IPSJ Transactions on Mathematical Modeling and Its Applications Vol.11 No.1 37–44 (Mar. 2018)

Regular Paper

Implementation of Computing Partial Singular Value Decomposition for Principal Component Analysis Using ARPACK

Masami Takata[1,a)] Sho Araki[2,b)] Kinji Kimura[2,c)]
Yuki Fujii[2,d)] Yoshimasa Nakamura[2,e)]

Received: July 13, 2017, Revised: October 16, 2017,
Accepted: November 18, 2017

Abstract: In this paper, we propose a new implementation for computing partial SVD (singular value decomposition) for principal component analysis, which is introduced from the viewpoint of the computational order and the caches of shared-memory multi-core processors. In the new implementation, a target matrix is a sparse matrix, which should be stored in CRS (compressed row storage) or CCS (compressed column storage) formats. SVD can be transformed into eigenvalue problem. 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). To get the SVDs using ARPACK, the transformed eigenvalue problems can be generally solved 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 is evaluated by experimentation. The experimental results show that the computation time of the proposed implementation is about 75% of that of the conventional implementation.

Keywords: Krylov subspace method, IRA algorithm, IRL algorithm, sparse matrix, matrix-vector multiplication

1. Introduction

In markets, the data of customers are obtained and stored in a matrix. The data matrix is a sparse matrix. The rows of the data matrix is customer's number, and the columns express the data of purchase and condition. In customers research, the feature of the data matrix is important. To obtain the feature of the data matrix, principal component analysis is generally adopted. In principal component analysis, the normalized data matrix is generated using the data matrix. The normalized data matrix is a dense matrix. In customers research, some given data matrices are usually rectangular, and those size become larger and larger. Therefore, since the size of the data matrices tends to exceed the size of cache, computation times become larger.

In SVD (singular value decomposition) of the normalized data matrix for principal component analysis, we need only larger singular values and the corresponding singular vectors, which are called as singular pairs in our research. 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 the eigenvalues are non-negative numbers.

In the case where only the partial singular pairs corresponding to the several singular values from the maximal or the minimal one, the singular pairs are computed by the Golub-Kahan-Lanczos (GKL) algorithm [6], the Jacobi-Davidson algorithm [11], the randomized algorithm [8], and the augmented implicitly restarted Lanczos bidiagonalization (AIRLB) algorithm [2], [3]. The GKL algorithm is a classical algorithm. The Jacobi-Davidson algorithm is suitable for the maximal singular value and its singular vectors. The randomized algorithm is suitable for a partial SVD whose singular values are not clustered. The AIRLB algorithm is appropriate for use as a computation library since it has low dependency on input matrices and can output solutions numerically stably.

In the case where only the partial eigenpairs, which are combined with an eigenvalue and the corresponding eigenvector, from the absolute maximum or the absolute minimum eigenvalue of the target matrix are needed, these eigenpairs are computed by the IRA (implicitly restarted Arnoldi) algorithm [12] and the IRL (implicitly restarted Lanczos) algorithm [13] using the ARPACK (ARnoldi PACKage) [9] 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

[1] Nara Women's University, Nara 630–8506, Japan
[2] Kyoto University, Kyoto 606–8501, Japan
[a)] takata@ics.nara-wu.ac.jp
[b)] araki@amp.i.kyoto-u.ac.jp
[c)] kimura.kinji.7z@kyoto-u.ac.jp
[d)] fujii@amp.i.kyoto-u.ac.jp
[e)] ynaka@i.kyoto-u.ac.jp

Krylov subspace.

The ARPACK software is one of the effective software packages. Since a lot of researchers and developers have used the ARPACK software over the years, this software has improved reliability through bugs have been removed. On the other hand, the above algorithms for computing the partial singular pairs are still a work in progress. Hence, to establish an high reliable algorithm to compute the partial singular pairs or eigenpairs, adoption of the ARPACK software is suitable.

