Accelerating the Reorthogonalization of Singular Vectors with a Multi-core Processor

Hiroki Toyokawa[1,2], Hiroyuki Ishigami[2], Kinji Kimura[2], Masami Takata[3], and Yoshimasa Nakamura[2]

[1] CyberAgent, Inc., Shibuya Markcity West 1-12-1 Dogenzaka, Shibuya-ku, Tokyo 150-0043, Japan
[2] Graduate School of Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, Japan
[3] Academic Group of Information and Computer Sciences, Nara Women's University, Kitauoyanishi-machi, Nara-city, Nara 630-8506, Japan

Abstract. The dLV twisted factorization is an algorithm to compute singular vectors for given singular values fast and in parallel. However the orthogonality of the computed singular vectors may be worse if a matrix has clustered singular values. In order to improve the orthogonality, reorthogonalization by, for example, the modified Gram-Schmidt algorithm should be done. The problem is that this process takes a longer time. In this paper an algorithm to accelerate the reorthogonalization of singular vectors with a multi-core processor is devised.

1 Introduction

Singular value decomposition (SVD) is one of most important matrix operations [1][2]. SVD is applied for data analysis, signal processing and has many other applications to engineering. Therefore SVD has been well studied for many years and some effective SVD algorithms are devised. The QR decomposition, the bisection algorithm, the divide and conquer algorithm (D&C), the MR^3 algorithm, and the I-SVD algorithm are known[1][9][8][12][13][17].

Some of those algorithms can be divided to computations of 2 phases. The first is the computation of singular values. The second is the computation of singular vectors of the computed singular values. Examples of such algorithms include the bisection algorithm and the I-SVD algorithm.

We can select an algorithm for computing singular vectors for given singular values. The dLV twisted factorization [6][7] is one of the possible choices and can compute singular vectors fast. However if a matrix has clustered singular values, the orthogonality of singular vectors computed by the dLV twisted factorization can be worse. In order to improve the orthogonality, we have to use other algorithms to compute singular vectors of such singular values. The inverse iteration algorithm with the modified Gram-Schmidt reorthogonalization can be used for this purpose, and the orthogonality becomes better indeed. The algorithm is known to be slower because the computational complexity is larger than the dLV twisted factorization. Furthermore in parallel computation with a multi-core/many-core processor, it is often that few computation cores are busy for computation but the others are idle [14][15]. Such a phenomenon occurs because the ordinary implementation of the modified Gram-Schmidt reorthogonalization is not parallelized and the task scheduling is not well-done.

In this paper, a new solution for faster computation of the SVD is devised. The main idea is as follows: we introduce a faster reorthogonalization algorithm which utilize all of the computation cores. For more efficient computation, We estimate the computation time to compute the singular vectors, and we arrange these tasks by the greedy algorithm, which is used for two-dimensional bin packing problem. The selection and arrangement of all of the tasks and algorithms for the computation are decided by auto-tuning technique. Finally we do the numerical experiments and examine the usefulness of the new reorthogonalization algorithm.

2 Algorithms of SVD Which Computes Singular Values and Singular Vectors Individually

There are some algorithms for SVD which consists of 2 phases of computation. The procedure of such algorithms is described as follows:

1. Compute singular values.
2. Compute all singular vectors of the computed singular values.

The bisection algorithm and the I-SVD algorihm can be classified to the group of algorithms. In the first phase, algoirthms which compute singular values are used. In the bisection algorithm, we prepare the Golub-Kahan matrix of the given matrix if the given matrix is bidiagonal. Then we can compute singular values by the use of the subroutine of the bisection algorithm which is implemented as the `xSTEBZ` in LAPACK. In the I-SVD algorithm, the mdLVs algorithm is used for this purpose.

In the second phase, algorithms which compute singular vectors of designated singular values are used. Examples of such algorithms include the inverse iteration and the dLV twisted factorization.

In this paper, we assume that given matrices are bidiagonal, and we use the bisection algorithm to obtain singular values.

2.1 The Hybrid Algorithm for Computing Singular Vectors

The dLV twisted decomposition can compute singular vectors of the given singular values. However it is known that the orthogonality of the computed singular vectors may be ill if the singular values are in the clusters. In order to improve the orthogonality, we have to compute singular vectors of singular values in the clusters in a different way.

