UNIVERSITÄT SALZBURG

Parallel One-Sided Block Jacobi SVD Algorithm with Dynamic Ordering and Variable Blocking: Performance Analysis and Optimization

Shuhei Kudo Yusaku Yamamoto Martin Bečka
Marián Vajteršic

Technical Report 2016-02 March 2016

Department of Computer Sciences

Jakob-Haringer-Straße 2
5020 Salzburg
Austria
www.cosy.sbg.ac.at

Technical Report Series

Parallel One-Sided Block Jacobi SVD Algorithm with Dynamic Ordering and Variable Blocking: Performance Analysis and Optimization

Shuhei Kudo* Yusaku Yamamoto† Martin Bečka‡ Marián Vajteršic§

March 18, 2016

Abstract. *The one-sided block Jacobi (OSBJ) method is known to be an efficient method for computing the singular value decomposition on a parallel computer. In this paper, we focus on the most recent variant of the OSBJ method, the one with parallel dynamic ordering and variable blocking, and present both theoretical and experimental analyses of the algorithm. In the first part of the paper, we provide a detailed theoretical analysis of its convergence properties. In the second part, based on preliminary performance measurement on the Fujitsu FX10 and SGI Altix ICE parallel computers, we identify two performance bottlenecks of the algorithm and propose new implementations to resolve the problem. Experimental results show that they are effective and can achieve up to 2.0 and 1.5 times speedup of the total execution time on the FX10 and the Altix ICE, respectively. As a result, our OSBJ solver outperforms ScaLAPACK PDGESVD in almost all cases when computing the SVD of matrices of order 2160 to 8640 on these machines using 8 to 144 nodes.*

1 Introduction

Singular value decomposition (SVD) is one of the most fundamental matrix computations and has applications in such diverse fields as signal processing, information retrieval and scientific simulation. For example, the principal component analysis involves computing the SVD of a data matrix, which describes m properties of n samples [13]. SVD is also used as a preprocessing step for more sophisticated statistical analysis techniques such as the independent component analysis [6]. In electronic structure calculation using the filter diagonalization method, SVD is used to construct an orthogonal basis of the subspace extracted by the filter [14]. All of these applications require computing the SVD of a large matrix with thousands of rows and columns. Hence, there is a strong need for an efficient SVD algorithm for parallel computers. In this paper, we deal with the computation of SVD of an $m \times n$ dense matrix A ($m \geq n$) on a distributed-memory parallel computer.

The standard algorithm for SVD consists of three steps, namely, transformation of the input matrix to bidiagonal form, SVD of the resulting bidiagonal matrix, and back-transformation of the singular vectors [5]. Common matrix libraries such as LAPACK and ScaLAPACK adopt this approach. However, from the viewpoint of high performance computing, this approach has two drawbacks. First, the bidiagonalization step has fine-grained parallelism and requires $O(n)$ interprocessor communications. This often causes performance bottleneck. Second, half of the computational work in the bidiagonalization step is done in the form of level-2 BLAS (matrix-

*Department of Communication Engineering and Informatics, University of Electro-Communications, Tokyo, Japan, email: k1541013@edu.cc.uec.ac.jp

†Department of Communication Engineering and Informatics, University of Electro-Communications, Tokyo, Japan, email: yusaku.yamamoto@uec.ac.jp

‡Mathematical Institute, Slovak Academy of Sciences, Bratislava, Slovak Republic, email: martin.becka@savba.sk

§Department of Computer Sciences, University of Salzburg, Austria, and Mathematical Institute, Slovak Academy of Sciences, Bratislava, Slovak Republic, email: marian@cosy.sbg.ac.at

vector multiplication). As the level-2 BLAS is a memory-intensive operation and cannot use cache memory efficiently, this tends to lower the performance, especially when the matrix size is large.

