Performance Evaluation of Golub-Kahan-Lanczos Algorithm with Reorthogonalization by Classical Gram-Schmidt Algorithm and OpenMP

Masami Takata[1], **Hiroyuki Ishigami**[2], **Kinji Kimura**[2],
Yuki Fujii[2], **Hiroki Tanaka**[2], **and Yoshimasa Nakamura**[2]
[1]Research Group of Information and Communication Technology for Life,
Nara Women's University, Nara, Nara, JAPAN
[2]Graduate School of Informatics, Kyoto University, Kyoto, Kyoto, JAPAN

Abstract—*The Golub-Kahan-Lanczos algorithm with reorthogonalization (GKLR algorithm) is an algorithm for computing a subset of singular triplets for large-scale sparse matrices. The reorthogonalization tends to become a bottleneck of elapsed time, as the iteration number of the GKLR algorithm increases. In this paper, OpenMP-based parallel implementation of the classical Gram-Schmidt algorithm with reorthogonalization (OMP-CGS2 algorithm) is introduced. The OMP-CGS2 algorithm has the advantage of data reusability and is expected to achieve higher performance of the reorthogonalization computations on shared-memory multi-core processors with large caches than the conventional reorthogonalization algorithms. Numerical experiments on shared-memory multi-core processors show that the OMP-CGS2 algorithm accelerates the GKLR algorithm more effectively for computing a subset of singular triplets for a sparse matrix than the conventional reorthogonalization algorithms.*

Keywords: Subset computation of singular triplets, Golub-Kahan-Lanczos algorithm with reorthogonalization, Classical Gram-Schmidt algorithm with reorthogonalization, OpenMP, Shared-memory multi-core processing

1. Introduction

Let A be a real $m \times n$ matrix and $\mathrm{rank}(A) = r$ ($r \le \min(m, n)$). Then A has the r singular values $\sigma_1, \ldots, \sigma_r \in \mathbb{R}$, which satisfies $\sigma_1 \ge \cdots \ge \sigma_r > 0$, and their corresponding left and right singular vectors $\boldsymbol{u}_i \in \mathbb{R}^m$, $\boldsymbol{v}_i \in \mathbb{R}^n$ ($1 \le i \le r$). A subset of singular triplets, i.e. the l largest singular values $\sigma_1, \ldots, \sigma_l$ and their corresponding singular vectors, is often required in low-rank matrix approximation [17] and statistical processings such as principal component analysis and the least-squares method. In such applications, the target matrix is often large and sparse, and l is often much smaller than both m and n. It is difficult to perform the computation of singular triplets directly from a large-scale sparse matrix because of the computational cost and need for large amounts of memory.

The Krylov subspace methods are better for such computations. They transform the target matrix into a significantly smaller matrix than the target matrix and the singular values of the generated matrix sufficiently approximate a subset of singular values of the target matrix. The Golub-Kahan-Lanczos (GKL) algorithm [5], [6] is one of the Krylov subspace methods and generates approximate bidiagonal matrices from the target matrix. However, the GKL algorithm usually loses the orthogonality of the Krylov subspace because of the computational error. To improve the orthogonality, let us incorporate a reorthogonalization process into the GKL algorithm. Such an algorithm is referred to as the GKL algorithm with reorthogonalization (GKLR algorithm) [1]. Note that these algorithms are generally parallelized in terms of the Basic Linear Algebra Subprograms (BLAS) [12], such as the matrix multiplications and the matrix-vector multiplications, because they are iterative algorithms. In addition, we implement the bisection algorithm and the inverse iteration algorithm [10], [8] for computing a subset singular triplets of the approximate matrices generated by the GKLR algorithm.

Although the GKLR algorithm is stable because of the reorthogonalization, the reorthogonalization tends to become a bottleneck in terms of the computational cost and the elapsed time as the iteration number increases. However, since the reorthogonalization of the GKLR algorithm is mainly implemented using the matrix-vector multiplications, even in parallel computing, the reorthogonalization is not effectively accelerated and then the overall elapsed time of the GKLR algorithm is not effectively reduced.

In this paper, to accelerate the reorthogonalization of the GKLR algorithm more effectively in parallel computing, we introduce a parallel implementation of the classical Gram-Schmidt algorithm with reorthogonalization (CGS2 algorithm) [3], which is parallelized using the OpenMP [13]. Hereafter, this implementation of the CGS2 algorithm is referred to as the OMP-CGS2 algorithm. This parallelization technique enables to use the cache of CPUs effectively and then the computation is expected to be accelerated more effectively than the conventional reorthogonalization algorithms, which are parallelized in terms of the BLAS operations.

The rest of this paper is organized as follows. In Section 2, the GKLR algorithm and its implementation in this paper are described. In Section 3, a BLAS-based parallel implementation of reorthogonalization algorithms and the OMP-CGS2 algorithm are presented. Section 4 provides performance evaluations of the OMP-CGS2 algorithm on multi-core processors. We end with conclusions and future works in Section 5.