Since ARPACK adopts reverse communication interface [9], users simply compute matrix-vector operations. In general, a partial eigenvalue problem can be solved 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) [9] 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 the proposed implementation, it is required that target matrices are sparse matrices, which are stored in CRS (compressed row storage) or CCS (compressed column storage) formats [5]. On the other hand, actually the normalized data matrix in principal component analysis is a dense matrix. Thus, the proposed implementation should not be employed directly. To employ the IRL algorithm in ARPACK software, the SVD of a target matrix is transformed into an eigenvalue problem through multiplication of the target matrix by the transpose matrix. Therefore, once the normalized data matrix, which is a dense matrix, is expanded using the given data matrix, which is a sparse matrix in customers research, then the proposed implementation can be adopted. In general, the computational order in a sparse matrix is smaller than that in a dense matrix. Consequently, the proposed implementation using ARPACK is suitable to perform principal component analysis for customers research.

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:

$$\mathcal{K}(A, \mathbf{q}_1, k) = \mathrm{span}\{\mathbf{q}_1, A\mathbf{q}_1, \ldots, A^{k-1}\mathbf{q}_1\}. \tag{1}$$

Algorithm 1 Arnoldi algorithm

```
1: Set initial vector q_1;
2: v_1 := q_1/|q_1|;
3: for j := 1 to k do
4:     r_j := A v_j;
5:     for i := 1 to j do
6:         h_ij := v_i^T r_j;
7:         r_j := r_j − h_ij v_i;
8:     end for
9:     h_{j+1,j} := |r_j|;  v_{j+1} := 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:

$$\mathcal{K}(A, \mathbf{q}_1, k) = \mathcal{K}(A, \mathbf{v}_1, k) = \mathrm{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}, \tag{2}$$

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 [7].

After the k-th iteration for $k = 2, 3, \ldots$ in Algorithm 1, the following equation holds:

$$AV_k = V_k H_k + \mathbf{r}_k \mathbf{e}_k^\top, \tag{3}$$

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

$$H_k := \begin{bmatrix} h_{11} & h_{12} & \cdots & \cdots & h_{1k} \\ h_{21} & h_{22} & \cdots & \cdots & h_{2k} \\ 0 & h_{32} & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{k,k-1} & h_{kk} \end{bmatrix}. \tag{4}$$

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,

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: $\quad \mathbf{r}_j := A\mathbf{v}_j$;
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**

the following equation is satisfied:

$$H_k \mathbf{y}_j^{(k)} = \lambda_j^{(k)} \mathbf{y}_j^{(k)}. \tag{5}$$

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

$$(A + E)\mathbf{x}_j^{(k)} = \lambda_j^{(k)}\mathbf{x}_j^{(k)}. \tag{12}$$

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.

2.2 Lanczos Algorithm

As well as the Arnoldi algorithm, the Lanczos algorithm [10] 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,

$$\begin{aligned} h_{ij} &= \mathbf{v}_i^\top A \mathbf{v}_j = \mathbf{v}_i^\top A^\top \mathbf{v}_j \\ &= (A\mathbf{v}_i)^\top \mathbf{v}_j = \left(\mathbf{v}_j^\top (A\mathbf{v}_i)\right)^\top = h_{ji} \end{aligned} \tag{13}$$

