computation time. Computational results show that as the machine and job correlations increase, the problem instances become more difficult for mathematical models to resolve.

Keywords: scheduling, parallel machines, correlation, release time, total weighted tardiness.

1 Introduction

This research consider the problem of scheduling *n* jobs on *m* parallel machines with release times to minimize total weighted tardiness. Each job *j* has a release date (r_j), a processing time (p_{ij}) on machine *i*, a due date (d_j) and a weight (w_j). Job preemptions are not allowed. We consider different levels of machine correlations and job correlations in the processing times. The total weighted tardiness ($\sum w_j T_j$) is a measure of customer satisfaction where the tardiness of job *j* is defined as $T_j = max(C_j - d_j, 0)$. This problem is NP-hard.

Traditionally, parallel machine scheduling problems have been classified as identical parallel machines (P_m), uniform parallel machines (Q_m), and unrelated parallel machines(R_m) (Pinedo, 2012). Although there is an extensive amount of literature sources on parallel machine scheduling problems, it is mostly limited to the above three traditional defined environments.

In addition to the three traditional definitions of parallel machine environments, research has been evolving to recognize dependencies between jobs and machines within a workgroup. For example, based on surveys of real-world manufacturing facilities, Panwalkar *et al.* (1973) have surveyed real-world manufacturing facilities and concluded that the correlation structures existing among jobs and machines in industry. Machine correlation means that processing times might be ordered by a machine due to the processing speed of the machines. Job correlation means that processing times might be ordered by a job if the jobs have large differences in size or complexity. Both machine and job correlated means that the processing time of a job depends on the complexity of the job, and the speed of the processing

ISBN: 1-60132-444-8, CSREA Press ©

machine.

Moreover, researchers have been aware that the amounts of job correlations and machine correlations may impact the performance of an algorithm or heuristic. In order to allow for possible variations in heuristic performance, some researchers tested their heuristics in various correlated environments (Hariri and Potts, 1991 and Vredeveld and Hurkens 2002). Although both Hariri and Potts (1991) and Vredeveld and Hurkens (2002) tested their heuristics in various correlated environments, they did not explore how these different types of correlated environments affect solution quality. There was no determination of whether or not the heuristics would be robust in relation to different types of correlated environments.

Recently, Lin et al. (2014) have provided a comprehensive classification of parallel machine environments. It defined nine cases that considered different levels and combinations of machine correlations and job correlations, as shown in Table 1. Lin et al. (2014) also proposed processing time generation schemes for the nine cases. Lin et al. (2014) introduced parameters Γ and Δ to control the relatedness of the generated processing times for the machine-correlated and job-correlated environments, respectively. Γ is inversely proportional to the relative dispersion of processing times between machines, and Δ is inversely proportional to the relative dispersion of processing times between jobs. The Γ and Δ values are passed into the generation scheme and the scheme generates a problem instance. Table 1 shows the nine cases and the corresponding environments based on traditional definitions of parallel machine environments. In Table 1, we can see the following nine different types of correlated environments:

Case 1 is the uncorrelated environment;

Case 2 is the machine-correlated environment;

Case 3 is the job-correlated environment;

Case 4 is an environment with several identical machines, all of which have full job uniformity;

Case 5 is an environment with full job uniformity and different levels of machine relatedness;

Case 6 is an environment with full machine correlation and different levels of job uniformity;

Case 7 is an environment with equal levels of machine relatedness and job uniformity;

Case 8 is an environment in which machine relatedness is larger than job uniformity;

Case 9 is an environment in which job uniformity is larger than machine relatedness.