2. GKLR algorithm

This section considers the GKLR algorithm and describes the implementation of the GKLR algorithm in this paper.

2.1 GKLR algorithm

The GKL [5], [6] algorithm generates new bases $\boldsymbol{p}_k \in \mathbb{R}^n$ and $\boldsymbol{q}_k \in \mathbb{R}^m$ at the k-th iteration. The $\boldsymbol{p}_k$ is an orthonormal basis of the Krylov subspace $\mathcal{K}(A^\top A,\ \boldsymbol{p}_1,\ k)$, and the $\boldsymbol{q}_k$ is an orthonormal basis of the alternative Krylov subspace $\mathcal{K}(AA^\top, A\boldsymbol{p}_1,\ k)$. In the GKLR algorithm [1], each time a new basis is added with the expansion of the Krylov subspace, the existing orthonormal basis, and the new basis are reorthogonalized.

Algorithm 1 shows the pseudocode of the GKLR algorithm. Lines 6 and 10 show the reorthogonalization process, respectively. At the beginning of the k-th iteration for $k = 1,\ 2,\ \ldots$ in Algorithm 1, the $k \times k$ approximate matrices

$$B_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & \alpha_k \end{bmatrix} \qquad (1)$$

are obtained and the following equations hold

$$AP_k = Q_k B_k, \qquad (2)$$
$$A^\top Q_k = P_k B_k^\top + \beta_k \boldsymbol{p}_{k+1} \boldsymbol{e}_k^\top, \qquad (3)$$

where $\boldsymbol{e}_k$ is the k-th column of the $k \times k$ identity matrix. Note that if the l largest singular values of B_k sufficiently approximate those of A, we can stop the iterations of the GKLR algorithm. On line 8 in Algorithm 1 , we check whether the l largest singular values of B_k sufficiently approximate those of A or not. Criteria for this check are discussed in Sec. 2.2.1.

Let $\sigma_j^{(k)}$, $\boldsymbol{s}_j^{(k)} \in \mathbb{R}^k$, and $\boldsymbol{t}_j^{(k)} \in \mathbb{R}^k$ $(j = 1,\ \ldots,\ k)$ be a singular value of B_k, the left singular vector, and the right singular vector corresponding to $\sigma_j^{(k)}$, respectively. If $\sigma_j^{(k)}$ approximates σ_j well, then $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ corresponds to $\boldsymbol{u}_j^{(k)}$ and $\boldsymbol{v}_j^{(k)}$ defined as the following equations, respectively:

$$\boldsymbol{u}_j^{(k)} = Q_k \boldsymbol{s}_j^{(k)},\ \boldsymbol{v}_j^{(k)} = P_k \boldsymbol{t}_j^{(k)}. \qquad (4)$$

Algorithm 1 GKLR algorithm

```
 1: Set an $n$-dimensional unit vector $\boldsymbol{p}_1$
 2: $\boldsymbol{q} = A\boldsymbol{p}_1,\ \alpha_1 = \|\boldsymbol{q}\|_2,\ \boldsymbol{q}_1 = \boldsymbol{q}/\alpha_1$
 3: $P_1 = [\boldsymbol{p}_1], Q_1 = [\boldsymbol{q}_1]$
 4: do $k = 1,\ 2,\ \ldots$
 5:     $\boldsymbol{p} = A^\top \boldsymbol{q}_k$
 6:     $\tilde{\boldsymbol{p}} = \mathrm{Reorthogonalization}(P_k,\ \boldsymbol{p})$
 7:     $\beta_k = \pm\|\tilde{\boldsymbol{p}}\|_2,\ \boldsymbol{p}_{k+1} = \tilde{\boldsymbol{p}}/\beta_k$
 8:     Check the singular values of $B_k$
 9:     $\boldsymbol{q} = A\boldsymbol{p}_{k+1}$
10:     $\tilde{\boldsymbol{q}} = \mathrm{Reorthogonalization}(Q_k,\ \boldsymbol{q})$
11:     $\alpha_{k+1} = \pm\|\tilde{\boldsymbol{q}}\|_2,\ \boldsymbol{q}_{k+1} = \tilde{\boldsymbol{q}}/\alpha_{k+1}$
12:     $P_{k+1} = \begin{bmatrix} P_k & \boldsymbol{p}_{k+1} \end{bmatrix},\ Q_{k+1} = \begin{bmatrix} Q_k & \boldsymbol{q}_{k+1} \end{bmatrix}$
13: end do
```

In order to improve the accuracy of singular vectors, this computation is implemented to the combination with the QR factorization [11].

As seen in Algorithm 1, the GKLR algorithm must be parallelized in terms of the computations on each line. Since the computation on each line can be implemented using the BLAS operations, we parallelize the GKLR algorithm in terms of each the BLAS operations.

2.2 Implementation of GKLR algorithm

In this section, we discuss the methods to check whether the singular values of B_k approximate sufficiently those of A or not. We then introduce a stopping strategy of the GKLR algorithm and the implementation for the subset computation of singular triplets for approximate matrices in this paper.