The inverse iteration algorithm with the modified Gram-Schmidt reorthogonalization can also compute singular vectors. The orthogonality of the computed singular vectors is better than those of the singular vectors computed by the dLV twisted decomposition. But the reorthogonalization is slower than the dLV twisted decomposition. In this paper, we call this algorithm *the mGS algorithm.*

Thus we can compute all the singular vectors in the hybrid algorithm as follows (Fig. 1):

1. Compute singular values and some singular vectors.
2. Find clusters of close singular values.
3. Compute singular vectors corresponding to the close singular values with the mGS algorithm.
4. Compute singular vectors of the other singular values by the dLV twisted factorization.

In the first phase, all of the singular values are computed with the bisection algorithm. In the second phase, all of the computed singular values are checked whether there are any singular values which are very close to the next singular value, namely, whether the distance of the neighboring singular values is sufficiently small. In the program used for the experiments described in Sect. 4, a pair of singular values $(\sigma_i, \sigma_{i+1}) \quad (\sigma_i < \sigma_{i+1})$ are regarded as close if the pair satisfies either condition as follows:

$$\frac{\sigma_{i+1}^2 - \sigma_i^2}{\|M\|_1} < \mathrm{ortol}_1,$$
$$\frac{\sigma_{i+1} - \sigma_i}{\|M\|_1} < \mathrm{ortol}_2,$$

where σ_i is the i-th singular value, M is the given bidiagonal matrix, $\mathrm{ortol}_1 = 10^{-8}$, and $\mathrm{ortol}_2 = 10^{-3}$. Note that singular values are nonnegative real numbers. The groups of such singular values are called *clusters.* The singular vectors which are corresponding to the clustered singular values are computed in the first phase are discarded since the orthogonality may be ill.

In the third phase, the singular vectors corresponding to the clustered singular values are computed with the mGS algorithm.

In the fourth phase, singular vectors which are not in the clusters are computed by the dLV twisted factorization.

This idea of the hybrid algorithm can be applied to the I-SVD algorithm. The detail is described in [15].

2.2 Problems of the Performance

The computation time of the mGS algorithm is $O(k^2N)$, where k is the size of clusters $(1 \leq k \leq N)$. Thus if the k is larger, the whole computation time can be much longer. As a preliminary experiment, we measure the computation time to achieve SVD by the method described in Sect. 2.1. Here we prepare matrices of 2 types, M_1 and M_2. The characteristics of them is described in Table 1 and Fig. 2. This experiment is done on the computer whose features are described in Table 3.

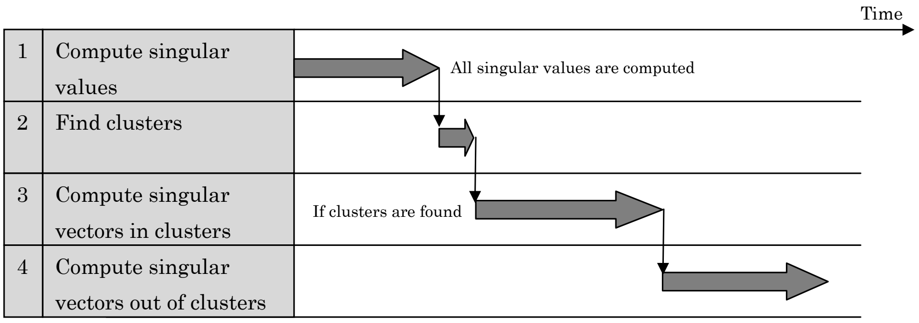
Time
1
Compute singular values
All singular values are computed
2
Find clusters
3
Compute singular vectors in clusters
If clusters are found
4
Compute singular vectors out of clusters

Fig. 1. The main flow of the hybrid algorithm

Table 1. The 3 types of matrices

Matrix type	M_1	M_2
Bidiagonal elements	Random	1
Subdiagonal elements	Random	1, 10^{-6}
Distribution of singular values	Some are clustered	Clustered

$$M_2 = \begin{pmatrix} \hat{W} & 10^{-6} & & & \\ & \hat{W} & 10^{-6} & & \\ & & \ddots & \ddots & \\ & & & \hat{W} & 10^{-6} \\ & & & & \hat{W} \end{pmatrix}, \hat{W} = \begin{pmatrix} 1 & 1 & & & \\ & 1 & 1 & & \\ & & \ddots & \ddots & \\ & & & 1 & 1 \\ & & & & 1 \end{pmatrix}$$

$\hat{W}$: $\hat{N} \times \hat{N}$ matrix, $\hat{N} = 17$.