Table1Nine cases of parallel machine environments (Lin et al. 2014)

	Γ					
Δ	0.0	0.2	0.4	0.6	0.8	1.0
0.0	1 R_m	2	2	2	2	2 Q_m^*
0.2	3	7	8	8	8	6 Q_m^*
0.4	3	9	7	8	8	6 Q_m^*
0.6	3	9	9	7	8	6 Q_m^*
0.8	3	9	9	9	7	6 Q_m^*
1.0	3 P_m	5 P_m	5 P_m	5 P_m	5 P_m	4 $p_{ij} = p$

Superscript shows case number

Center shows corresponding environment

Γ : machine relatedness factor

Δ : job uniformity factor

Q_m^*: This is a special case of Q_m

Lin et al. (2014) compared the performance of various heuristics and one metaheuristic for unrelated parallel machine scheduling problems. The objective functions to be minimized are makespan, total weighted completion time, and total weighted tardiness. They use the Least Significant Difference (LSD) test to identify robust heuristics that perform significantly better than others for the nine cases mentioned above with these three performance measures.

In this research, we try to solve the problem of scheduling parallel machines with release times to minimize total weighted tardiness. We consider different levels and all combinations of machine correlations and job correlations, as defined by Lin et al. (2014). We analyze the nine cases by using an existing mathematical model to determine if some cases are more difficult to solve than others.

2 Mathematical model

As mentioned before, some researchers have been aware that the amounts of job correlations and machine correlations may impact the performance of an algorithm or heuristic. Hence, some of them tested their heuristics or algorithms on different variations of job correlation and machine correlation environments. Here, we test the nine cases defined by Lin et al. (2014) by using a mathematical model proposed by Lin and Lin (2013). The mathematical model can be used to solve scheduling unrelated parallel machines with release times to minimize total weighted tardiness. The mathematical model is described below. A binary variable, x_{ijt}, which is equal to 1 if job *j* is scheduled on machine *i* starts at time *t* and is equal to zero otherwise. Assuming $T \geq \max_{1 \leq j \leq n} r_j + \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$.

objective: $$\min \sum_{j=1}^{n} w_j T_j \quad (1)$$

subject to:

$$\sum_{i=1}^{m} \sum_{t=r_j}^{T-p_{ij}+1} x_{ijt} = 1 \qquad \forall j \quad (2)$$

$$\sum_{j=1}^{n} \sum_{s=\max(r_j, t-p_{ij}+1)}^{t} x_{ijs} \leq 1 \qquad \forall i,t \quad (3)$$

$$\sum_{i=1}^{m} \sum_{t=0}^{r_j} x_{ijt} = 0 \qquad \forall j \quad (4)$$

$$C_j = \sum_{i=1}^{m} \sum_{t=0}^{T-p_{ij}+1} x_{ijt}(t-1+p_{ij}) \qquad \forall j \quad (5)$$

$$C_j - d_j \leq T_j \qquad \forall j \quad (6)$$

$$T_j \geq 0 \qquad \forall j \quad (7)$$

$$x_{ijt} \in \{0,1\} \qquad \forall i,j,t \quad (8)$$

Equation (1) as an objective function. Constraints (2) enforce that each job can start only at exactly one particular time on exactly one machine. Constraints (3) ensure that at any given time on each machine at most one job can be processed. Constraints (4) enforce that each job cannot be processed before it is released. Using the time-indexed variables, the

completion time of a job *j* can be written as constraints (5). Constraints (6) calculate total tardiness. Constraints (7) are non-negative constraints. Constraints (8) state the integrality restriction.

3 Computational Results

In this section, we present computational results of the mathematical model for different levels of machine correlations and job correlations. The mathematical model is coded in AMPL and implemented in CPLEX 11.2. Also, the mathematical model is executed on a computer with a 3.4 GHz CPU and 8GB of memory. Processing times p_{ij} are generated based on Lin et al. (2014). The value of w_j for each j is generated from the uniform distribution [1,10]. Release dates and due dates are generated in a manner similar to that of Mönch *et al.* (2005). We first generate release dates r_j from the uniform distribution $[0, \frac{\alpha}{m}\frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{m}]$. In the next step, we generate slack times between due dates and earliest completion times from a uniform distribution $[0, \frac{\beta}{m}\frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{m}]$, *i.e.*, $d_j - (r_j + \bar{p}_j)$ where $\bar{p}_j = \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}/m$, α controls the range of release dates, and β controls the range of due dates. Higher values of α tend to produce widely separated release dates, while higher values of β tend to produce loose due dates. As in Mönch *et al.* (2005), α and β were set at 0.25, 0.5, and 0.75. Since solving a mathematical model can be very time consuming, we only test it on 4 machines with 20 jobs ($4m20n$). For each combination of α, β and problem instance size, 5 problem instances were randomly generated and tested.

