Performance Evaluation of Some Inverse Iteration Algorithms on PowerXCell™ 8i Processor

Masami Takata¹, Hiroyuki Ishigami², Kinji Kimura², and Yoshimasa Nakamura²

¹Academic Group of Information and Computer Sciences, Nara Women’s University, Nara, Nara, JAPAN
²Graduate School of Informatics, Kyoto University, Kyoto, Kyoto, JAPAN

Abstract—In this paper, we compare with the inverse iteration algorithms on PowerXCell™ 8i processor, which has been known as a heterogeneous environment. When some of all the eigenvalues are close together or there are clusters of eigenvalues, reorthogonalization must be adopted to all the eigenvectors associated with such eigenvalues. Reorthogonalization algorithms need a lot of computational cost. The Classical Gram-Schmidt (CGS) algorithm, the modified Gram-Schmidt (MGS) algorithm, and the Householder orthogonalization algorithm in terms of the compact WY representation have been known as reorthogonalization algorithms. These algorithms can be computed using BLAS level-1 and level-2. Since synergistic processor elements in PowerXCell™ 8i processor archive the high performance of BLAS level-2 and level-3, the orthogonalization algorithms except the MGS algorithm can be computed high-speed on parallel computers.

Keywords: inverse iteration, eigenvalue decomposition, Classical Gram-Schmidt, Householder transformation, modified Gram-Schmidt, PowerXCell™ 8i processor

1. Introduction

The eigenvalue decomposition of a symmetric matrix is one of the most important operations in linear algebra. It is used in molecular orbital of chemical, vibrational analysis, image processing, data searches, etc.

Owing to recent improvements in the performance of computers equipped with multicore processors, we have had more opportunities to perform calculations on parallel computers. As a result, there has been an increase in the demand for an eigenvalue decomposition algorithm that can be effectively parallelized.

Any \( n \times n \) symmetric matrix is transformed into a symmetric tridiagonal matrix by using a sequence of Householder transformations \([4],[9]\). This preconditioning process helps to shorten computational time drastically. Hence, eigenvalue decomposition algorithms of symmetric tridiagonal matrices are important. Several eigenvalue decomposition algorithms of a symmetric tridiagonal matrix have been proposed \([3],[7],[10],[12],[13],[17]\). They are classified into two types. The first type of algorithm computes simultaneously all the eigenvalues and the eigenvectors. Algorithms of this type include the QR algorithm \([10]\) and the divide-and-conquer algorithm \([3],[13]\). The second type of algorithm computes all or some eigenvalues and all or some eigenvectors. Algorithms for computing eigenvalues include the root-free QR algorithm \([12]\) and the bisection algorithm \([10]\). Algorithms for computing eigenvectors include the MR³ algorithm \([7]\) and the inverse iteration algorithm with the modified Gram-Schmidt (MGS) algorithm \([10],[17]\). LAPACK (Linear Algebra PACKage) \([16]\), which is a software library for numerical linear algebra, has codes that integrate all the above-mentioned algorithms. These algorithms can be parallelized, except the root-free QR algorithm.

The inverse iteration algorithm is an algorithm for computing eigenvectors independently associated with mutually distinct eigenvalues. However, when some eigenvalues are very close to each other, the eigenvectors, which are computed using the inverse iteration algorithm, must be reorthogonalized. As reorthogonalization algorithms, the Classical Gram-Schmidt (CGS) algorithm \([10]\), the MGS algorithm, the Householder orthogonalization algorithm \([15]\) are known. Reorthogonalization algorithms need a lot of computational cost. The CGS algorithm is suitable algorithm for parallel computing. The orthogonality of eigenvectors computed by the CGS algorithm depends on the square of the condition number of the eigenvectors, which are generated using the inverse iteration, in the same cluster of the eigenvalues \([20]\). The MGS algorithm is sequential and inefficient for parallel computing. The orthogonality of eigenvectors computed by the MGS algorithm depends on the condition number. The Householder orthogonalization algorithm can orthogonalize eigenvectors by using the Householder transformation \([19]\). The orthogonality in the Householder orthogonalization algorithm does not depend on the condition number. The Householder algorithm is sequential and inefficient for parallel computing. Ishigami et. al. have developed parallel algorithms for the Householder orthogonalization algorithm in terms of the compact WY representation \([15]\), which is named as the cWY algorithm in this paper.