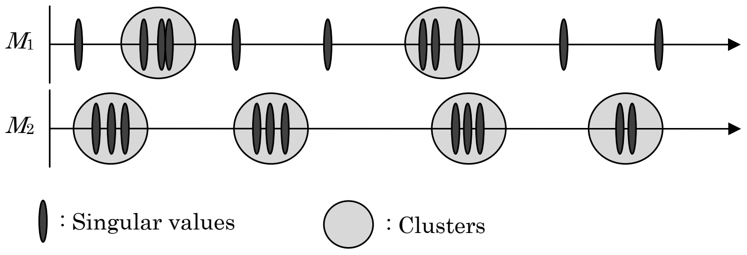

Fig. 2. The distribution of singular values

Table 2. The computation time to achieve SVD

Matrix size N	M_1	M_2
3400	2.6	4.3
6800	9.9	29.7
10200	24.2	130.3

in second: [s]

Table 2 shows the computation time of several matrix of size N. This shows that it takes much longer time to achieve SVD of M_2 compared to M_1.

Fig. 3 shows the number of the working CPU cores during the computation of the M_2 ($N = 6800$). According to the graph, only 1 core works at the latter time and it lengthens the total computation time, while the other cores are idle. It is obviously inefficient.

The M_2 contains 17 large clusters. The computer which is used for the numerical experiment has 8 computation cores. Therefore 16 ($= 8 \times 2$) clusters are computed by each cores, but the left 1 cluster is computed by 1 core after computing the 16 clusters (Fig. 4).

In this article, we accelerate the total computation of SVD by improving the above mentioned inefficiency.

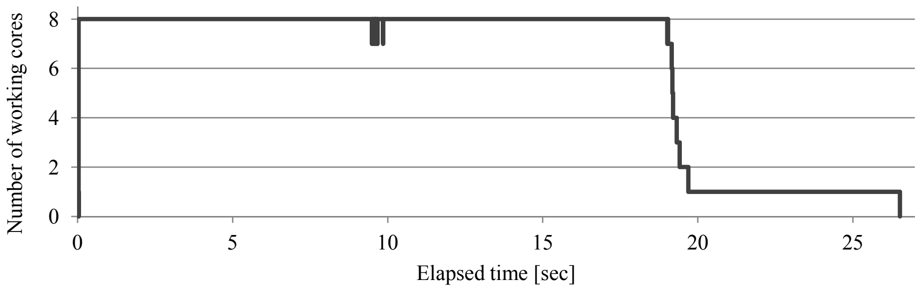

Fig. 3. The changes of the number of working cores

Table 3. The features of computers

CPU	Intel Xeon CPU X5570 2.93GHz
	(2 processors $\times$ 4 cores)
Memory	32GBytes
OS	Fedora Linux 17 (x86 64bit)
Compiler	icpc, ifort 12.1 (-O3 option)
Libraries	Intel Math Kernel Library 10.3

3 Accelerating the Reorthogonalization of Singular Vectors

In order to accelerate the reorthogonalization, we first introduce another inverse iteration method in Sect. 3.1. In Sect. 3.2, a hybrid algorithm to utilize the alternative inverse iteration method as well as the mGS algorithm is explained.