2.2.1 Stopping strategy of GKLR algorithm

Recalling $(\sigma_j^{(k)},\ \boldsymbol{s}_j^{(k)}\ \boldsymbol{t}_j^{(k)})$, the j-th singular triplets for B_k $(j = 1,\ \ldots,\ l)$, we then have the following equations:

$$B_k \boldsymbol{t}_j^{(k)} = \sigma_j^{(k)} \boldsymbol{s}_j^{(k)},\ B_k^\top \boldsymbol{s}_j^{(k)} = \sigma_j^{(k)} \boldsymbol{t}_j^{(k)}. \qquad (5)$$

Using Eqs. (2), (3), (4), and (5), we obtain

$$\begin{aligned} A^\top \boldsymbol{u}_j^{(k)} - \sigma_j^{(k)} \boldsymbol{v}_j^{(k)} &= A^\top Q_k \boldsymbol{s}_j^{(k)} - \sigma_j^{(k)} P_k \boldsymbol{t}_j^{(k)} \\ &= \left(A^\top Q_k - P_k B_k^\top\right) \boldsymbol{s}_j^{(k)} \\ &= \beta_k \boldsymbol{p}_{k+1} \boldsymbol{e}_k^\top \boldsymbol{s}_j^{(k)} \\ &= \beta_k s_j^{(k)}(k) \boldsymbol{p}_{k+1}, \end{aligned} \qquad (6)$$

where $s_j^{(k)}(k)$ is the k-th element of $\boldsymbol{s}_j^{(k)}$. Thus, the following inequality holds:

$$\left\| A^\top \boldsymbol{u}_j^{(k)} - \sigma_j^{(k)} \boldsymbol{v}_j^{(k)} \right\|_2 \leq \left| \beta_k s_j^{(k)}(k) \right|. \qquad (7)$$

As the results, if the right-hand side of inequality (7) is sufficiently small, then the singular value $\sigma_j^{(k)}$ of B_k can be regarded to sufficiently approximate that of A. Hence, the

Algorithm 2 Stopping strategy of GKLR algorithm

```
1: Compute $(\sigma_l^{(k)},\ \boldsymbol{s}_l^{(k)},\ \boldsymbol{t}_l^{(k)})$
2: if $\left|\beta_k s_l^{(k)}(k)\right| \leq \delta$, then
3:     Compute $(\sigma_j^{(k)},\ \boldsymbol{s}_j^{(k)},\ \boldsymbol{t}_j^{(k)})$ for $j = 1,\ \ldots,\ l$
4:     if $\left|\beta_k s_j^{(k)}(k)\right| \leq \delta$ for $j = 1,\ \ldots,\ l$, then
5:         Stop the iteration of GKLR algorithm
6:     end if
7: end if
```

following inequality can be considered as one of the stopping criteria of the GKLR algorithm:

$$\left|\beta_k s_j^{(k)}(k)\right| \leq \delta, \quad j = 1,\ \ldots,\ l, \tag{8}$$

where δ is a threshold value for stopping the iteration of the GKLR algorithm and determined arbitrarily by users. If we use this criterion based on inequality (8), we have to compute the l singular triplets for B_k, i.e. $(\sigma_j^{(k)},\ \boldsymbol{s}_j^{(k)},\ \boldsymbol{t}_j^{(k)})$, $j = 1,\ \ldots,\ l$, before checking if inequality (8) is satisfied. The computational cost of computing singular triplets for B_k is more expensive than that of checking if inequality (8). In order to reduce the total elapsed time for the GKLR algorithm, the computational cost of computing singular triplets for B_k has to be reduced. Hereafter, let k_{t} be the number of iterations where inequality (8) is satisfied for the first time.

Now let us consider the following inequality, which is one of the necessary conditions for inequality (8):

$$\left|\beta_k s_l^{(k)}(k)\right| \leq \delta. \tag{9}$$

We have only to compute the l-th largest singular triplet for B_k in order to check if inequality (9) is satisfied. Hence, from the viewpoint of the computational cost, inequality (9) is more suitable for the stopping criterion of the GKLR algorithm than inequality (8). In addition, if let k_{n} be the number of iterations where satisfy inequality (9) for the first time, it is observed that $k_{\mathrm{t}} = k_{\mathrm{n}}$ in many cases of numerical experiments. From these facts, inequality (9) can be also considered as one of the stopping criteria of the GKLR algorithm. However, since the theorems in [14] imply that the value of k_{t} depends on the distribution of singular values for the target matrix, $k_{\mathrm{t}} = k_{\mathrm{n}}$ is not always guaranteed. Thus, even if inequality (9) is satisfied, we must check if inequality (8) is also satisfied for all j.