In ExaFLOP computing, since it is critical issue to minimize electricity, heterogeneous environments are suitable. Consequently, it is important to validate the inverse iteration algorithms with the CGS algorithm, the MGS algorithm, and the cWY in heterogeneous environments. As a heteroge-
neous environment, cell processor has PowerPC Processor Element (PPE) and eight cores of Synergistic Processor Elements (SPEs). PPE and SPEs can share the same memory. Since SPEs are consisted as multicore, SPEs archive the high performance of BLAS level-2 and level-3 [1]. Basic Linear Algebra Subprograms (BLAS) is an application programming interface standard for publishing libraries to perform basic linear algebra operations such as vector and matrix multiplications. BLAS level-1 can compute vector operations such as inner products, dot products and vector norms. BLAS level-2 and level-3 contain matrix-vector and matrix-matrix operations, respectively. The CGS algorithm and the MGS algorithm can be computed using BLAS level-2 and level-1, respectively. The cWY needs BLAS level-1 and level-2. Therefore, these orthogonalization algorithms should be performed in SPEs. By using PPE, an implementation of an inverse iteration is easy. In this paper, we compare with the CGS algorithm, the MGS algorithm, and the cWY on PowerXCellTM 8i processor.

In Section 2, we give a brief review on eigenvalue decomposition. In Section 3, we explain an inverse iteration algorithm and describe its orthogonalization algorithms. In Section 4, we confirm each performance in the inverse iteration algorithm is the inverse iteration algorithm. When some of all the eigenvalues are close together or there are clusters of eigenvalues, reorthogonalization must be needed to all the eigenvectors associated with such eigenvalues, since the eigenvectors needs to be orthogonal to each other. Therefore, reorthogonalization algorithms should be adopted.

In Section 3.1, we explain a concept of the inverse iteration algorithm. In Section 3.2, 3.3, and 3.4, the CGS algorithm, the MGS algorithm and the cWY are described, respectively. In Section 3.5, these orthogonalization algorithm are compared. In Section 3.6, we describe a relationship between BLAS and the orthogonalization algorithms.

3. Inverse iteration algorithm

In this section, we introduce the inverse iteration algorithm. When some of all the eigenvalues are close together or there are clusters of eigenvalues, reorthogonalization must be needed to all the eigenvectors associated with such eigenvalues, since the eigenvectors needs to be orthogonal to each other. Therefore, reorthogonalization algorithms should be adopted.

In Section 3.1, we explain a concept of the inverse iteration algorithm. In Section 3.2, 3.3, and 3.4, the CGS algorithm, the MGS algorithm and the cWY are described, respectively. In Section 3.5, these orthogonalization algorithm are compared. In Section 3.6, we describe a relationship between BLAS and the orthogonalization algorithms.

3.1 Concept

When \( \tilde{\lambda}_j \) is an approximate value of \( \lambda_j \) and a starting vector \( v_j^{(0)} \) are given, the inverse iteration algorithm can compute an eigenvector of \( T \). To this end, the following equation is solved iteratively:

\[
(T - \tilde{\lambda}_j I)v_j^{(k)} = v_j^{(k-1)}
\]

If the eigenvalues of \( T \) are mutually well-separated, the solution of \( v_j^{(k)} \) in Eq.(10) generically converges to the eigenvector associated with \( \lambda_j \) as \( k \) goes to \( \infty \). The above iteration algorithm is the inverse iteration algorithm. When \( m \) eigenvectors are computed, the computational cost of this algorithm is of order \( mn \). The computational cost is less than that of other algorithms. In the implementation, the vector \( v_j^{(k)} \) must be normalized to avoid overflow.