We evaluate the average computation time and proportion of problem solved by mathematical model on the nine cases defined by Lin et al. (2014) to determine whether if some cases are more difficult to resolve than others. Table 2 showed the results of the nine cases. The second column of Table 2 shows the percentage of problem instances be solved within a time limit of two hours by using a mathematical model. It shows that Case 1 (unrelated parallel machines environment) is the easiest one for the mathematical model to solve. 100% of the problem instances has been solved quickly. When machine or job correlations got involved in the generated processing times, the problems became difficult for the mathematical model to solve. For example, Case 5 is the same as the traditional definition of identical parallel machines (Pm). Only 40% of problem instances has been solved within the time limit for Case 5. Most problem instances failed to be solved due to memory ran out, or the solving time exceeded the time limit. This is probably because as the machine correlation and/or job correlation increase(s), the job processing times become very similar to each other on all machines. The

branch-and-bound algorithm imbedded in CPLEX cannot fathom non-optimal solutions efficiently during branching. It ends with running out of memory. Moreover, for Case 5, in average it takes 3862.96 seconds to solve an instance.

Table2 Computation results for nine cases

	% of problem solved by Mathematical model	Computation time(s)	Parameters setting Num. of instances tested
Case1	100%	9.12	Γ=0.0; Δ=0.0 5 instances
Case2	60%	1633.34	Γ=0.2, 0.4, 0.6, 0.8, 1.0; Δ=0.0 25 instances
Case3	56%	2452.27	Γ=0.0; Δ=0.2, 0.4, 0.6, 0.8, 1.0 25 instances
Case4	40%	1128.08	Γ=1.0; Δ=1.0 5 instances
Case5	45%	3862.96	Γ=0.2, 0.4, 0.6, 0.8; Δ=1.0 20 instances
Case6	55%	1075.28	Γ=1.0; Δ=0.2, 0.4, 0.6, 0.8 20 instances
Case7	75%	2453.12	Γ=0.2, 0.4, 0.6, 0.8; Δ=0.2, 0.4, 0.6, 0.8 20 instances
Case8	43%	3185.51	Γ=0.4, 0.6, 0.8; Δ=0.2, 0.4, 0.6 30 instances
Case9	53%	2909.83	Γ=0.2, 0.4, 0.6; Δ=0.4, 0.6, 0.8 30 instances
Average	58.56%	2078.84	

Table 3 shows the results of all combination settings of machine correlation and job correlation. It is a 6×6 matrix. We use a;b;c to indicate the proportion of solving status for the 5 instances. 'a' represents the proportion of problem has been solved, and an optimal solution has been found within 2 hours of time limit. 'b' represents the proportion of problem cannot be solved due to memory ran out. 'c' represents the proportion of problem cannot be solved within the time limit. For example, when Γ=1.0 and Δ=0.2, it corresponding to Case 5. 2/5 of problem instances has been solved within 2 hours time limit; 2/5 of problem instances failed to be solved due to memory ran out, and 1/5 of problem instances failed to be solved within the time limit. Moreover, it takes 4757.1 seconds to solve an instance on average. Again, when machine or job correlations got involved in the generated processing times, the problems became difficult for the mathematical model to solve.