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:

$$AV_k = V_k T_k + \mathbf{r}_k \mathbf{e}_k^\top, \tag{14}$$

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

$$T_k := \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ddots & \vdots \\ 0 & \beta_2 & \alpha_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{k-1} \\ 0 & \cdots & 0 & \beta_{k-1} & \alpha_k \end{bmatrix}. \tag{15}$$

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 [12], [13], 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 [9].

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:

$$\tilde{H}_m^{(i)} = \tilde{Q}_i \tilde{R}_i \tag{16}$$

$$\tilde{H}_m^{(i+1)} = \tilde{R}_i \tilde{Q}_i. \tag{17}$$

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:

$$\begin{aligned} \tilde{H}_m^{(i)} &= \tilde{Q}_{i-1}^\top \cdots \tilde{Q}_2^\top \tilde{Q}_1^\top H_m^{(1)} \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_{i-1} && (18)\\ &= \tilde{Q}^\top \tilde{H}_m^{(1)} \tilde{Q} \quad (\tilde{Q} := \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_{i-1}). && (19) \end{aligned}$$

On the other hand, in the implicitly shifted QR steps, the shift values $\mu_i \in \mathbb{R}$ are introduced:

$$\tilde{\tilde{H}}_m^{(i)} - \mu_i I = \tilde{\tilde{Q}}_i \tilde{\tilde{R}}_i \tag{20}$$

$$\tilde{\tilde{H}}_m^{(i+1)} = \tilde{\tilde{R}}_i \tilde{\tilde{Q}}_i + \mu_i I. \tag{21}$$

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:

$$\begin{aligned} \tilde{\tilde{H}}_m^{(i)} &= \tilde{\tilde{Q}}_{i-1}^\top \cdots \tilde{\tilde{Q}}_2^\top \tilde{\tilde{Q}}_1^\top H_m^{(1)} \tilde{\tilde{Q}}_1 \tilde{\tilde{Q}}_2 \cdots \tilde{\tilde{Q}}_{i-1} && (22)\\ &= \tilde{Q}^\top H_m^{(1)} \tilde{Q} \quad (\tilde{Q} := \tilde{\tilde{Q}}_1 \tilde{\tilde{Q}}_2 \cdots \tilde{\tilde{Q}}_{i-1}) && (23) \end{aligned}$$

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 $\{\tilde{\tilde{R}}_1\}_{m,m} = 0$:

$$\tilde{\tilde{H}}_m^{(1)} = \begin{bmatrix} * & * & \cdots & \cdots & * \\ * & * & \cdots & \cdots & * \\ 0 & * & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * & * \end{bmatrix}, \tag{24}$$

$$\tilde{\tilde{R}}_1 = \begin{bmatrix} * & * & \cdots & \cdots & * \\ 0 & * & \cdots & \cdots & * \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}. \tag{25}$$

Thus, $\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1$ becomes an upper Hessenberg matrix with $\{\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1\}_{m,m-1} = 0$ and $\{\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1\}_{m,m} = 0$:

$$\tilde{\tilde{R}}_1\tilde{\tilde{Q}}_1 = \begin{bmatrix} * & * & \cdots & \cdots & \cdots & * \\ * & * & \cdots & \cdots & \cdots & * \\ 0 & * & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * & * \\ 0 & \cdots & \cdots & 0 & 0 & 0 \end{bmatrix}. \tag{26}$$

Then, we obtain

$$\tilde{\tilde{H}}_m^{(2)} = \begin{bmatrix} * & * & \cdots & \cdots & \cdots & * \\ * & * & \cdots & \cdots & \cdots & * \\ 0 & * & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * & * \\ 0 & \cdots & \cdots & 0 & 0 & \mu_1 \end{bmatrix}. \tag{27}$$

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 $\mathbf{v}_{\ell+1}^+ \in \mathbb{R}^n$ 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 $\mathbf{v}_{\ell+1}^+$.