3.2 Classical Gram-Schmidt algorithm

The CGS algorithm has been proposed as the first re-orthogonalization algorithm. In the CGS algorithm, a basis
1: $x_1 = v_1$
2: for $j = 2$ to $m$
3: Generate $v_j$ in an algorithm.
4: Eq.(11) and Eq.(12): Orthogonalize $v_j$ to $x_j$ by using $x_1, \ldots, x_{j-1}$.
5: end for

Fig. 1: Classical Gram-Schmidt algorithm.

1: for $j = 1$ to $n$
2: Generate $v_j^{(0)}$ from random numbers.
3: $k = 0$
4: repeat
5: $k \leftarrow k + 1$.
6: Normalize $v_j^{(k-1)}$.
7: Eq.(10): Compute $v_j^{(k)}$ by using $v_j^{(k-1)}$.
8: if $|\lambda_j - \bar{\lambda}_{j-1}| \leq 10^{-3}||T||$, then
9: for $i = j_1$ to $j - 1$ do
10: $v_j^{(k)} \leftarrow v_j^{(k)} - [x_1, x_2, \ldots, x_{j-1}] v_j^{(k)}$
11: end for
12: else
13: if $j_1 = j$
14: end if
15: until some condition is met.
16: Normalize $v_j^{(k)}$ to $x_j$.
17: end for

Fig. 2: Inverse iteration algorithm with the CGS algorithm. $j_1$ means the index $j$ of the first eigenvalue of a cluster.

vector $x_j$, which is an orthogonal vector in $v_j$, is computed as follows:

$$x'_j = v_j - \sum_{i=1}^{j-1} \langle v_j, x_i \rangle x_i, \quad (11)$$

$$x_j = \frac{x'_j}{||x'_j||} \quad (12)$$

In Eq.(11), $\langle v_j, x_i \rangle x_i$ means an orthographic projection on the direction to $x_i$ of $v_j$. Through $v_j$ is subtracted the orthographic projection, $v_j$ can be picked out of elements $x_1, x_2, \ldots, x_{j-1}$. Thus, $x_j$ is orthogonalized.

Figure 1 shows the orthogonalization algorithm using the CGS algorithm. Since Eq.(11) and Eq.(12) are computed using an inner product, BLAS level-1 has to be adopted. Therefore, to adopt BLAS level-2, Eq.(11) and Eq.(12) should be transformed into the following vector product.

$$x'_j = v_j - [x_1, x_2, \ldots, x_{j-1}] \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_{j-1}^T \end{bmatrix} v_j, \quad (13)$$

Figure 2 is a code, which is based on DSTEIN in LAPACK and modified the orthogonalization process from the MGS algorithm to the CGS algorithm. Specifically, line 10 in Figure 2 is changed to Eq.(13).

3.3 Modified Gram-Schmidt algorithm

If the MGS algorithm is adopted to reorthogonalize eigenvectors, the computational cost is of order $m^2 n$. Therefore, the computational cost, for which eigenvectors of a matrix $T$ are computed, increases significantly. In general, to implement the inverse iteration algorithm on computers, the MGS algorithm with the Peters-Wilkinson method [17] is adopted as the standard orthogonalization process. The MGS algorithm with the Peters-Wilkinson method is also available on DSTEIN, which is implemented in the LAPACK code [16] of the inverse iteration algorithm for computing eigenvectors of a real symmetric tridiagonal matrix. In the Peters-Wilkinson method, when the distance between the close eigenvalues is less than $10^{-3}||T||$, these close eigenvalues are regarded as members of the same cluster of eigenvalues, and all of the eigenvectors associated with these eigenvalues are orthogonalized.

Figure 3 shows the inverse iteration algorithm based on the MGS algorithm with the Peters-Wilkinson method. This loop includes the iteration based on Eq.(10) and the orthogonalization of the eigenvectors. This orthogonalization process becomes a bottleneck of the inverse iteration with respect to the computational time. The MGS algorithm is mainly based on BLAS level-1 such as the inner product operation and the AXPY operation [1].