	Γ					
Δ	0.0	0.2	0.4	0.6	0.8	1.0
0.0	1 5/5;0;0 9.12s	2 4/5;1/5;0 1136.36s	2 1/5;4/5;0 2519.38s	2 3/5;2/5;0 1848.3s	2 4/5;1/5;0 1938.54s	2 3/5;2/5;0 724.1s
0.2	3 2/5;3/5;0 2640.54s	7 4/5;1/5;0 2186.4s	8 2/5;3/5;0 3071.42s	8 2/5;3/5;0 2281.36s	8 2/5;3/5;0 3895.48s	6 3/5;2/5;0 945.18s
0.4	3 4/5;1/5;0 1504.58s	9 0;5/5;0 4722.18s	7 1/5;4/5;0 4197.42s	8 3/5;1/5;1/5 4454.58s	8 2/5;2/5;1/5 3170.86s	6 2/5;3/5;0 1338.62s
0.6	3 5/5;0;0 921.32s	9 2/5;3/5;0 4070.48s	9 3/5;1/5;1/5 2918.36s	7 5/5;0;0 1847.96s	8 2/5;3/5;0 2239.36s	6 4/5;1/5;0 479.54s
0.8	3 3/5;2/5;0 1911.3s	9 4/5;1/5;0 1477.74s	9 3/5;2/5;0 2229.78s	9 4/5;1/5;0 2040.46s	7 5/5;0;0 1580.7s	6 2/5;3/5;0 1537.78s
1.0	3 0;5/5;0 5283.6s	5 2/5;2/5;1/5 4575.1s	5 2/5;3/5;0 3834.28s	5 2/5;3/5;0 4863.18s	5 3/5;1/5;1/5 2179.26s	4 2/5;3/5;0 1128.08s

4 Conclusions

In this research, we study the problem of scheduling parallel machines with release time to minimize total weighted tardiness. Several different levels and combinations of machine correlation and job correlation in the processing times are considered. A mathematical model has been applied to examine if some problems are more difficult to olve than the others when machine correlation or job correlation get involved in generating processing times. Computational results show that both machine correlation and job correlation influence the performance and computation time of the mathematical model.

References

Hariri AMA, Potts CN. Heuristics for scheduling unrelated parallel machines. Computers and Operations Research 1991;(18):323-31.

Vredeveld T, Hurkens C. Experimental comparison of approximation algorithms for scheduling unrelated parallel machines. INFORMS Journal on Computing Linthicum 2002;(14):175-90.

Pinedo, M. 2012. Scheduling Theory, Algorithms, and Systems (4th ed.). Prentice Hall.

Panwalkar, S. S., Dudek, R. A., and Smith, M. L. 1973. Sequencing research and the industrial scheduling problem. Elmaghraby, S.E. editor. *Proceedings of Symposium on Theory of Scheduling and its Applications*, Springer-Verlag, New York, 29-38.

Hall, N.G., Posner, M.E. 2001. Generating experimental data for computational testing with machine scheduling applications. *Operations Research* 49 854-865.

Lin, Yang-Kuei, Lin, Chi-Wei, 2013. Dispatching rules for unrelated parallel machine scheduling with release dates. International Journal of Advanced Manufacturing Technology,

67, 269-279.

Lin, Y. K., Pfund, M. E., & Fowler, J. W. 2014. Processing time generation schemes for parallel machine scheduling problems with various correlation structures. Journal of Scheduling, 17(6), 569-586.

Mönch L, Balasubramanian H, Fowler JW, Pfund ME (2005) Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. Computers and Operations Research 32:2731-2750

The 2016 World Congress in Computer Science, Computer Engineering and Applied Computing

Keynote Speech: Developing Synergistic Intelligent Computing and Big Data Analytics Approaches to Facilitate Precision Medicine Research

Prof. Mary Yang

Director of MidSouth Bioinformatics Center and Director of Joint Bioinformatics Ph.D. Program, University of Arkansas Little Rock George Washington Donaghey College of Engineering & Information Technology and University of Arkansas for Medical Sciences, 2801 S. University Avenue, Little Rock, Arkansas 72204 U.S.A. Email: mqyang [at] ualr.edu