As an alternative to the bidiagonalization-based method, the one-sided block Jacobi (OSBJ) method [4, 1, 2, 9] attracts attention recently. In this method, one first partitions the input matrix logically into block columns as

$$A = [A_1, A_2, \ldots, A_\ell], \tag{1}$$

where ℓ is even, A_i has n_i columns ($1 \le i \le \ell$) and $n_1 + n_2 + \cdots + n_\ell = n$. The method starts with

$$A^{(0)} = A \tag{2}$$

and computes for $r = 0, 1, \ldots$ the iterates

$$A^{(r+1)} = A^{(r)} V^{(r)}. \tag{3}$$

Here, $V^{(r)}$ is an $n \times n$ orthogonal matrix designed to make the column vectors of the submatrix $[A_{i_r}^{(r)}, A_{j_r}^{(r)}]$ ($1 \le i_r, j_r, \le \ell$) orthogonal to each other. The indices (i_r, j_r) ($r = 0, 1, \ldots$) are chosen appropriately based on some criterion, as will be explained later. This process is repeated until all the column vectors of $A^{(r)}$ are orthogonal to working accuracy. Then $A^{(r)}$ can be written as $A^{(r)} = U\Sigma$, where U is an $m \times n$ orthogonal matrix and $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a diagonal matrix. Thus one can compute the singular values $\sigma_1, \ldots, \sigma_n$ as the norms of the column vectors of $A^{(r)}$ and the left singular vectors $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n] \in \mathbf{R}^{m \times n}$ as the normalized column vectors of $A^{(r)}$. The right singular vectors $V = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n] \in \mathbf{R}^{n \times n}$ can be obtained by solving $AV = A^{(r)}$. Although this process requires more computational work than the bidiagonalization-based method, it has larger grain parallelism and requires smaller number of interprocessor communications. Moreover, most of the computational work is performed in the form of level-3 BLAS, which is a cache-friendly operation. Thanks to these features, the OSBJ method has the potential to outperform the bidiagonalization-based method on modern distributed-memory parallel machines.

There are many variants of the OSBJ method which differ in the order $\{(i_r, j_r)\}_{r=0,1,\ldots}$ in which the block columns are orthogonalized and in the way the matrix is distributed across the computing nodes. As for the ordering, we focus on the so-called *parallel dynamic ordering strategy* [1]. In this strategy, at each step r, one chooses $\ell/2$ independent pairs $(i_{r,1}, j_{r,1})$, …, $(i_{r,\ell/2}, j_{r,\ell/2})$ based on the mutual perpendicularity of all pairs of block columns, and then orthogonalize the columns of the $\ell/2$ submatrices $[A_{i_{r,1}}^{(r)}, A_{j_{r,1}}^{(r)}], \ldots, [A_{i_{r,\ell/2}}^{(r)}, A_{j_{r,\ell/2}}^{(r)}]$ in parallel. It has been shown that this strategy is effective in reducing the number of iterations of the OSBJ method for a wide class of random matrices. As for the matrix distribution, we assume *variable blocking* [2], in which the blocks $A_i^{(r)}$ ($i = 1, 2, \ldots, \ell$) are divided into column subblocks and each subblock is allocated to one computing node. The approach is shown to achieve better parallel efficiency than the conventional fixed-size blocking, where the pair $[A_i^{(r)}, A_j^{(r)}]$ is allocated to one node.

In this paper, we provide both theoretical and experimental performance analyses of the OSBJ method with parallel dynamic ordering and variable blocking. In the theoretical part, we discuss its global convergence and asymptotic convergence properties. In the parallel dynamic ordering strategy, computing the mutual perpendicularity of all pairs of block columns is costly, so they are usually computed using some approximation [2]. We also discuss the effect of this approximation on the convergence properties. In the experimental part, we first evaluate the performance of the OSBJ method on two distributed-memory parallel computers, namely, Fujitsu PRIMEHPC FX10 and SGI Altix ICE, and analyze the results in detail. Then we identify two performance bottlenecks and propose new implementations to overcome them. Finally, we evaluate the performance of the improved implementation on the two parallel machines using up to 144 nodes. The proposed implementations prove to be quite effective and can achieve up to 2.0 and 1.5 times speedup on the FX10 and Altix ICE, respectively. Performance comparison with the ScaLAPACK SVD routine PDGESVD is also given.