3.4 Householder orthogonalization algorithm

The Householder orthogonalization algorithm is one of the alternative orthogonalization algorithms. When some vectors $v_j, w_j \in \mathbb{R}^n$ satisfy $||v_j||_2 = ||w_j||_2$, there exists the symmetric matrix $H_j$ satisfying $H_j H_j^T = H_j^T H_j = I$. $H_j v_j = w_j$ defined by

$$H_j = I - s_j y_j y_j^T, \quad (14)$$

where $y_j = v_j - w_j$ and $s_j = 2/||y_j||_2^2$. The transformation by $H_j$ is called the Householder transformation. Figure 4 shows the Householder orthogonalization algorithm. The
for $j = 1$ to $m$ do
2: Generate $v_j$ in an algorithm.
3: $v_j' = (I - s_{j-1}y_j'y_j') \cdots (I - s_2y_2y_2') (I - s_1y_1y_1') v_j$.
4: Compute $y_j$ and $s_j$ by using $v_j'$.
5: $x_j = (I - s_1y_1y_1') \cdots (I - s_2y_2y_2') e_j$.
6: end for

Fig. 4: Householder orthogonalization algorithm.

vector $y_j$ is the vector, in which the elements from 1 to
$j - 1$ are the same as the elements of $v_j'$ and the elements
from $j + 1$ to $n$ are zero. $v_j'$ and $w_j$ are defined as follows:

$$v_j' = \begin{bmatrix} v_j'(1) & \cdots & v_j'(j-1) & v_j'(j) & \cdots & v_j'(n) \end{bmatrix}^T$$

$$= H_{j-1}H_{j-2} \cdots H_1v_j,$$ (15)

$$w_j = \begin{bmatrix} v_j'(1) & \cdots & v_j'(j-1) & c_j & 0 \end{bmatrix}^T,$$ (16)

where,

$$c_j = -\text{sgn}(v_j'(j)) \sqrt{\sum_{i=j}^{n} v_j'(i)'^2}.$$ (17)

$H_j$, $y_j$ and $s_j$ are computed using $v_j$ as follows:

$$H_j = I - s_jy_jy_j^T$$ (18)

$$y_j = v_j' - w_j$$ (19)

$$\|y_j\|^2_2 = (v_j'(j) - c_j)^2 + \sum_{i=j+1}^{n} v_j'(i)^2$$ (20)

$$= \sum_{i=j}^{n} v_j'(i)^2 - 2v_j'(j)c_j + c_j^2$$ (21)

$$= 2\left(c_j^2 - v_j'(j)c_j\right).$$ (22)

$$s_j = \frac{1}{\|y_j\|^2_2} = \frac{1}{c_j^2 - v_j'(j)c_j}.$$ (23)

The vector $e_j$ in Figure 4 is the $j$-th vector of an $n$-
dimensional identity matrix.

The orthogonality of the vectors $x_j$ generated by
the Householder orthogonalization algorithm does not depend
on the condition number of a matrix $T$. Therefore, the
Householder orthogonalization algorithm is more stable than
the MGS algorithm. On the other hand, being similar to the
MGS algorithm, it is sequential algorithm that is mainly
based on BLAS level-1. Its computational cost is higher
than that of the MGS algorithm. Thus the Householder
orthogonalization algorithm is an ineffective algorithm in parallel computing.

By combination with the compact WY representation
[18], the Householder orthogonalization algorithm becomes
capable of computation with BLAS level-2 [20]. Hence, in
this paper, the cWY is adopted to an inverse iteration. Let
$Y_1 = y_1 \in \mathbb{R}^{n \times 1}$ and $S_1 = s_1 \in \mathbb{R}^{1 \times 1}$. Matrices $Y_j$

$$q_j = (I - Y_jS_jY_j^T) e_j.$$ (25)

In this case, the following equation holds

$$H_1H_2 \cdots H_j = I - Y_jS_jY_j^T.$$ (26)

As shown by Eq.(26), the product of the Householder matrices
$H_1H_2 \cdots H_j$ can be rewritten in a simple block matrix form. Here $I - Y_jS_jY_j^T$ is called the compact WY representation of the product of the Householder matrices. Figure 5 shows the orthogonalization algorithm.