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;

$$Q^+ := Q_1 Q_2 \cdots Q_{m-\ell}, \tag{30}$$

$$V_m^+ := V_m Q^+, \tag{31}$$

$$H_m^+ := (Q^+)^\top H_m Q^+, \tag{32}$$

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: $\mathbf{v}_{\ell+1}^+ := \mathbf{v}_{m+1}$; $h_{\ell+1,\ell}^+ = h_{m+1,m} Q^+(m, \ell)$;
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**

$$H_m^+ = \begin{bmatrix} * & \cdots & & & & & \cdots & * \\ * & & & & & & & \vdots \\ 0 & \ddots & & & & & & \\ \vdots & \ddots & \ddots & \ddots & & & & \\ & & & * & * & & & \\ & & & & 0 & \mu_{m-\ell} & & \\ & & & & & \ddots & \ddots & \vdots \\ \vdots & & & & & \ddots & \ddots & \mu_2 & * \\ 0 & \cdots & & & & \cdots & 0 & 0 & \mu_1 \end{bmatrix}. \tag{33}$$

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:

$$AV_m^+(:, 1:\ell) = V_m^+ H_m^+(:, 1:\ell) + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+(:, 1:\ell) \tag{38}$$

$$= V_m^+(:, 1:\ell) H_m^+(1:\ell, 1:\ell) + h_{\ell+1,\ell}^+ \mathbf{v}_{\ell+1}^+ \mathbf{e}_\ell^\top \tag{39}$$

where $\mathbf{v}_{\ell+1}^+ := \mathbf{v}_{m+1}$ and $h_{\ell+1,\ell}^+ = h_{m+1,m} Q^+(m, \ell)$. Thus, we are able to restart the Arnoldi decomposition with the initial vector $\mathbf{v}_{\ell+1}^+$ and Eq. (39).

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.

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 $\sigma_j^{(r)} \geq 0$ ($j : 1 \leq i \leq n$) $\in \mathbb{R}$ 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:

$$\begin{aligned} A^{(r)\top}A^{(r)}\mathbf{v}_j^{(r)} &= A^{(r)\top}\left(\sigma_j^{(r)}\mathbf{u}_j^{(r)}\right) \\ &= \sigma_j^{(r)}\left(A^{(r)\top}\mathbf{u}_j^{(r)}\right) = \sigma_j^{(r)2}\mathbf{v}_j^{(r)}, \end{aligned} \tag{40}$$

$$\mathbf{u}_j^{(r)} = \frac{A^{(r)}\mathbf{v}_j^{(r)}}{\|A^{(r)}\mathbf{v}_j^{(r)}\|_2}. \tag{41}$$

Equation (40) 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. (41).

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:

$$\begin{aligned} A^{(c)}A^{(c)\top}\mathbf{u}_j^{(c)} &= A^{(c)}\left(\sigma_j^{(c)}\mathbf{v}_j^{(c)}\right) \\ &= \left(A^{(c)}\mathbf{v}_j^{(c)}\right)\sigma_j^{(c)} = \mathbf{u}_j^{(c)}\sigma_j^{(c)2}, \end{aligned} \tag{42}$$

$$\mathbf{v}_j^{(c)} = \frac{A^{(c)\top}\mathbf{u}_j^{(c)}}{\|A^{(c)\top}\mathbf{u}_j^{(c)}\|_2}. \tag{43}$$

Equation (42) 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^T 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^T, x⟩ (a_i = A^(r)(i, :));
5:     r = r + t a_i^T;
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 $A^{(r)}(i, :)$ ($i : 1 \leq i \leq n$) $\in \mathbb{R}^n$ 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 8 bytes, the size of caches needs, theoretically, to be more than $n \times 8$ bytes.

In the case that $A^{(r)}$ can be stored in caches or $A^{(r)}(i, :)$ can not be stored in caches, the computational times of the conventional implementation are as well as those of the proposed implementation.

4.3 Principal Component Analysis

The proposed implementation can be made efficient for use in the case of sparse matrices , which should be stored in CRS and CCS formats . However, in principal component analysis, the normalized data matrix is a dense matrix. Thus, the proposed implementation should not be employed directly. Therefore, equations in principal component analysis is expanded.