Summarizing the above discussions, the stopping strategy for the GKLR algorithm is shown by Algorithm 2. In the experiments mentioned in Sec. 4, we set $\delta = 1.0 \times 10^{-14}$ as the stopping criterion. Note that Algorithm 2 is used on line 8 in Algorithm 1. After stopping the iteration of the GKLR algorithm, we compute the l largest singular triplets of A, i.e. $(\sigma_j,\ \boldsymbol{u}_j,\ \boldsymbol{v}_j)$ for $j = 1,\ \ldots,\ l$, using Eqs. (4).

2.2.2 Subset computation algorithms for singular triplets of approximate matrices

As mentioned in Section 2.2.1, a subset of singular triplets for the approximate matrices is required for stopping the GKLR algorithm. In this subsection, we discuss the subset computations of singular triplets of the approximate matrices on lines 1 and 3 in Algorithm 2.

The approximate matrix B_k, generated by the GKLR algorithm, is a lower bidiagonal matrix. As mentioned in [5], the singular value problem of the bidiagonal matrix can be transformed into the eigenvalue problem of the symmetric tridiagonal matrix without any computational cost. From the above fact, the singular triplets of the lower bidiagonal matrix can be obtained using the bisection algorithm and the inverse iteration algorithm (BI algorithm) for symmetric tridiagonal matrices [10], [8]. The BI algorithm enables us to compute only the required eigenpairs and is suitable for the subset computation of singular triplets in Algorithm 2. While computing l singular triplets (line 3 in Algorithm 2), we parallelize the subset computation of singular triplets as follows: The bisection algorithm is parallelized in terms of each singular value, and the inverse iteration algorithm is parallelized in terms of the BLAS operations.

3. Reorthogonalization algorithms

To improve the orthogonality of the Krylov subspace and the accuracy of the resulting singular vectors, the reorthogonalization is inevitable for the GKLR. However, the computational cost of the reorthogonalization is larger than the other processes of the GKLR, as the iteration number increases. Thus, it is important to accelerate the reorthogonalization in the GKLR.

In this section, at first, we consider three conventional reorthogonalization algorithms for the GKLR algorithm. The classical Gram-Schmidt with reorthogonalization (CGS2) algorithm [3], the modified Gram-Schmidt (MGS) algorithm [6], and the reorthogonalization algorithm using the Householder transformations in terms of the compact WY representation (cWY algorithm) [19], [9]. These algorithms are parallelized in terms of the BLAS operations in recent days. Secondly, we present the OpenMP-based parallel implementation of the CGS2 algorithm for shared-memory multi-core processors and describe the advantage of this implementation with respect to the data usability.

In the followings, we discuss the computation of $\boldsymbol{x}_i \in \mathbb{R}^m$, the reorthogonalized vector of $\boldsymbol{a}_i \in \mathbb{R}^m$ $(2 \leq i \leq n)$, where satisfies $\langle \boldsymbol{x}_i,\ \boldsymbol{x}_k \rangle = 0$ for $j \neq k$. In addition, let X_{i-1} be $X_{i-1} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_{i-1} \end{bmatrix}$ $(2 \leq i \leq n)$. Note that X_{i-1}, $\boldsymbol{x}_i$, and $\boldsymbol{a}_i$ correspond to P_k, $\tilde{\boldsymbol{p}}$, and $\boldsymbol{p}$ on line 6 in Algorithm 1, and also correspond to Q_k, $\tilde{\boldsymbol{q}}$, and $\boldsymbol{q}$ on line 10 in Algorithm 1.

Algorithm 3 CGS2 algorithm

```
1: function CGS2($X_{i-1}(= [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}])$, $\boldsymbol{a}_i$)
2:     do $j = 1, 2$
3:         $\boldsymbol{w} = X_{i-1}^{\top}\boldsymbol{a}_i$
4:         $\boldsymbol{a}_i = \boldsymbol{a}_i - X_{i-1}\boldsymbol{w}$
5:     end do
6:     return $\boldsymbol{x}_i = \boldsymbol{a}_i$
7: end function
```

3.1 BLAS-based parallel implementation algorithms

3.1.1 CGS2 algorithm

The classical Gram-Schmidt (CGS) algorithm [6] is a well-known reorthogonalization algorithm. The reorthogonalization of $\boldsymbol{a}_i$ using the CGS algorithm is formulated as follows:

$$\boldsymbol{x}_i = \boldsymbol{a}_i - \sum_{k=1}^{i-1} \langle \boldsymbol{x}_k,\ \boldsymbol{a}_i \rangle \boldsymbol{x}_k. \tag{10}$$

Eq. (10) is composed of Level 1 BLAS operations, such as inner-dot products and AXPY operations. The computational cost of the CGS algorithm is about $2mk^2$ if the reorthogonalization of $\boldsymbol{a}_i$ for $i = 1, \ldots, k$ is performed. Using the matrix-vector multiplications, Eq. (10) is also replaced as

$$\boldsymbol{x}_i = \boldsymbol{a}_i - X_{i-1}X_{i-1}^{\top}\boldsymbol{a}_i. \tag{11}$$

In general, to achieve better performance, we reduce the number of data synchronizations on shared-memory multi-core processors as much as possible. The level 2 BLAS operations, such as the matrix-vector multiplications, have less data synchronization than the level 1 BLAS operations. Thus, the level 2 BLAS operations achieves better performance than the level 1 BLAS operations in parallel computing. Given this property, the CGS is conventionally implemented using matrix-vector multiplications.