Figure 6 is a code, which is based on DSTEIN in
LAPACK and changed the orthogonalization process from the MGS algorithm to the cWY algorithm. In other words, the MGS algorithm (from line 4 to 15 in Figure 3) is rewritten as the cWY algorithm. In the code, the index $j_c$ denotes the $j_c$-th eigenvalue of the cluster in computing the $j_c$-th eigenvector. This index $j_c$ needs to compute and update $S_j$ and $Y_j$.
Table 1: Comparison of the orthogonalization algorithms [5] [20].

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Computation</th>
<th>Synchronization</th>
<th>Orthogonality</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGS</td>
<td>almost $2m^2n$</td>
<td>$O(m)$</td>
<td>$O(e(A)^2)$</td>
</tr>
<tr>
<td>MGS</td>
<td>almost $2m^2n$</td>
<td>$O(m^2)$</td>
<td>$O(e(A))$</td>
</tr>
<tr>
<td>House</td>
<td>almost $4m^2n$</td>
<td>$O(m^2)$</td>
<td>$O(e)$</td>
</tr>
<tr>
<td>cWY</td>
<td>almost $4m^2n$</td>
<td>$O(m)$</td>
<td>$O(e)$</td>
</tr>
</tbody>
</table>

Therefore, a variable $j_c$ should be confirmed on line 9 in Figure 6.

The cWY algorithm has a stable orthogonality arising from the Householder transformations, and its mathematical calculation is mainly performed by BLAS level-2 such as the product of a matrix and a vector and a rank-1 update operation.

3.5 Comparison of the orthogonalization algorithms

The cWY algorithm has a stable orthogonality arising from the Householder transformations, and its mathematical calculation is mainly performed by BLAS level-2 such as the product of a matrix and a vector and a rank-1 update operation. As a result, this orthogonalization has more stable and sophisticated orthogonality, and it is more effective for parallel computing than the MGS algorithm. Table 1 displays the differences in performance of the four orthogonalization methods, considered in the above sections. In this table, “House” denotes the Householder orthogonalization algorithm. Computation denotes the order of the computational cost. Synchronization denotes the order of the number of synchronizations. Orthogonality denotes the norm $\| V^T V - I \|$, where $V = [v_1, \ldots, v_m]$. $\epsilon$ denotes the machine epsilon and $\kappa$ denotes the condition number of a matrix. These are the results obtained from [5] and [20].

On the other hand, the computational cost in the CGS algorithm is twice less than that in the cWY algorithm. Therefore, when high orthogonality is not needed, the CGS algorithm is also the suitable selection for the orthogonalization.

3.6 Adoption of BLAS

The line from 1 to 7 on each algorithm is the code in the inverse iteration algorithm without an orthogonalization algorithm. This computational cost $mn$ is relatively smaller than that in the inverse iteration algorithm with an orthogonalization algorithm shown in Table 1. Therefore, we adopt SPEs to orthogonalization algorithms.

In the CGS algorithm, the line 10 on Figure 2 can be computed using BLAS level-2. In the MGS algorithm, BLAS level-1 is adopted in the line 10 on Figure 3. In the cWY algorithm, the line 13 and 16 on Figure 6 can be performed with BLAS level-2, and the line 11 and 14 can be performed with BLAS level-1.

4. Experiments

In this section, we describe some numerical experiments performed using the CGS algorithm, the MGS algorithm, and the cWY algorithm on PowerXCell $T^M_8i$ processor.

In the experiments, we use GigaAccel 180, which is a PCI Express board with PowerXCell $T^M_8i$ processor. PowerXCell $T^M_8i$ processor is one of Cell Broadband Engine $T^M$. The theoretical performances of a single and double precision floating-point arithmetic operation on an SPE in PowerXCell $T^M_8i$ processor are 180GFLOPS and 90GFLOPS in 2.8GHz, respectively. We implement those algorithms by using Cell SDK 3.1 [2], which is developed by the IBM corporate [14]. Cell SDK 3.1 includes the parallelized BLAS for Cell Broadband Engine $T^M$. The MGS algorithm is implemented in Cell SDK 3.1.