The normalized data matrix $B \in \mathbb{R}^{w\times n}$ is generated using the data matrix $C \in \mathbb{R}^{w\times n}$.

$$B = (C - M)\,S, \tag{44}$$

$$M := \begin{bmatrix} \frac{1}{w}\left(\sum_{i=1}^{w} C_{i,1}\right) & \cdots & \frac{1}{w}\left(\sum_{i=1}^{w} C_{i,n}\right) \\ \frac{1}{w}\left(\sum_{i=1}^{w} C_{i,1}\right) & \cdots & \frac{1}{w}\left(\sum_{i=1}^{w} C_{i,n}\right) \\ \vdots & \vdots & \vdots \\ \frac{1}{w}\left(\sum_{i=1}^{w} C_{i,1}\right) & \cdots & \frac{1}{w}\left(\sum_{i=1}^{w} C_{i,n}\right) \end{bmatrix}, \tag{45}$$

$$S := \begin{bmatrix} \frac{1}{\sqrt{\sum_{i=1}^{w}\left((C-M)_{i,1}\right)^2}} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\sqrt{\sum_{i=1}^{w}\left((C-M)_{i,n}\right)^2}} \end{bmatrix}. \tag{46}$$

[*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.

In principal component analysis for customers research, obtaining the feature of the normalized data matrix B is important. Feature quiantity of the matrix B is obtained by using principal component analysis. To perform principal component analysis, the IRL algorithm is employed. Thus, $B^\top B\mathbf{x}$ can be computed. $B^\top B\mathbf{x}$ is expanded as follows:

$$\begin{aligned} B^\top B\mathbf{x} &= S^\top \left(C^\top - M^\top\right)(C - M)\, S\mathbf{x} \\ &= S^\top \left(C^\top - M^\top\right)(C\,(S\mathbf{x}) - M\,(S\mathbf{x})) \\ &= S^\top \Big(C^\top (C\,(S\mathbf{x}) - M\,(S\mathbf{x})) \\ &\quad - M^\top (C\,(S\mathbf{x}) - M\,(S\mathbf{x}))\Big) \end{aligned} \tag{47}$$

Since the matrix S is a diagonal matrix, the computational order of $S\mathbf{x}$ is $O(n)$. All column in the matrix M consists of the same elements. Thus, once a row in the matrix M is multiplied by $S\mathbf{x}$, all elements can be computed since all elements in the vector $MS\mathbf{x}$ is the same. Consequently, the computational order of $MS\mathbf{x}$ is $O(n)$. By using Eq. (47), once $S\mathbf{x}$ and $MS\mathbf{x}$ are computed, $C^\top (C\,(S\mathbf{x}) - M\,(S\mathbf{x}))$ can be easily computed using the proposed implementation. In the matrix $M^\top$, all elements of a row is the same value. Hence, the computational order of $M^\top (C\,(S\mathbf{x}) - M\,(S\mathbf{x}))$ is $O(n)$. As the remark, $C\,(S\mathbf{x})$ is computed through the computation of $C^\top (C\,(S\mathbf{x}) - M\,(S\mathbf{x}))$ can be easily computed using the proposed implementation. The pseudocode is shown in Algorithm 6 and 7. Computational order of Eq. (47) is as same as that of $B^\top B\mathbf{x}$. However, the data in the matrix C can be reused in the proposed implementation. Moreover, since data matrix C is a sparse matrix, the principal component analylsis can be computed using the sparse matrix.

Algorithm 6 Set up code for principal component analysis

1: $\overline{\mathbf{M}}_{(j)} = \frac{1}{w}\left(\sum_{i=1}^{w} C_{i,j}\right)$;
2: $\overline{\mathbf{S}}_{(j)} = \frac{1}{\sqrt{\sum_{i=1}^{w}\left((C-M)_{i,j}\right)^2}}$;

Algorithm 7 Computation of $B^\top B\mathbf{x}$

1: **for** $i = 1$ to n **do**
2: $\quad \overline{Sx}_{(i)} = \overline{\mathbf{S}}_{(i)}\mathbf{x}_{(i)}$;
3: **end for**
4: $\overline{MSx} = 0$;