The rest of this paper is organized as follows. In Section 2, we describe the algorithm of the parallel OSBJ method with dynamic ordering and variable blocking. Section 3 provides theoretical convergence analysis of the algorithm, including global convergence and asymptotic convergence properties. Performance results on the FX10 and Altix ICE are presented in Section 4. In Section 5, we propose possible improvements and investigate their effectiveness by numerical experiments. Section 6 gives some concluding remarks.

2 Parallel one-sided block Jacobi algorithm with dynamic ordering and variable blocking

In this section, we describe the algorithm and implementation of the parallel OSBJ method with dynamic ordering and variable blocking [2]. We assume that at the beginning of the algorithm, the matrix A is logically partitioned as given in (1), whereby each submatrix A_i is divided into k column subblocks and each subblock is allocated to one node. Thus the total number of computing nodes is $p = \ell k$. In the following, we assume $n_1 = n_2 = \cdots = n_\ell = n/\ell$ for simplicity.

2.1 The algorithm

The algorithm consists of preprocessing, iteration and postprocessing. Here we explain each of these stages without going into the details of parallelization and data distribution, which will be discussed in the next subsection.

2.1.1 Preprocessing

The preprocessing is intended to concentrate the Frobenius norm of $A^\top A$ near the diagonal, thereby accelerating the convergence of the OSBJ method (see [4, 9] for details). The preprocessing consists of the following two tasks. Both of them are computed using ScaLAPACK.

QR preprocessing The input matrix A is decomposed as $A = Q_1 R$, where $Q_1 \in \mathbf{R}^{m\times n}$ is a matrix with orthonormal columns and $R \in \mathbf{R}^{n\times n}$ is an upper triangular matrix.

LQ preprocessing The matrix R obtained in the previous step is decomposed as $R = LQ_2$, where $L \in \mathbf{R}^{n\times n}$ is a lower triangular matrix and $Q_2 \in \mathbf{R}^{n\times n}$ is an orthogonal matrix. After the decomposition, we set $A^{(0)} = L$.

2.1.2 Iteration

After preprocessing, iteration process (3) gets started with following tasks performed in one iteration.

Weight computation (WC) [2] For every block column pair $[A_i^{(r)}, A_j^{(r)}]$, where $1 \le i < j \le \ell$, we compute its weight $\hat{w}_{ij}^{(r)}$ by

$$\hat{w}_{ij}^{(r)} \equiv \frac{\|(A_i^{(r)})^\top A_j^{(r)}\mathbf{e}\|_2}{\|\mathbf{e}\|_2}, \tag{4}$$

where $\mathbf{e} = (1, 1, \ldots, 1)^\top \in \mathbf{R}^{n/\ell}$. The value of $\hat{w}_{ij}^{(r)}$ can be viewed as an approximate indicator of how mutually inclined the two subspaces $\mathrm{Im}(A_i^{(r)})$ and $\mathrm{Im}(A_j^{(r)})$ are, and will be used to define the ordering. A more exact indicator would be $w_{ij}^{(r)} \equiv \|(A_i^{(r)})^\top A_j^{(r)}\|_F$, but it is computationally too heavy, requiring $2m(n/\ell)^2$ work per pair. In contrast, (4) requires only $4m(n/\ell)$ work and has been shown to give a reasonably accurate results [2]. In the following, we sometimes call $\hat{w}_{ij}^{(r)}$ the *approximate weight* or *simplified weight* and $w_{ij}^{(r)}$ the *exact weight*.

Greedy ordering [1, 2] Let $\mathcal{I} = \{1, 2, \ldots, \ell\}$. In the greedy ordering, we choose the first block column pair by $(i_{r,1}, j_{r,1}) = \mathrm{argmax}_{i,j\in\mathcal{I}, i<j} \hat{w}_{ij}^{(r)}$, so that the most mutually inclined pair is orthogonalized. Then we set $\mathcal{I} := \mathcal{I}\backslash\{i_{r,1}, j_{r,1}\}$ and choose the second pair again by $(i_{r,2}, j_{r,2}) = \mathrm{argmax}_{i,j\in\mathcal{I}, i<j} \hat{w}_{ij}^{(r)}$. This process is repeated until $\mathcal{I}$ becomes empty and $\ell/2$ block column pairs $\{(i_{r,s}, j_{r,s})\}_{s=1}^{\ell/2}$ are obtained. It is clear from the construction that these $\ell/2$ pairs can be orthogonalized independently. Note that the greedy ordering is an inherently sequential operation, so it will not be parallelized but will be executed by all nodes redundantly.