As experimental matrices, we use three types. Type 1 is a random matrix, of which elements are set to the random number on the interval from 0 to 1. Type 2 is shown as follows:

\[
\begin{bmatrix}
1 & 1 & \\
1 & 1 & 1 \\
\vdots & \ddots & \ddots \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix},
\]

(27)

Type 3 is the glued-Wilkinson matrix $W_g^i$, which is real symmetric and has dimensions on the order of thousands. The glued-Wilkinson matrix has been used to evaluate the performance of the inverse iteration algorithms as the benchmark problems of eigenvalue decomposition [6], [8]. $W_g^i$ consists of the block matrix $W_{21}^i \in \mathbb{R}^{21 \times 21}$ and the scalar parameter $\delta \in \mathbb{R}^{1 \times 1}$ and is defined as follow:

\[
W_g^i = \begin{bmatrix}
W_{21}^i & \delta & \\
\delta & W_{21}^i & \delta \\
\delta & \delta & W_{21}^i
\end{bmatrix},
\]

(28)

where $W_{21}^i$ is defined by

\[
W_{21}^i = \begin{bmatrix}
10 & 1 & \\
1 & 9 & 1 \\
& \ddots & \ddots \\
& 0 & \ddots \\
& & \ddots & 1 \\
& & & 1 & 10
\end{bmatrix},
\]

(29)

and $\delta$ satisfies $0 < \delta < 1$ and is also the semi-diagonal element of $W_g^i$. Since $W_g^i$ is real symmetric tridiagonal and its semi-diagonal elements are nonzero, all the eigenvalues of
\[ W_j \] are distinct and real, and they are divided into 21 clusters of close eigenvalues. When \( \delta \) is small, the distance between the minimum and maximum eigenvalues in any cluster is small. In our experiments, we set \( \delta = 10^{-4} \).

Figure 7 and Table 2 show the experimental results of the orthogonalization algorithms. Time in Table 2 is the computational time. \( \| AV - VD \|_F \) and \( \| V^TV - I \|_F \) mean the Frobenius norm of synchronization and orthogonalization, respectively.

In type 1, each eigenvalue is usually separated. On the other hand, eigenvalues in type 2 and 3 become cluster. Therefore, \( \| AV - VD \|_F \) and \( \| V^TV - I \|_F \) are smaller than that in type 2 and 3.

In 2100 dimension size of type 1 in Table 2, the computational time in the MGS algorithm, which is implemented by the IBM corporate, is smaller than that in the other orthogonalization algorithms. However, the increasing rate of the computational time in the MGS algorithm is higher as shown in Figure 7(a). The MGS algorithm is computed using BLAS level-1. On the other hand, the CGS algorithm and the cWY algorithm are almost computed using BLAS level-2. Therefore, in the computational time, the CGS algorithm and the cWY algorithm are better.

In Figure 7(b) and Table 2, \( \| AV - VD \|_F \) in the CGS algorithm is nearly equal to that in the MGS algorithm. \( \| AV - VD \|_F \) of the cWY algorithm is the smallest, except the case of 6300 dimension size in type 3. The exception is likely to be caused by the order of \( v_j \). In the experiments, \( v_j \) is listed in descending order of eigenvalues, which are related to eigenvectors. Therefore, by using the cWY algorithm with suitable order of \( v_j \), accuracy of eigenvector computation can become more properly.

In type 1 and 2 of Figure 7(c), \( \| V^TV - I \|_F \) in the CGS algorithm is nearly equal to that in the MGS algorithm. The CGS algorithm and the MGS algorithm are focused on the orthogonality of eigenvectors. On the other hand, in the cWY algorithm, accuracy of eigenvalue decomposition is given importance. Therefore, the orthogonality of eigenvectors is something lower than that in the CGS algorithm and the MGS algorithm.