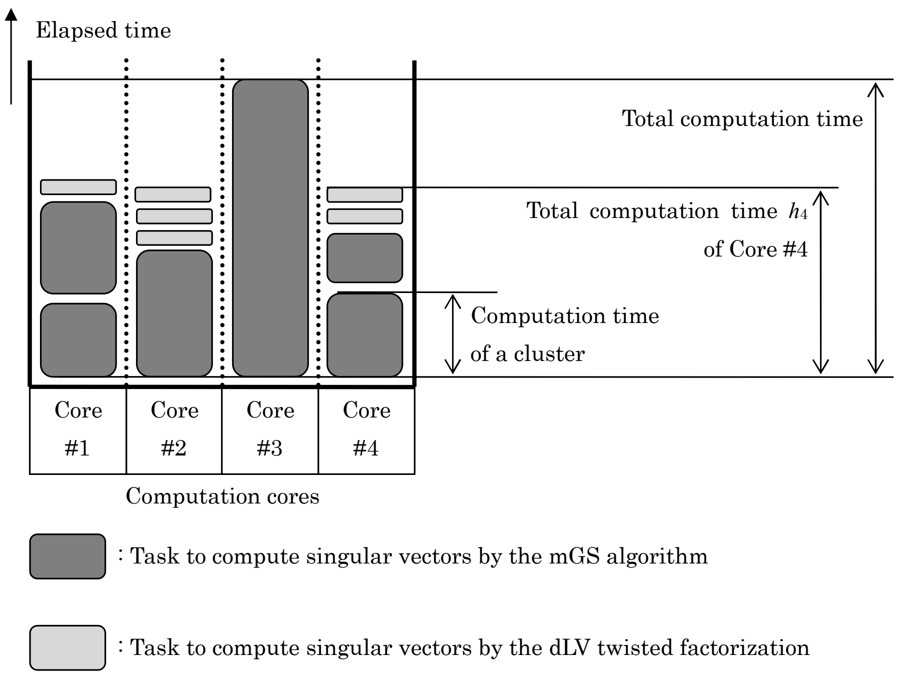
Elapsed time
Total computation time
Total computation time h_4 of Core #4
Computation time of a cluster
Core #1
Core #2
Core #3
Core #4
Computation cores
: Task to compute singular vectors by the mGS algorithm
: Task to compute singular vectors by the dLV twisted factorization

Fig. 4. An inefficient task scheduling

3.1 Inverse Iteration Method with the Compact WY Orthogonalization

The algorithm in this section is based on the idea of the use of the block Householder orthogonalization in terms of the compact WY representation[16]. The calculation mainly consists of the product of matrices and vectors and rank-1 update operation. Thus these operations can be done with the level-2 BLAS. These computations can be parallelized using the BLAS for parallel computation such as Intel Math Kernel Library, GotoBLAS 2, and so on. The detail of the algorithm is explained in [3]. In this paper, this algorithm is called *the cWY algorithm.*

During the execution of this algorithm, all of the computation cores are used for the computation. Consequently the computation finishes faster than the mGS algorithm, providing that sufficient cores are used for the computation.

3.2 Utilization of the 2 Algorithms

Using the cWY algorithm, we can accelerate the reorthogonalization by using all of the cores. However the product of *CPU time* and *the number of CPU cores* is apt to be larger than that of the inverse iteration and the modified Gram-Schmidt reorthogonalization. Therefore we should not use the cWY algorithm blindly. In fact, we should apply the mGS algorithm to small clusters, and apply

Table 4. The features of the two algorithms

Algorithm	mGS	cWY
Required computation cores for 1 cluster	1 core	All cores
Computation time	Longer	Shorter

mGS: The inverse iteration and the modified Gram-Schmidt reorthogonalization
cWY: The cWY algorithm

the cWY algorithm to large clusters, which takes much longer time. Table 4 shows the features of the 2 algorithms.

To utilize the 2 algorithms in a well-defined hybrid form, we have to keep in mind the 2 points as follows:

1. The algorithm which is applied to a cluster should be selected properly.
2. Computation for all the clusters should be scheduled to each other properly.

These points are described in the next paragraph.

Selection of the Algorithm. Judging from Table 4, the main strategy for selecting algorithm is that: for clusters of small number of singular values, the modified Gram-Schmidt reorthogonalization should be selected. For large clusters, the cWY algorithm should be selected. For more proper selection, we should estimate the computation time for each cluster using the 2 algorithms, and select the faster one.

However it is difficult to determine the obvious conditions and thresholds to judge which algorithm is faster before the computation starts.

It is known that the computational complexity of mGS algorithm is $O(k^2N)$, and that of the cWY algorithm is $O(k^2N)$, except for coefficients. First we collect enough sample data of $\{N, k, t_{a,\mathrm{mGS}}, t_{a,\mathrm{cWY}}\}$, where $t_{a,\mathrm{mGS}}$ is the actual time to compute singular vectors by the mGS algorithm, and $t_{a,\mathrm{cWY}}$ is the actual time to compute singular vectors by cWY algorithm. Then we can obtain the estimated computation times $t_{e,\mathrm{mGS}}$ and $t_{e,\mathrm{cWY}}$ of the given $\{N, k\}$ by using the least-squares method, where $t_{e,\mathrm{mGS}}$ is the estimated computation time of the modified Gram-Schmidt algorithm and $t_{e,\mathrm{cWY}}$ is the estimated computation time of the cWY algorithm, respectively.