Advances of new technologies have generated massive big data that can facilitate the emerging precision medicine research. This has created tremendous demands for the development of novel intelligent computing approaches to handle the massive amount of genomic big data effectively and timely. In particular, cancer is a disease that is not only complex, in that many genetic variations contribute to malignant transformation, but also wildly heterogeneous, in that genetic mechanisms can vary significantly between patients. Early diagnosis and effective treatment of cancer have been always remained challenging. Synergistic integration of multidimensional genomic big data at systems level can shed new light on molecular mechanisms at cellular level such as disease initiation and progression, and also lead to new pathway-based biomarker and drug target identifications. The Systems Genomics Laboratory along with the MidSouth Bioinformatics Center and Joint Bioinformatics Program of University of Arkansas Little Rock College of Engineering & Information Technology and University of Arkansas for Medical Sciences aims to leverage the research by combining different genomic information including genetic mapping, long non-coding RNA (lncRNA) studies, differential expression of genes (DEG), protein-protein and protein–nucleotide interactions to construct gene networks for integrative genome-phoneme studies at higher systems level. To this end, synergistic integration of multi-layer genomic big data can further advance biomedical research and contribute to precision medicine research.

In this keynote lecture, I will present synergistic intelligent computing, statistical and biochemical approaches to effectively integrate gene expression profiles with protein interactions in constructing gene networks that can be used for identification of biomarkers and disease associated pathways. By further combing with genotype information, we have discovered important genomic alterations in cancer development. Furthermore, our synergistic approaches also incorporate the study of lncRNAs and have identified differentially expressed lncRNAs in cancer. We found that many over-regulated lncRNAs were bidirectionally oriented with neighboring protein-coding genes. These protein-coding genes are enriched in biological processes implicated in cancer. In addition, we have developed an online tool called **IDEAS** to **I**dentify **D**ifferential **E**xpression of genes for **A**pplications in genome-wide

ISBN: 1-60132-444-8, CSREA Press ©

Studies. We have utilized this tool to facilitate synergistic knowledge discovery from multi-layer genomic big data and made a number of discoveries. Our integrative systems genomics approaches enable comprehensive identification of biomarkers, drug targets and disturbed pathways to facilitate precision medicine research.

Biography of the Keynote Speaker

Dr. Mary Yang is the Director of MidSouth Bioinformatics Center and Director of the Joint Bioinformatics Ph.D. Program of University of Arkansas Little Rock College of Engineering & Information Technology and University of Arkansas for Medical Sciences. After finishing her M.S.E.C.E. (Computer Engineering), M.S. (Biological Physics), and Ph.D. degrees from Purdue University with interdisciplinary Bilsland Dissertation Fellowship award, she joined the National Human Genome Research Institute at the National Institutes of Health (NIH) in Washington DC area in 2005. During her tenure at NIH, she made significant contributions to various large-scale genomics and systems biology research projects. Dr. Yang has been Founding Editor-in-Chief of *International Journal of Computational Biology and Drug Design*, a NIH PubMed fully indexed journal and is on editorial broads of *Journal of Supercomputing* and *International Journal of Pattern Recognition and Artificial Intelligence*. She was recruited to the University of Arkansas in 2013 to guide the MidSouth Bioinformatics Center and joint bioinformatics program. She has served on Steering Committee for NIH funded Arkansas INBRE and Review Committee for United States National Science Foundation (NSF) Advances in Biological Informatics (ABI). She has been the recipient of NIH Fellows Award for Research Excellence, NIH Academic Research Enhancement Award, Purdue Research Foundation Fellowship, IEEE and ISIBM Bioinformatics and Bioengineering Outstanding Achievement Awards, and Basic Science Research Award of Arkansas Science and Technology Authority (ASTA). Dr. Yang is a tenured faculty with the University of Arkansas and her Systems Genomics Laboratory is currently supported by NIH, FDA and ASTA. She has published over one hundred research articles in computer science and biomedical sciences.

ISBN: 1-60132-444-8, CSREA Press ©

ISBN: 1-60132-444-8, CSREA Press ©