Computation of the Gramian (GRAM) For each pair $[A^{(r)}_{i_{r,s}}, A^{(r)}_{j_{r,s}}]$ $(1 \leq s \leq \ell/2)$, we compute the 2×2 block Gram matrix $G_s \in \mathbf{R}^{(2n/\ell)\times(2n/\ell)}$ by

$$G_s = [A^{(r)}_{i_{r,s}}, A^{(r)}_{j_{r,s}}]^\top [A^{(r)}_{i_{r,s}}, A^{(r)}_{j_{r,s}}]. \tag{5}$$

2×2 *block EVD (EVD)* For $1 \leq s \leq \ell/2$, diagonalize the Gram matrix as $G_s = V_s D_s V_s^\top$, where V_s is an orthogonal matrix and D_s is a diagonal matrix.

Matrix-matrix multiplication (MM) For $1 \leq s \leq \ell/2$, compute the block columns of $A^{(r+1)}$ at the next iteration by the matrix-matrix multiplication

$$[A^{(r+1)}_{i_{r,s}}, A^{(r+1)}_{j_{r,s}}] = [A^{(r)}_{i_{r,s}}, A^{(r)}_{j_{r,s}}] V_s. \tag{6}$$

Then it is easy to see that the column vectors of $[A^{(r+1)}_{i_{r,s}}, A^{(r+1)}_{j_{r,s}}]$ are mutually orthogonalized.

The iteration process is repeated until all columns of $A^{(r)}$ are mutually orthogonal to working accuracy.

2.1.3 Postprocessing

After finishing the iteration process at iteration r, we compute the singular values and singular vectors of A as follows.

Computation of the singular values and left singular vectors of $A^{(0)}$ The singular values of $A^{(0)}$ are computed as the norms of the column vectors of $A^{(r)}$. Of course, they are also the singular values of A. The left singular vectors $U^{(0)} = [\mathbf{u}^{(0)}_1, \ldots, \mathbf{u}^{(0)}_n]$ of $A^{(0)}$ are computed as the normalized column vectors of $A^{(r)}$.

Computation of the right singular vectors of $A^{(0)}$ The right singular vectors of $A^{(0)}$ are computed by solving the equation $A^{(0)}V^{(0)} = A^{(r)}$ [2]. Note that this can be solved very easily because $A^{(0)}$ is a lower triangular matrix. The right singular vectors could be computed by accumulating the orthogonal transformations used in (3), but the above process is computationally more efficient. Also, it has been shown that there is no problem regarding the orthogonality of computed $V^{(0)}$, even though $V^{(0)}$ is given as a solution to the triangular equation and no explicit orthogonalization is performed.

QR postprocessing Compute the left singular vectors of A by $U = Q_1 U^{(0)}$.

LQ postprocessing Compute the right singular vectors of A by $V = Q_2^\top V^{(0)}$.

2.2 Parallelization and data distribution

In the algorithm described above, three types of data distribution are used.

At the beginning of the algorithm, each logical block A_i $(1 \leq i \leq \ell)$ is partitioned into $k = p/\ell$ column subblocks and each subblock is allocated to one computing node. We call this data distribution *DIST1*.

In the preprocessing and postprocessing stages, the QR decomposition, the LQ decomposition, the QR postprocessing and the LQ postprocessing are done using ScaLAPACK. Because these are operations on the entire matrix, all computing nodes are involved. Thus the matrix is distributed among all the nodes using block cyclic data layout. We call this data distribution *DIST2*.

In the weight computation (WC) task, we again use DIST1. The computation of $\hat{w}^{(r)}_{ij}$ is performed by those $2k$ nodes, to which the block columns $A^{(r)}_i$ and $A^{(r)}_j$ are allocated. Thus $\ell/2$ weights can be computed in parallel.