Assuming that enough data of $\{N, k, t_{a,\mathrm{cWY}}\}$ are given, the procedure to estimate the computation time $t_{e,\mathrm{cWY}}$ of the cWY algorithm is as follows:

1. Fix k and regard $t_{e,\mathrm{cWY}}$ as a linear function of N, and estimate its value at the given N by the least-squares method by using $\{t_{a,\mathrm{cWY}}\}$.
2. Repeat Step 1. with several values $k = \{k_i\}$, and estimate $\{t_{e,\mathrm{cWY}}\}_{k=k_i}$.
3. Regard $\{t_{e,\mathrm{cWY}}\}_{k=k_i}$ as a sample data at the given N, and estimate $t_{e,\mathrm{cWY}}$ for given $\{N, k\}$ by the least-squares method.

The computation time $t_{e,\mathrm{mGS}}$ can also be estimated by the same way.

Scheduling the Tasks. For more acceleration, the computation should also be scheduled well. Scheduling tasks of computing singular vectors in clusters can be regarded as the two-dimensional bin packing problem. By scheduling, the total computation time should be shorter, but we should not consume time for the scheduling itself. Therefore we adopt the greedy algorithm, which can make so good answer and does not cost so longer time.

Assuming that computation time of all of the tasks are estimated by the technique described above, the procedure of the greedy algorithm which is used in this paper is as follows:

1. Compute the current total computation time $\{h_i\}$ of each cores, where h_i means the total computation time of tasks which are assigned to core $\#i$ (see Fig. 4).
2. Assign a task to the core whose total computation time h_i is minimum.
3. Go to Step 1 until all of the tasks are assigned.

3.3 The New Algorithm

Based on the techniques above, the procedure of the new algorithm can be described as follows:

1. Only once: Measure the actual computation time for some combinations of parameters $\{N, k, t_a\}$ and collect the measured sample data.
2. Compute all the singular values by the mdLVs algorithm.
3. Find clusters of close singular values.
4. Select algorithms and schedule the tasks.
   (a) Estimate the computation time of all the tasks.
   (b) Arrange the tasks by the estimated computation time in descending order.
   (c) Set $m \leftarrow 0$.
   (d) Estimate the computation time $\left\{t_{e,\mathrm{mGS}}^{(i)}\right\}_{i=m+1}^{M}, \left\{t_{e,\mathrm{cWY}}^{(i)}\right\}_{i=1}^{m}$ of the clusters, and set the estimated computation time $\left\{t_e^{(i)}\right\}_{i=1}^{M} = \begin{cases} t_{e,\mathrm{mGS}}^{(i)} & (m < i \leq M) \\ t_{e,\mathrm{cWY}}^{(i)} & (1 \leq i \leq m) \end{cases}$.
   (e) Arrange the tasks to minimize the total computation time by the greedy algorithm.
   (f) Check whether the total estimated computation time is shorter than that of previous trial. If the time is not shorter, we adopt the previous selection of algorithms and task schedule, and go to Step 5
   (g) Set $m \leftarrow m + 1$ and go to Step 4d.
5. Compute all the singular vectors in clusters by the selected algorithms.
6. Compute other singular vectors by the dLV twisted factorization.

The Step 1 should be done only once. The results of the step can be stored in a file, and we can load the results from the file and can reuse them. After the procedure, the computation become more efficient and the total computation time will be shorter (Fig. 5). The Step 1 should be only once. Once this step is done, we do not have to do it again. Therefore this step should be done before the computation, for example, on installing the library or at night. The Step 1 and Step 4 are added to the original procedure described in Sect. 2.1.