However, the orthogonality of the vectors computed by the CGS algorithm deteriorates if the condition number of the original vectors is large. To improve the orthogonality, the variants of the CGS algorithm have been proposed.

The CGS algorithm with reorthogonalization (CGS2 algorithm) [3] is one of the variants. A pseudocode of the CGS2 is shown in Algorithm 3. Repeating the CGS algorithm twice, we are able to improve the orthogonality. However, the computational cost of the CGS2 is twice that of the CGS.

3.1.2 MGS algorithm

Another variant of the CGS algorithm is the modified Gram-Schmidt (MGS) algorithm. The MGS algorithm is composed of inner-dot product and AXPY operations. Then level 1 BLAS operations are mainly used. However, compared with the CGS, the MGS improves the orthogonality. Furthermore, the computational cost of the MGS is $2mk^2$ since the MGS is algebraically equivalent to the CGS.

Algorithm 4 OpenMP-based parallel implementation of CGS2 algorithm

```
1: function OMP-CGS2($X_{i-1}(= [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}])$, $\boldsymbol{a}_i$)
2:     #omp parallel private($j$, $s$)
3:     do $j = 1, 2$
4:         #omp single
5:         $\boldsymbol{w} = \boldsymbol{a}_i$                          ▷ Perform serially
6:         #omp end single
7:         #omp do reduction(+:$\boldsymbol{a}_i$)
8:         do $k = 1$ to $i - 1$
9:             $s = -\langle \boldsymbol{x}_k,\ \boldsymbol{w} \rangle$
10:            $\boldsymbol{a}_i = \boldsymbol{a}_i + s\boldsymbol{x}_k$                ▷ Array reduction
11:        end do
12:        #omp end do
13:    end do
14:    #omp end parallel
15:    return $\boldsymbol{x}_i = \boldsymbol{a}_i$
16: end function
```

3.1.3 Compact WY algorithm

The Householder transformations [6] are also used for the reorthogonalization. However, the reorthogonalization using the Householder transformations is composed of the level 1 BLAS operations. Hence, we cannot achieve higher performance using parallel computation.

To overcome this difficulty, a reorthogonalization algorithm using the Householder transformations in terms of the compact WY representation [16] is proposed in [19]. Hereafter, this algorithm is referred to as the cWY algorithm. In this algorithm, we can rewrite the product of the Householder matrices in a simple block matrix form. Hence, the cWY can be performed mainly using the level 2 BLAS operations. This algorithm can achieve the high orthogonality theoretically and high scalability in parallel computing. In addition, the computational cost of the cWY algorithm can be reduced from $4mk^2+k^3$ to $4mk^2-k^3$ [9].

3.2 OpenMP-based parallel implementation of CGS2 algorithm

Recalling Eq. (10), the CGS and CGS2 algorithms can be parallelized in terms of the summation. Such parallel implementation is easily realized by adding OpenMP directives for shared-memory multi-core processors. From these facts, an OpenMP-based parallel implementation of the CGS2 algorithm can be represented as shown in Algorithm 4. Note that where $\boldsymbol{w}$ is a vector where preserves the original vector of $\boldsymbol{a}_i$. Hereafter, this implementation of the CGS2 algorithm is referred to as the OMP-CGS2 algorithm.

The parallel computation in terms of the summation is represented as the parallelism of do-loop as shown in line 7.

Table 1: Comparison of reorthogonalization algorithms [4]

	CGS2	MGS	cWY	OMP-CGS2
Computation	$4mk^2$	$2mk^2$	$4mk^2 - k^3$	$4mk^2$
Orthogonality	$O(\epsilon)^\dagger$	$O(\epsilon\kappa(A))$	$O(\epsilon)$	$O(\epsilon)^\dagger$
BLAS	Level 2	Level 1	Level 2	Level 1

†: Realized if the condition $O(\epsilon\kappa(A)) < 1$ is satisfied.

As the result, the inner-dot product (line 9) and the AXPY operations (line 10) in terms of the different index k are performed on each thread. In addition, the array reduction must be implemented for the summation of $\boldsymbol{a}_i$ on line 10. The array reduction in Fortran code is supported by using the `reduction` clause of OpenMP.

The advantage of this implementation is the high reusability of data. Since we compute $\boldsymbol{a}_i = \boldsymbol{a}_i + s\boldsymbol{x}_k$ (line 10) as soon as $s = -\langle \boldsymbol{x}_k,\ \boldsymbol{w}\rangle$ (line 9) is computed, the reusability of $\boldsymbol{w}$, $\boldsymbol{x}_k$, and $\boldsymbol{a}_i$ becomes higher on each thread computation. Thus, the OMP-CGS2 algorithm is expected to accelerate more effectively the reorthogonalization computation on shared-memory multi-core processors with large caches than other reorthogonalization algorithms if the vectors $\boldsymbol{w}$, $\boldsymbol{x}_k$, and $\boldsymbol{a}_i$ are stored in the L3 cache of each CPU.