Table 1: Computational work, pattern and granularity of each task.

task	work	routine	matrix size
WC	$n^2\ell^2/p$	DGEMV	$n \times (n/\ell)$
GRAM	n^3/p	DSYRK	$n \times (2n/\ell)$
EVD	$n^3/(\ell p)$	EVD	$(2n/\ell) \times (2n/\ell)$
MM	n^3/p	DGEMM	$n \times (2n/\ell)$, $(2n/\ell) \times (2n/\ell)$

In the GRAM, EVD and MM tasks, operations on the disjoint $\ell/2$ subblock pairs $[A^{(r)}_{i_{r,s}}, A^{(r)}_{j_{r,s}}]$ $(1 \leq s \leq \ell/2)$ can be done independently. Operations on each subblock pair are performed by $2k$ nodes using ScaLAPACK. Hence, the subblock pair is distributed among these nodes using block cyclic data distribution. We call this data distribution *DIST3*.

From the data distribution stated above, it is clear that redistribution of the matrix data is necessary before and after the preprocessing and postprocessing stages, before the GRAM task, and after the MM task.

2.3 Computational work of each task

In the variable blocking scheme, ℓ can be chosen arbitrarily as long as ℓ is even, $\ell \geq 2$ and p is divisible by ℓ. Here we consider how the computational work and granularity of each task change depending on ℓ. Since the computations in the preprocessing and postprocessing stages do not depend on ℓ, we consider only the tasks in the iteration. We also exclude the greedy ordering, which requires far less computational work than the other tasks.

When the number of block columns is ℓ, there are $\ell(\ell-1)/2$ block pairs and $\ell/2$ of them can be orthogonalized in one iteration in parallel. Hence, for performing $\ell(\ell-1)/2$ orthogonalizations, there are $\ell-1$ iterations needed, which we define as one *sweep*. For each of the WC, GRAM, EVD and MM tasks, its order of computational work per sweep and per node, computational pattern and the size of matrices appearing in it are summarized in Table 1. Here, DGEMV, DSYRK and DGEMM denote routines for matrix-vector multiplication, multiplication of a matrix with its transpose, and multiplication of two matrices, respectively. As a result of preprocessing, the matrix is an $n \times n$ square matrix. It can be seen that the complexity of WC increases with ℓ, while that of EVD decreases with ℓ. Thus it is expected that the optimal value of ℓ is determined from the trade-off between these two effects.

3 Theoretical convergence properties

In this section, we discuss theoretical convergence properties of the one-sided block Jacobi method with parallel dynamic ordering strategy. It can be shown that the method has convergence properties similar to those of the (sequential) classical point Jacobi method [11][12]. Specifically, if the exact weights $w^{(r)}_{ij} = \|(A^{(r)}_i)^\top A^{(r)}_j\|_F$ are used, the method is globally convergent and the order of convergence is asymptotically quadratic. We prove this by using the convergence theorems of the block Jacobi method for the symmetric eigenvalue problem [15]. Of course, our implementation uses simplified weights, namely, $\hat{w}^{(r)}_{ij} = \|(A^{(r)}_i)^\top A^{(r)}_j \mathbf{e}\|_2 / \|\mathbf{e}\|_2$ and the effect of this must be taken into account. We consider this problem at the end of this section.

In the following, we assume that the initial matrix $A^{(0)}$ is logically partitioned into ℓ blocks as in (1). Note that the physical partition of each block $A^{(0)}_i$ $(1 \leq i \leq \ell)$ does not affect convergence behaviors.

3.1 Relationship with the block Jacobi method for the symmetric eigenproblem

To study the convergence properties of our algorithm, we first note that by applying the one-sided block Jacobi method to $A^{(0)}$, we are implicitly applying the block Jacobi method for the symmetric eigenvalue problem to $B^{(0)} \equiv \left(A^{(0)}\right)^\top A^{(0)}$. In fact, making the column vectors of the submatrix $[A_i^{(r)}, A_j^{(r)}]$ mutually orthogonal by multiplying $V^{(r)}$ to $A^{(r)}$ from the right is equivalent to applying the orthogonal transformation

$$B^{(r+1)} = (V^{(r)})^\top B^{(r)} V^{(r)} \tag{7}$$

to $B^{(r)} = (A^{(r)})^\top A^{(r)}$ to diagonalize its 2×2 subblock

$$\begin{bmatrix} B_{ii}^{(r)} & B_{ij}^{(r)} \\ B_{ji}^{(r)} & B_{jj}^{(r)} \end{bmatrix}. \tag{8}$$