5: **for** $i = 1$ to n **do**
6: $\quad \overline{MSx} = \overline{MSx} + \overline{\mathbf{M}}_{(i)}\overline{Sx}_{(i)}$;
7: **end for**
8: $\mathbf{r} = 0$;
9: $t_2 = 0$;
10: #omp parallel for private(t_1) reduction(+:$\mathbf{r}$) reduction(+:t_2)
11: **for** $i = 1$ to n **do**
12: $\quad t_1 = \langle \mathbf{c}_i^\top, \overline{Sx}\rangle$ ($\mathbf{c}_i = C(i,:)$);
13: $\quad t_1 = t_1 - \overline{MSx}$;
14: $\quad \mathbf{r} = \mathbf{r} + t_1\mathbf{c}_i^\top$;
15: $\quad t_2 = t_2 + t_1$;
16: **end for**
17: #omp end parallel for
18: **for** $i = 1$ to n **do**
19: $\quad \mathbf{r}_{(i)} = \mathbf{r}_{(i)} - t_2\overline{\mathbf{M}}_{(i)}$;
20: **end for**
21: **for** $i = 1$ to n **do**
22: $\quad \mathbf{r}_{(i)} = \mathbf{r}_{(i)}\overline{\mathbf{S}}_{(i)}$;
23: **end for**

4.4 Sparse Matrix

In the case of a $w \times n$ ($w \geq n$ and $w < n$) rectangular matrix $A^{(r)}$, these elements should be stored in CRS and CCS formats, respectively. The CRS and CCS formats are the most general [4]. 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 $\mathbf{val}_R$, **col_ind** and **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. **col_ind** is a vector of integers indicates the column indices of the elements in $\mathbf{val}_R$. **row_ptr** is also a vector of integers indicates the locations in $\mathbf{val}_R$ that start a row. The last entry of **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 $\mathbf{val}_R$, **col_ind** and **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. **row_ind**, a vector of integers, indicates the row indices of the elements in $\mathbf{val}_R$. **col_ptr**, a vector of integers, indicates the locations in $\mathbf{val}_R$ that start a column. The last entry of **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

$$A^{(S')} = \begin{bmatrix} 1 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 0 & -1 & 0 & 3 \\ 2 & 7 & 3 & 2 & 6 & 0 \\ 0 & 3 & 8 & 4 & 0 & 0 \\ 3 & 5 & 0 & 9 & 5 & 9 \\ 0 & 0 & 0 & 0 & 2 & 6 \end{bmatrix}. \tag{48}$$

The CRS format for $A^{(S')}$ is

$$\mathbf{val}_R = \{1, 3, 2, -1, 3, 2, \cdots, 5, 9, 2, 6\}, \tag{49}$$

$$\mathbf{col\_ind} = \{1, 5, 2, 4, 6, 1, \cdots, 5, 6, 5, 6\}, \tag{50}$$

$$\mathbf{row\_ptr} = \{1, 3, 6, 11, 14, 19, 20\}. \tag{51}$$

The CCS format for $A^{(S')}$ is

$$\mathbf{val}_C = \{1, 2, 3, 2, 7, 3, \cdots, 2, 3, 9, 6\}, \tag{52}$$

$$\mathbf{row\_ind} = \{1, 3, 5, 2, 3, 4, \cdots, 6, 2, 5, 6\}, \tag{53}$$

$$\mathbf{row\_ptr} = \{1, 4, 8, 10, 14, 18, 20\}. \tag{54}$$

5. Experiment

We have performed experiments to evaluate the performance of the proposed implementation. The dimension of the sample sparse matrix , which is stored in CRS format, is 200,000 × 100,000 and this matrix is composed of the 1,000 non-zero elements in every row. The number of required singular pairs ℓ are

Table 1 Specifications of the experimental environment.

	Environment (CRAY CS400 2820XT), Kyoto University
CPU	Intel Xeon E5-2695 v4 @2.1 GHz, 36 cores (18 cores × 2) L3 cache: 45 MB
RAM	128 GB
Compiler	Intel C++/Fortran Compiler 16.0.4
Options	-qopenmp -O3 -fg-model precise -ipo -xCORE-AVX2
Software	Intel Math Kernel Library 11.3.4

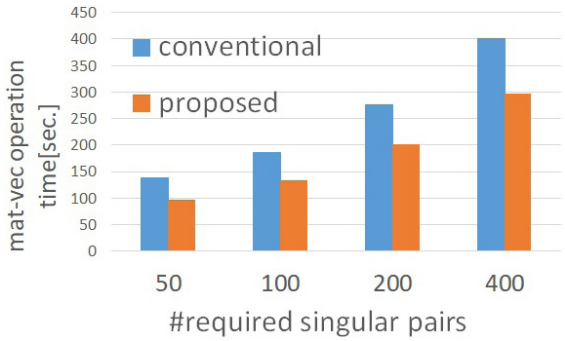