3.3 Comparison of reorthogonalization algorithms

As the summary of this section, Table 1 shows that the theoretical performance of the reorthogonalization algorithms. *Computation* denotes the flops of the computational cost, *Orthogonality* indicates the bound of the norm $\|X^\top X - I\|$, and *BLAS* denotes the level of BLAS operations of which each algorithm is mainly composed. ϵ is the machine epsilon, and $\kappa(A)$ denotes the condition number of the original matrix $A = \begin{bmatrix}\boldsymbol{a}_1 & \cdots & \boldsymbol{a}_k\end{bmatrix}$.

4. Numerical experiments

In this section, we report results of numerical experiments in order to evaluate the performance of the OpenMP-based parallel implementation of the CGS2 algorithm.

4.1 Configurations of numerical experiments

In the numerical experiments, we compare the elapsed time for computing the l largest singular triplets of the same target matrix using a code of the GKLR algorithm with different l. Here, l is the number of required singular triplets; $l = 100,\ 200,\ 400,\ 800$.

We compare the elapsed time for computing subsets of singular triplets using four different codes of the GKLR algorithms. Each GKLR code is implemented with the following reorthogonalization algorithms mentioned in Section 3. **GKLR with MGS** is implemented with the MGS algorithm. **GKLR with CGS2** is implemented with the CGS2 algorithm. **GKLR with cWY** is implemented with the cWY algorithm. The reorthogonalization algorithms of the above three code are parallelized in terms of the BLAS routines. **GKLR with OMP-CGS2** is implemented with the OpenMP-based parallel implementation of the CGS2 algorithm.

Table 2: Specifications of the experimental environment

	1 node of Appro 2548X at ACCMS, Kyoto University
CPU	Intel Xeon E5-4650L@2.6 GHz, 32 cores (8 cores × 4)
	L3 cache: 20MB × 4
RAM	DDR3-1066 1.5 TB, 136.4GB/sec
Compiler	Intel C++/Fortran Compiler 14.0.2
Options	`-O3 -xHOST -ipo -no-prec-div`
	`-openmp -mcmodel=medium -shared-intel`
Software	Intel Math Kernel Library 11.1.2

In the experiments, we use three $m \times n$ real sparse matrices T_1, T_2, and T_3. All of T_1, T_2, and T_3 are set to be 256 nonzero elements, which are set to be random numbers in the range $(0, 1)$ and are randomly allocated, in each row. T_1, T_2, and T_3 are only different in the size of m and n from each other as follows: $m = 16,000$ and $n = 8,000$ for T_1. $m = 32,000$ and $n = 16,000$ for T_2. $m = 64,000$ and $n = 32,000$ for T_3. In addition, the condition number is 4.803×10^1 for T_1, 4.754×10^1 for T_2, and 4.757×10^1 for T_3, respectively.

Finally, all the experiments are run with 32 threads on a machine shown in Table 2. We use the Intel Math Kernel Library (MKL) [7] for parallelizing the level 2 and level 3 BLAS routines. The Intel MKL also provides the level 1 BLAS routines, but the implementation depends on the dimension of the target vectors and the performance of them is unstable. Thus, we use the hand-made level 1 BLAS routines, which is parallelized by using OpenMP, in the experiments.

4.2 Results of performance evaluation

Figs. 1, 2, and 3 graph the experimental results and shows the number of required singular triplets and the elapsed time for computing singular triplets of each target matrix T_1, T_2, or T_3 using the four code of the GKLR algorithm, respectively. From the figures, **GKLR with OMP-CGS2** is faster than the other code in all the cases. Thus, the OMP-CGS2 accelerates the computation of the GKLR algorithm more effectively than the other reorthogonalization algorithms.

In addition, Tables 3, 4, and 5 show the number of required singular triplets and the elapsed time spending for the reorthogonalization process in computing the singular triplets of each target matrix T_1, T_2, and T_3 using the four code of the GKLR algorithm, respectively. The tables show that the OMP-CGS2, the reorthogonalization in **GKLR with OMP-CGS2**, is at least twice faster than the CGS2 and cWY algorithms.

Note that the number of iterations at the point (k_{end}), where the GKLR algorithm stops, is the same regardless to the reorthogonalization algorithms in each of the ex-

Table 3: The number of required singular triplets (l) and the elapsed time (sec.) spending for the reorthogonalization process in computing the singular triplets of T_1 using each code of the GKLR algorithms.

# of required singular triplets	100	200	400	800
GKLR with MGS	79	176	337	1,121
GKLR with CGS2	21	57	118	315
GKLR with cWY	24	58	132	302
GKLR with OMP-CGS2	7	20	44	102

Table 4: The number of required singular triplets (l) and the elapsed time (sec.) spending for the reorthogonalization process in computing the singular triplets of T_2 using each code of the GKLR algorithms.