The exact weight $w_{ij}^{(r)} = \|(A_i^{(r)})^\top A_j^{(r)}\|_F$ in the OSBJ method corresponds to the Frobenius norm $\|B_{ij}\|_F$ of the (i,j)th off-diagonal block in the block Jacobi method for the symmetric eigenvalue problem. In the following two subsections, we consider only the exact weights. The effects of using the approximate weights will be discussed in subsection 3.4.

3.2 Global convergence

For the block Jacobi method for the symmetric eigenvalue problem, we can show the following global convergence theorem. This is a slight extension of the global convergence theorem given in [15]. In the following, let $L = \ell(\ell-1)/2$ and denote the sum of squares of Frobenius norms of off-diagonal blocks at the rth step by $S^{(r)} = \sum_{i \neq j} \|B_{ij}^{(r)}\|_F^2$.

Theorem 3.1. *In the block Jacobi method for the symmetric eigenvalue problem, assume that the off-diagonal block $B_{i_r,j_r}^{(r)}$ ($r = 0, 1, \ldots$) chosen for elimination at each step has a squared Frobenius norm $\|B_{i_r,j_r}^{(r)}\|_F^2$ that is larger than or equal to the average squared Frobenius norm over the $\ell(\ell-1)$ off-diagonal blocks of $B^{(r)}$. Then $S^{(r)}$ satisfies*

$$S^{(r+1)} \leq \left(1 - \frac{1}{L}\right) S^{(r)}, \tag{9}$$

and therefore converges to zero as $r \to \infty$.

Proof. Let us rewrite $B^{(r)}$, $B^{(r+1)}$, i_r and j_r as B, $\hat{B}$, x and y, respectively, for simplicity. Since the unitary transformation by $V^{(r)}$ works only on the xth and yth block rows and the xth and yth block columns of B, the Frobenius norm of the 2×2 block submatrix

$$\tilde{B} = \begin{bmatrix} B_{xx} & B_{xy} \\ B_{yx} & B_{yy} \end{bmatrix} \tag{10}$$

remains unchanged after the transformation. Since $\hat{B}_{xy} = O$ and $\hat{B}_{yx} = O$, we have

$$\|\hat{B}_{xx}\|_F^2 + \|\hat{B}_{yy}\|_F^2 = \|B_{xx}\|_F^2 + \|B_{yy}\|_F^2 + 2\|B_{xy}\|_F^2. \tag{11}$$

Noting that other diagonal blocks of B than B_{xx} and B_{yy} are unchanged by the unitary transformation and $\|B\|_F^2 = \|\hat{B}\|_F^2$, we have

$$\begin{aligned} S^{(r+1)} = \sum_{i \neq j} \|\hat{B}_{ij}\|_F^2 &= \sum_{i \neq j} \|B_{ij}\|_F^2 - 2\|B_{xy}\|_F^2 \\ &\leq \sum_{i \neq j} \|B_{ij}\|_F^2 - \frac{1}{L} \sum_{i \neq j} \|B_{ij}\|_F^2 \\ &= \left(1 - \frac{1}{L}\right) \sum_{i \neq j} \|B_{ij}\|_F^2 = \left(1 - \frac{1}{L}\right) S^{(r)}, \end{aligned} \tag{12}$$

where we used

$$\|B_{xy}\|_F^2 \geq \frac{1}{2L} \sum_{i \neq j} \|B_{ij}\|_F^2 \tag{13}$$

from the assumption. □

Now let us define the *parallel* block Jacobi method for the symmetric eigenvalue problem as a method that eliminates $\ell/2$ off-diagonal blocks $B^{(r)}_{i_{r,1},j_{r,1}}, \ldots, B^{(r)}_{i_{r,\ell/2},j_{r,\ell/2}}$, along with the blocks in their transposed positions, at each step. Here, $i_{r,1}, j_{r,1}, \ldots, i_{r,\ell/2}, j_{r,\ell/2}$ are assumed to be some permutation of $1, 2, \ldots, \ell$. The global convergence property of this method follows immediately from Theorem 3.1.