Fig. 1 Computation time of $C^\top C\mathbf{x}$ (C is a 200,000 × 100,000 real matrix), comparison the conventional and the proposed.

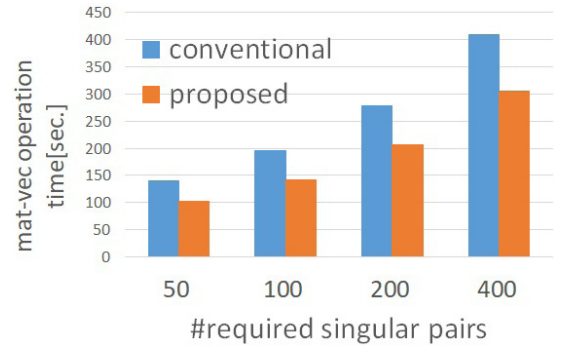

Fig. 2 Computation time of $B^\top B\mathbf{x}$ (C is a 200,000 × 100,000 real matrix), comparison the conventional and the proposed.

50, 100, 200 and 400 from the maximum singular value of C. 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 C is 64 bits in the case of double precision. Therefore, the size of a row in the matrix C is much smaller than the size of the L3 cache. Consequently, all data in $C(i,:)$ are stored to the caches at the same time. Hence, the iteration of the proposed implementation can be performed efficiently because the data of $C(i,:)$ and $C^\top(:,i)$ are the same and have been stored in the caches.

Figure 1 shows the results of the experiments in $C^\top C$. The number of required singular pairs is listed on the horizontal axis, and the vertical axis indicates the computation time for $C^\top C\mathbf{x}$. The results show that the computation time for $C^\top C\mathbf{x}$ of the proposed implementation is about 75% of that of the conventional implementation.

Figure 2 shows the results of the experiments in $B^\top B$. B is generated using Eq. (44) and $B^\top B\mathbf{x}$ is expanded as Eq. (47). In Fig. 2, the vertical axis indicates the computation time for $B^\top B\mathbf{x}$. The results show that the computation time for $B^\top B\mathbf{x}$ of the proposed implementation is about 75% of that of the conventional implementation.

The size of the matrix C is larger than the size of cache. On the other hand, the size of $C(i,:)$ is smaller than the size of cache. By the results of Figs. 1 and 2, all data of $C(i,:)$ are stored to the cache. Therefore, the proposed implementation is effective.

6. Conclusions

In principal component analysis, only larger singular values and the corresponding singular vectors are needed. To obtain such 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 solved 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, which is a sparse matrix using CRS or CCS formats, can be stored in the caches, high cache hit ratios can be archived.

By using Eq. (47), the normalized data matrix, which is a dense matrix, can be expanded using the given data matrix, which is a sparse matrix in customers research, then the proposed implementation can be adopted. Since the computational order in a sparse matrix is smaller than that in a dense matrix, the proposed implementation using ARPACK is suitable.

We performed experiments to evaluate the proposed implementation. In the experiments, we used a machine with 45 MB L3 caches. Therefore, the size of a row in a target matrix with dimension size 100,000 is much smaller than the size of the L3 cache. The experimental results showed that the computation time of the proposed implementation is about 75% of that of the conventional implementation.

Acknowledgments This work was supported by JSPS KAKENHI Grant Number 17H02858.

References

[1] Arnoldi, W.E.: The principle of minimized iterations in the solution of the matrix eigenvalue problem, *Quart. Appl. Math.*, Vol.9, pp.17–29 (1951).
[2] Baglama, J. and Reichel, L.: Augmented implicitly restarted Lanczos bidiagonalization methods, *SIAM Journal on Scientific Computing*, Vol.27, No.1, pp.19–42 (2005).