# of required singular triplets	100	200	400	800
GKLR with MGS	100	293	750	1,664
GKLR with CGS2	70	154	351	774
GKLR with cWY	71	166	360	796
GKLR with OMP-CGS2	25	59	151	310

Table 5: The number of required singular triplets (l) and the elapsed time (sec.) spending for the reorthogonalization process in computing the singular triplets of T_3 using each code of the GKLR algorithms.

# of required singular triplets	100	200	400	800
GKLR with MGS	261	533	1,432	2,815
GKLR with CGS2	169	372	861	1,921
GKLR with cWY	182	393	861	2,095
GKLR with OMP-CGS2	83	183	344	844

Table 6: The number of iterations at the point (k_{end}), where the GKLR algorithm stops, needed in each of the experiments. l denotes the number of required singular triplets.

l	100	200	400	800
Matrix T_1	1,000	1,600	2,400	4,000
Matrix T_2	1,300	2,000	3,200	4,800
Matrix T_3	1,600	2,400	3,600	5,600

periments. Table 6 summarizes k_{end} needed in each of the experiments.

4.3 Discussion about cache use in OMP-CGS2

As mentioned in Sec. 3.2, the high performance of OMP-CGS2 arises from the higher reusability of cache in CPU. Here, we discuss the limit size of the vectors when we perform the reorthogonalization by using OMP-CGS2. Let the number of threads in a CPU be T and the capacity of L3 cache in the CPU be C MB. The one data of the elements needs 8 bytes when we use a double-precision floating-point number.

Recalling Algorithm 4, the vectors $\boldsymbol{w}$, $\boldsymbol{a}_i$, and $\boldsymbol{x}_k$ appear at each of **do**-loop in terms of k. If all these vectors are stored in the L3 cache of CPU, we can achieve the higher performance of the reorthogonalization by using OMP-CGS2. However, $\boldsymbol{x}_k$ is not shared by different threads while $\boldsymbol{w}$ is accessed by all computing threads. In addition, each thread should access the copy of $\boldsymbol{a}_i$ before reducing arrays.

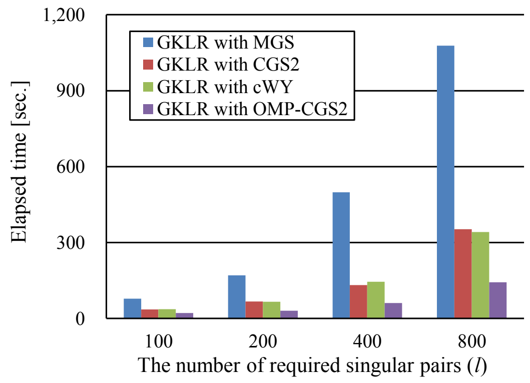

Fig. 1: The number of required singular triplets and the elapsed time for computing the l largest singular triplets of T_1 using the GKLR algorithm with different reorthogonalization implementation.

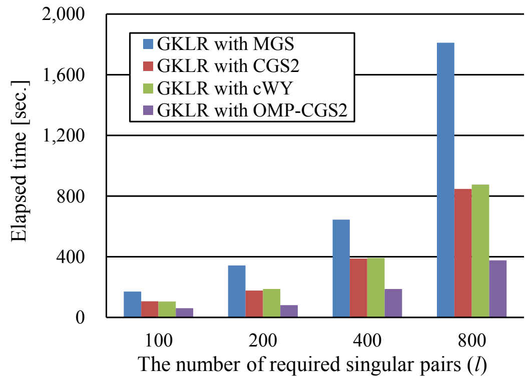

Fig. 2: The number of required singular triplets and the elapsed time for computing the l largest singular triplets of T_2 using the GKLR algorithm with different reorthogonalization implementation.

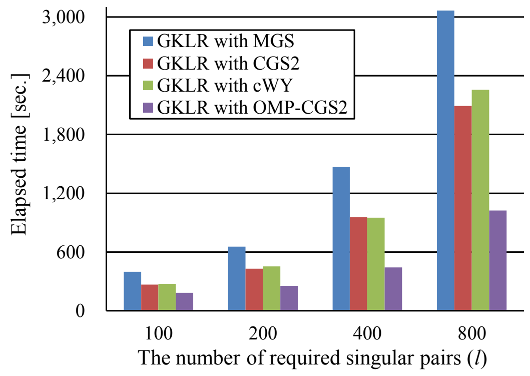

Fig. 3: The number of required singular triplets and the elapsed time for computing the l largest singular triplets of T_3 using the GKLR algorithm with different reorthogonalization implementation.

As the results, the number of the vectors which should be stored in the cache is $(T \times 2 + 1)$.