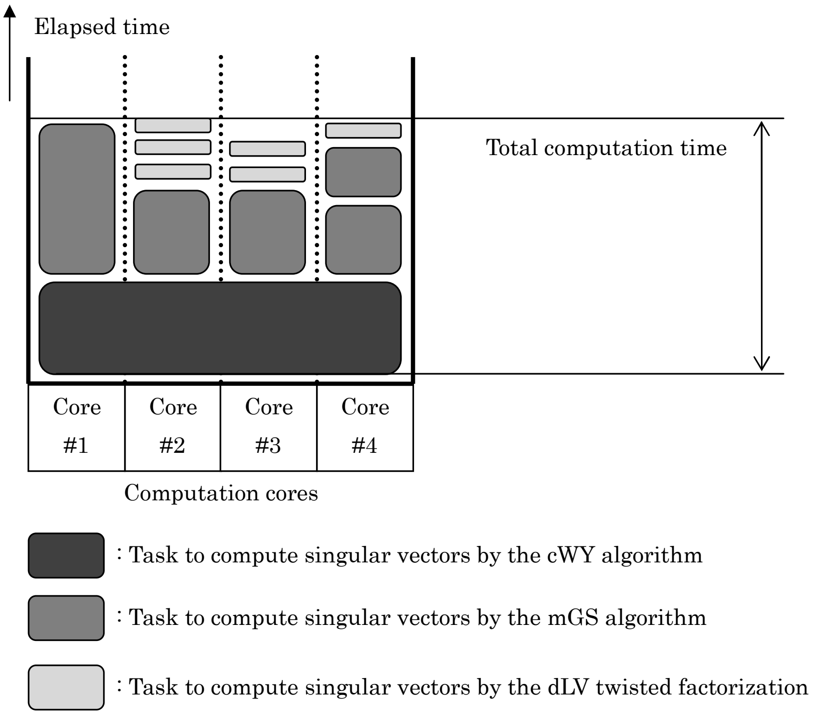
Elapsed time
Total computation time
Core #1
Core #2
Core #3
Core #4
Computation cores
: Task to compute singular vectors by the cWY algorithm
: Task to compute singular vectors by the mGS algorithm
: Task to compute singular vectors by the dLV twisted factorization

Fig. 5. An efficient task scheduling

4 Numerical Experiments

In order to examine the effect of the new algorithm, we give numerical experiments. In the experiments, we see the length of shortened time by the new algorithm. The computers described in Table 3 are used for the experiments. For the experiments, we use the 2 types of matrices $\{M_1, M_2\}$, which are described in Sect. 2.2. We use several size N of these matrices.

In the experiments, we use the GotoBLAS2 for matrix operations. This library is called parallelly and individually by working threads, which execute their own tasks. During the computation of the mGS algorithm, the operation of the library in 1 task should be done by only 1 core. In order to force the policy, we use `omp_set_num_threads` function to adjust the number of threads which are used for matrix operations.

The results of the experiments are shown in Table 5 for M_1, and in Table 6 for M_2. The experiments are done under the condition that the number of cores allocated for the experiments is 1, 2, 4, and 8.

Result of M_1. According to Table 5, any effect in acceleration of the new algorithm is not observed. That is because the new algorithm accelerates only the computation for clusters. Table 7 shows the distribution of the size of the clusters of M_1 ($N = 20400$). According to the table, the size of most of the clusters are less than 10. Only 1 cluster is larger, but the cluster size is not so large compared to that of M_2. Thus the effect of the new algorithm is so little.

Result of M_2. According to Table 6, the new algorithm accelerates the orthogonalization if N is large enough and 8 cores are used. The most computation for M_2 consists of the reorthogonalization because there are large clusters. The size of clusters $k = N/17$. Furthermore at the end of the computation, the computation of the remaining 1 large core is done on 1 core while other cores are idle. Thus the effect of the new algorithm can be observed if enough cores are used for the cWY algorithm. In this computer, the cWY algorithm is slower than the mGS algorithm if 2 cores are used. Thus the cWY algorithm is not used when 2 cores are used, and the computation time is not shorten by the proposed algorithm.

Table 5. The computation time of M_1

Matrix	Maximum cluster	1 core	2 cores		4 cores		8 cores	
size N	size max $\{k\}$	Org.	Org.	New	Org.	New	Org.	New
3400	52	16	8	9	5	5	3	3
6800	95	64	33	33	18	18	10	10
10200	169	143	73	75	42	42	24	24
13600	237	253	131	135	75	74	45	45
17000	296	396	207	210	119	118	73	72
20400	369	573	306	307	177	176	110	110

Org.: The original algorithm, New: The new algorithm
in second : [s]

Table 6. The computation time of M_2

Matrix	Maximum cluster	1 core	2 cores		4 cores		8 cores	
size N	size max $\{k\}$	Org.	Org.	New	Org.	New	Org.	New
3400	200	16	8	10	6	5	4	4
6800	400	111	58	59	41	42	28	30
10200	600	390	201	203	139	143	126	130
13600	800	912	472	474	328	324	308	304
17000	1000	1725	899	903	639	633	598	585
20400	1200	2901	1509	1509	1059	1065	1037	1020