In type 3 of Figure 7(c), \( \| V^TV - I \|_F \) in the CGS algorithm is worse than that in the other orthogonalization algorithm. In \( \delta = 10^{-4} \), eigenvalues in type 3 are extremely close together. Therefore, the CGS algorithm is aborted that \( v_j \) is picked out.

In summarization, the computational time in the cWY algorithm is adequate speedy. Furthermore, \( \| AV - VD \|_F \) and \( \| V^TV - I \|_F \) in the cWY algorithm is sufficient accuracy.

Hence, the cWY algorithm is suitable, except case that the high-orthogonality of eigenvectors is given importance.

5. Conclusions

In this paper, we validated the parallel performance of the inverse iteration algorithms with the CGS algorithm, the MGS algorithm, and the cWY algorithm on PowerXCell8i processor. PowerXCell8i processor is one of heterogeneous environments. In ExaFLOP computing, since it is critical issue to minimize electricity, heterogeneous environments are suitable. SPEs in PowerXCell8i processor archive the high performance of BLAS level-2 and level-3. The inverse iteration algorithms are algorithms for computing eigenvectors and need a lot of computational cost. Therefore, the algorithms should be computed with SPEs. The experimental results show that the computational time
of the CGS algorithm and the cWY algorithm are shorter and $\|AV - VD\|_F$ and $\|V^TV - I\|_F$ of the cWY algorithm are sufficiently small.

In a future work, the inverse iteration algorithms should be compared on General-purpose computing on graphics processing units (GPGPU).