[3] Calvetti, D. et al.: An implicitly restarted Lanczos method for large symmetric eigenvalue problems, *Electronic Trans. Numerical Analysis*, Vol.2, No.1, pp.1–21 (1994).
[4] Dongarra, J.: Templates for the Solution of Linear Systema: Building Blocks for Iterative Methods (1995), available from ⟨http://netlib.org/linalg/html_templates/node89.html⟩.
[5] Duff, I., Grimes, R. and Lewis, J.: Sparse matrix test problems, *ACM Trans. Math. Soft.*, Vol.15, pp.1–14 (1989).
[6] Golub, G.H. and Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix, *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, Vol.2, No.2, pp.205–224 (1965).
[7] Golub, G.H. and van Loan, C.F.: *Matrix Computations*, Baltimore, MD, USA: Johns Hopkins University Press (1996).
[8] Halko, N. et al.: Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, *SIAM*

Reviews, Vol.53, No.2, pp.217–288 (2011).
[9] Lehoucq, R.B., Sorensen, D.C. and Yang, C.: ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods (1998), available from ⟨http://www.caam.rice.edu/software/ARPACK⟩.
[10] Lanczos, C.: 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).
[11] Sleijpen, G.L. and Van der Vorst, H.A.: A Jacobi-Davidson iteration method for linear eigenvalue problems, *SIAM Review*, Vol.42, No.2, pp.267–293 (2000).
[12] Sorensen, D.C.: Implicit application of polynomial filters in a k-step Arnoldi method, *SIAM J. Matrix Anal. Appl.*, Vol.13, pp.357–385 (1992).
[13] Sorensen, D.C., Calvetti, D. and Reichel, L.: An Implicitly Restarted Lanczos Method for Large Symmetric Eigenvalue Problems, *Elect. Trans. Numer. Anal.*, Vol.2, pp.1–21 (1994).

Masami Takata is a lecturer of the Research Group of Information and Communication Technology for Life at Nara Women's University. She received her Ph.D. degree from Nara Women's University in 2004. Her research interests include numerical algebra and parallel algorithms for distributed memory systems.

Sho Araki received his B.E. and M.I. degrees from Kyoto University in 2012 and 2014. His research interests include parallel algorithms for eigenvalue and singular value decomposition.

Kinji Kimura received his Ph.D. degree from Kobe University in 2004. He became a PRESTO, COE, and CREST researcher in 2004 and 2005. He became an assistant professor at Kyoto University in 2006, an assistant professor at Niigata University in 2007, a lecturer at Kyoto University in 2008, and has been a program-specific associate professor at Kyoto University since 2009. He is an IPSJ member.

Yuki Fujii received his B.E. and M.I. degrees from Kyoto University in 2013 and 2015. His research interests include the parallel computation of the partial eigenvalue decomposition for sparse matrices.

Yoshimasa Nakamura has been a professor of Graduate School of Informatics, Kyoto University from 2001. His research interests include integrable dynamical systems which originally appear in classical mechanics. But integrable systems have a rich mathematical structure. His recent subject is to design new numerical algorithms such as the mdLVs and I-SVD for singular value decomposition by using discrete-time integrable systems. He is a member of JSIAM, SIAM, MSJ and AMS.