From the above discussion, the dimension of the matrix which achieves better performance in this environment is determined by following inequality:

$$m \times (T \times 2 + 1) \times 8 \leq C \times 1024 \times 1024, \qquad (12)$$

where m is the size of the vectors $\boldsymbol{w}$, $\boldsymbol{a}_i$, and $\boldsymbol{x}_k$. Then, since $T = 8$ and $C = 20$ from the specification of the CPUs used for the performance evaluation in this paper, the following inequality holds:

$$m \leq 154202. \qquad (13)$$

Thus, under the condition (13) of the performance evaluation in the experimental environment in Table 2, the OMP-CGS2 algorithm is guaranteed to achieve the higher performance than the other reorthogonalization algorithms.

5. Conclusions and future work

In this paper, we first introduce the GKLR algorithm for computing a subset of singular triplets for target matrices. To accelerate the reorthogonalization of the GKLR algorithm on shared-memory multi-core processors more effectively, we then present the OpenMP-based parallel implementation of the CGS2 algorithm. The OpenMP-based implementation of the CGS2 algorithm has the advantage of the data reusability.

We performed numerical experiments on shared-memory multi-core processors to evaluate the performance of the GKLR algorithm with the different parallel implementations of the reorthogonalization algorithm including the OpenMP-based implementation of the CGS2 algorithm. Experimental results show that the OpenMP-based implementation of the CGS2 algorithm accelerates the GKLR algorithm more effectively for computing a subset of singular triplets for a sparse matrix than other reorthogonalization algorithms.

One of future work, is to evaluate the performance of the GKLR algorithms for larger target matrices than those we used in the performance evaluation and to extend and confirm the validity of the modeling inequality (12) depending on CPUs. The other is to apply the OpenMP-based parallel implementation of the CGS2 algorithm presented in this paper to other algorithms, such as the inverse iteration method, GMRES algorithm [15], and implicitly restarted Arnoldi and Lanczos methods [18], [2] to accelerate their reorthogonalization processes.

Acknowledgment

The authors would like to express their gratitude to reviewers of this paper for their helpful comments. In this work, we used the supercomputer of ACCMS, Kyoto University. This work was supported by JSPS KAKENHI Grant Numbers 13J02820 and 24360038.

References

[1] J. L. Barlow, "Reorthogonalization for the Golub-Kahan-Lanczos bidiagonal reduction," *Numer. Math.*, pp. 1–42, 2013.
[2] D. Calvetti, L. Reichel, and D. C. Sorensen, "An implicitly restarted lanczos method for large symmetric eigenvalue problems," *ETNA*, vol. 2, pp. 1–21, 1994.
[3] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, "Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization," *Math. Comput.*, vol. 30, no. 136, pp. 772–795, 1976.
[4] L. Giraud, J. Langou, M. Rozloẑnìk, and J. van den Eshof, "Rounding error analysis of the classical Gram-Schmidt orthogonalization process," *Numer. Math.*, vol. 101, no. 1, pp. 87–100, 2005.
[5] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *SIAM J. Numer. Anal.*, vol. 2, no. 2, pp. 205–224, 1965.
[6] G. H. Golub and C. F. van Loan, *Matrix Computations*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.
[7] Intel Math Kernel Library, "Available electronically at https://software.intel.com/en-us/intel-mkl/," 2003.
[8] I. C. F. Ipsen, "Computing an eigenvector with inverse iteration," *SIAM Review*, vol. 39, no. 2, pp. 254–291, 1997.
[9] H. Ishigami, K. Kimura, and Y. Nakamura, "On implementation and evaluation of inverse iteration algorithm with compact WY orthogonalization," *IPSJ Transactions on Mathematical Modeling and Its Applications*, vol. 6, no. 2, pp. 25–35, 2013.
[10] W. Kahan, "Accurate eigenvalues of a symmetric tridiagonal matrix," *Technical Report, Computer Science Dept. Stanford University*, no. CS41, 1966.
[11] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users's Guide*. Philadelphia, PA, USA: SIAM, 1998.
[12] Netlib, "BLAS," accssesed 2015-01-16. [Online]. Available: http://www.netlib.org/blas/
[13] OpenMP, "Available electronically at http://openmp.org/wp/," 1997.
[14] Y. Saad, "On the rates of convergence of the Lanczos and the block-Lanczos methods," *SIAM J. Numer. Anal.*, vol. 17, no. 5, pp. 687–706, 1980.
[15] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, 1986.
[16] R. Schreiber and C. van Loan, "A storage-efficient WY representation for products of Householder transformations," *SIAM J. Sci. Stat. Comput.*, vol. 10, no. 1, pp. 53–57, 1989.
[17] H. D. Simon and H. Zha, "Low-rank matrix approximation using the Lanczos bidiagonalization process with applications," *SIAM J Sci. Comput.*, vol. 21, no. 6, pp. 2257–2274, 2000.
[18] D. C. Sorensen, "Implicit application of polynomial filters in a k-step Arnoldi method," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 357–385, Jan. 1992.
[19] Y. Yamamoto and Y. Hirota, "A parallel algorithm for incremental orthogonalization based on the compact WY representation," *JSIAM Letters*, vol. 3, pp. 89–92, 2011.