References


Table 2: Experimental results

<table>
<thead>
<tr>
<th>algorithm</th>
<th>time [sec.]</th>
<th>$|AV - VD|_F$</th>
<th>$|V^TV - I|_F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>type1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>10.35</td>
<td>$9.15 \times 10^{-15}$</td>
<td>$2.50 \times 10^{-14}$</td>
</tr>
<tr>
<td>MGS</td>
<td>7.32</td>
<td>$9.15 \times 10^{-15}$</td>
<td>$2.50 \times 10^{-14}$</td>
</tr>
<tr>
<td>cWY</td>
<td>13.51</td>
<td>$0.70 \times 10^{-15}$</td>
<td>$2.61 \times 10^{-14}$</td>
</tr>
<tr>
<td>(dimension size is 2100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>60.54</td>
<td>$1.25 \times 10^{-14}$</td>
<td>$3.31 \times 10^{-14}$</td>
</tr>
<tr>
<td>MGS</td>
<td>64.51</td>
<td>$1.25 \times 10^{-14}$</td>
<td>$3.32 \times 10^{-14}$</td>
</tr>
<tr>
<td>cWY</td>
<td>94.27</td>
<td>$0.067 \times 10^{-14}$</td>
<td>$3.36 \times 10^{-14}$</td>
</tr>
<tr>
<td>(dimension size is 4200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>118.52</td>
<td>$1.52 \times 10^{-14}$</td>
<td>$3.49 \times 10^{-14}$</td>
</tr>
<tr>
<td>MGS</td>
<td>478.53</td>
<td>$1.53 \times 10^{-14}$</td>
<td>$3.49 \times 10^{-14}$</td>
</tr>
<tr>
<td>cWY</td>
<td>318.04</td>
<td>$0.30 \times 10^{-14}$</td>
<td>$4.52 \times 10^{-14}$</td>
</tr>
<tr>
<td>(dimension size is 6300)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>768.40</td>
<td>$1.82 \times 10^{-14}$</td>
<td>$3.47 \times 10^{-14}$</td>
</tr>
<tr>
<td>MGS</td>
<td>5887.12</td>
<td>$1.81 \times 10^{-14}$</td>
<td>$3.47 \times 10^{-14}$</td>
</tr>
<tr>
<td>cWY</td>
<td>1408.27</td>
<td>$1.01 \times 10^{-14}$</td>
<td>$21.48 \times 10^{-14}$</td>
</tr>
<tr>
<td>(dimension size is 8400)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>43.15</td>
<td>$8.72 \times 10^{-14}$</td>
<td>$1.06 \times 10^{-13}$</td>
</tr>
<tr>
<td>MGS</td>
<td>263.82</td>
<td>$8.64 \times 10^{-14}$</td>
<td>$1.11 \times 10^{-13}$</td>
</tr>
<tr>
<td>cWY</td>
<td>78.75</td>
<td>$0.37 \times 10^{-14}$</td>
<td>$2.56 \times 10^{-13}$</td>
</tr>
<tr>
<td>(dimension size is 2100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>247.92</td>
<td>$1.79 \times 10^{-13}$</td>
<td>$1.84 \times 10^{-13}$</td>
</tr>
<tr>
<td>MGS</td>
<td>2392.14</td>
<td>$1.77 \times 10^{-13}$</td>
<td>$1.97 \times 10^{-13}$</td>
</tr>
<tr>
<td>cWY</td>
<td>456.93</td>
<td>$0.052 \times 10^{-13}$</td>
<td>$4.96 \times 10^{-13}$</td>
</tr>
<tr>
<td>(dimension size is 4200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>754.69</td>
<td>$2.64 \times 10^{-13}$</td>
<td>$2.83 \times 10^{-13}$</td>
</tr>
<tr>
<td>MGS</td>
<td>7864.63</td>
<td>$2.63 \times 10^{-13}$</td>
<td>$3.04 \times 10^{-13}$</td>
</tr>
<tr>
<td>cWY</td>
<td>1394.79</td>
<td>$0.061 \times 10^{-13}$</td>
<td>$7.45 \times 10^{-13}$</td>
</tr>
<tr>
<td>(dimension size is 8400)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>1718.53</td>
<td>$3.51 \times 10^{-13}$</td>
<td>$4.33 \times 10^{-13}$</td>
</tr>
<tr>
<td>MGS</td>
<td>16770.71</td>
<td>$3.48 \times 10^{-13}$</td>
<td>$4.53 \times 10^{-13}$</td>
</tr>
<tr>
<td>cWY</td>
<td>3186.58</td>
<td>$0.078 \times 10^{-13}$</td>
<td>$10.18 \times 10^{-13}$</td>
</tr>
<tr>
<td>(dimension size is 1200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>20.13</td>
<td>$1.11 \times 10^{-12}$</td>
<td>$1.00 \times 10^{-13}$</td>
</tr>
<tr>
<td>MGS</td>
<td>28.15</td>
<td>$1.11 \times 10^{-12}$</td>
<td>$1.07 \times 10^{-13}$</td>
</tr>
<tr>
<td>cWY</td>
<td>35.64</td>
<td>$0.18 \times 10^{-13}$</td>
<td>$1.06 \times 10^{-13}$</td>
</tr>
<tr>
<td>(dimension size is 2100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>89.47</td>
<td>$1.75 \times 10^{-12}$</td>
<td>$7.72 \times 10^{-12}$</td>
</tr>
<tr>
<td>MGS</td>
<td>202.16</td>
<td>$1.75 \times 10^{-12}$</td>
<td>$1.53 \times 10^{-12}$</td>
</tr>
<tr>
<td>cWY</td>
<td>158.18</td>
<td>$0.25 \times 10^{-12}$</td>
<td>$0.95 \times 10^{-12}$</td>
</tr>
<tr>
<td>(dimension size is 6300)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGS</td>
<td>210.98</td>
<td>$2.37 \times 10^{-11}$</td>
<td>$77.29 \times 10^{-10}$</td>
</tr>
<tr>
<td>MGS</td>
<td>678.24</td>
<td>$2.51 \times 10^{-11}$</td>
<td>$2.00 \times 10^{-10}$</td>
</tr>
<tr>
<td>cWY</td>
<td>371.78</td>
<td>$26.89 \times 10^{-11}$</td>
<td>$2.17 \times 10^{-10}$</td>
</tr>
<tr>
<td>(dimension size is 8400)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>