Org.: The original algorithm, New: The new algorithm
in second : [s]

Table 7. The distribution of the size of clusters ($N = 20400$, Matrix: M_1)

Cluster size k	Count
2	65
3	6
4	5
5	1
6	1
10	1
14	1
25	1
369	1

5 Conclusion

In this paper, a new method to shorten the time for reorthogonalization of singular vectors and then the time for SVD. If the singular values of a given matrix are heavily clustered, the reorthogonalization process is apt to take much longer time in the previous works[14][15]. The new algorithm in this paper can shorten this process by using 1) the cWY algorithm, 2) selecting algorithm to reorthogonalization, and scheduling the tasks. The new algorithm especially works well, providing that there are some large clusters and the computation for the large cluster is done on few cores while other tasks completes. The effect of the new algorithm is checked by some numerical experiments.

As a future work, the predominance in performance and accuracy of the new algorithm should be examined using more types of matrices. Test matrices which have designated singular values or whose conditional number is large can be generated[10][11]. The new algorithm is desired to be compared to existing algorithms implemented in LAPACK.

References

1. Demmel, J.W.: Applied Numerical Linear Algebra. SIAM, Philadelphia (1997)
2. Golub, G., Van Loan, C.: Matrix Computation, 3rd edn. John Hopkins Univ. Press, Baltimore (1996)
3. Ishigami, H., Kimura, K., Nakamura, Y.: Implementation and performance evaluation of inverse iteration with new reorthogonalization algorithm. In: Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2011), vol. II, pp. 775–780 (2011)
4. Iwasaki, M., Nakamura, Y.: Accurate computation of singular values in terms of shifted integrable schemes. Japan Journal of Industrial and Applied Mathematics 23, 239–259 (2006)
5. Iwasaki, M., Nakamura, Y.: Positivity of dLV and mdLVs algorithms for computing singular values. Electronic Transactions on Numerical Analysis 38, 184–201 (2011)
6. Iwasaki, M., Sakano, S., Nakamura, Y.: Accurate twisted factorization of real symmetric tridiagonal matrices and its application to singular value decomposition. Transactions of the Japan Society for Industrial and Applied Mathematics 15, 461–481 (2005) (in Japanese)

7. Konda, T., Takata, M., Iwasaki, M., Nakamura, Y.: A new singular value decomposition algorithm suited to parallelization and preliminary results. In: Proceedings of IASTED International Conference on Advances in Computer Science and Technology (ACST 2006), pp. 79–85 (2006)
8. Nakamura, Y.: Functionality of Integrable System. Kyoritsu Publishing, Tokyo (2006) (in Japanese)
9. Parlett, B.N., Dhillon, I.S.: Relatively robust representations of symmetric tridiagonals. Linear Algebra Appl. 309, 121–151 (2000)
10. Takata, M., Kimura, K., Iwasaki, M., Nakamura, Y.: Algorithms for generating bidiagonal test matrices. In: Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2007), pp. 732–738 (2007)
11. Takata, M., Kimura, K., Nakamura, Y.: Generating algorithms for matrices with large condition number to evaluate singular value decomposition. In: Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010), pp. 619–625 (2010)
12. Takata, M., Kimura, K., Iwasaki, M., Nakamura, Y.: Performance of a new scheme for bidiagonal singular value decomposition of large scale. In: Proceedings of IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2006), pp. 304–309 (2006)
13. Takata, M., Kimura, K., Iwasaki, M., Nakamura, Y.: Implementation of library for high speed singular value decomposition. Journal of Information Processing Society of Japan 47 SIG7(ACS 14), 91–104 (2006)
14. Toyokawa, H., Kimura, K., Takata, M., Nakamura, Y.: On parallelism of the I-SVD algorithm with a multi-core processor. JSIAM Letters 1, 48–51 (2009)
15. Toyokawa, H., Kimura, K., Takata, M., Nakamura, Y.: On parallelization of the I-SVD algorithm and its evaluation for clustered singular values. In: Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2009), pp. 711–717 (2009)
16. Yamamoto, Y., Hirota, Y.: A parallel algorithm for incremental orthogonalization based on the compact WY representation. JSIAM Letters 3, 89–92 (2011)
17. I-SVD Library, `http://www-is.amp.i.kyoto-u.ac.jp/lab/isvd/download/`