Corollary 3.2. *In the parallel block Jacobi method for the symmetric eigenvalue problem, assume that at least one of the $\ell/2$ off-diagonal blocks chosen for elimination at each step has a squared Frobenius norm that is larger than or equal to the average squared Frobenius norm over the $\ell(\ell-1)$ off-diagonal blocks of $B^{(r)}$. Then the off-diagonal elements of $B^{(r)}$ converges to zero as $r \to \infty$.*

Proof. Let the off-diagonal block that has a squared Frobenius norm larger than or equal to the average squared Frobenius be B_{xy}. As shown in Theorem 3.1, by eliminating this block[1], $S^{(r)}$ decreases at least by a factor of $1 - \frac{1}{L}$. By eliminating other $\ell/2 - 1$ off-diagonal blocks at the same step, $S^{(r)}$ does not increase. Hence $S^{(r)} \to 0$ as $r \to 0$ and all the off-diagonal blocks tend to zero. In addition, the off-diagonal elements of the diagonal blocks become zero by the orthogonal transformations at the first step and remain zero thereafter. □

By restating this result in the language of the OSBJ method, we finally obtain the global convergence theorem for the OSBJ method with parallel dynamic ordering strategy.

Theorem 3.3. *In the OSBJ method with parallel dynamic ordering, assume that at least one of the block column pairs $(i_{r,1}, j_{r,1}), \ldots, (i_{r,\ell/2}, j_{r,\ell/2})$ chosen at each step has a squared weight $w^2_{i_{r,k},j_{r,k}} = \|(A^{(r)}_{i_{r,k}})^\top A^{(r)}_{j_{r,k}}\|_F^2$ that is greater than or equal to the average squared weight over the $\ell(\ell-1)$ possible block pairs. Then the column vectors $\{\mathbf{a}_i^{(r)}\}_{i=1}^n$ of $A^{(r)}$ satisfy $\lim_{r\to\infty}(\mathbf{a}_i^{(r)})^\top \mathbf{a}_j^{(r)} = 0$ for $i \neq j$.*

Proof. This follows directly from Corollary 3.2 because the condition on the choice of the block column pairs translates into the condition on the choice of the off-diagonal blocks in Corollary 3.2 and $b_{ij}^{(r)} \to 0$ $(i \neq j)$ means $\lim_{r\to\infty}(\mathbf{a}_i^{(r)})^\top \mathbf{a}_j^{(r)} = 0$ $(i \neq j)$. □

Of course, the block column pairs chosen by the greedy parallel dynamic ordering satisfy the assumption of Theorem 3.3. This guarantees that our parallel OSBJ method is globally convergent if the weights are computed as $w_{ij}^{(r)} = \|(A_i^{(r)})^\top A_j^{(r)}\|_F$.

3.3 Local quadratic convergence

In the parallel block Jacobi method for the symmetric eigenvalue problem, the order of convergence is ultimately quadratic if all eigenvalues are simple and the $\ell/2$ off-diagonal blocks eliminated at each step are chosen by the greedy algorithm based on the Frobenius norm [15]. In the following, we present a slightly extended version of this theorem, in which the assumption on the choice of the off-diagonal blocks is relaxed. We assume that $B^{(0)}$ is an $n \times n$ symmetric matrix with *simple* eigenvalues and let $d = \min_{i\neq j} |\lambda_i - \lambda_j|$. We also denote the block size by $q = n/\ell$.

Theorem 3.4. *Assume that at the rth step of the parallel block Jacobi method, $\delta \equiv \max_{i\neq j} \|B_{ij}^{(r)}\|_F \leq \frac{d}{4\ell\sqrt{qL}}$ is satisfied*[2]*. Assume also that at the $(r+k)$th step ($k = 0, 1, \ldots, L-1$), the set of $\ell/2$ off-diagonal blocks is chosen so that it contains at least one block, say $B^{(r+k)}_{i_k,j_k}$, that satisfies either (i) or (ii) of the following conditions.*

[1] Of course, by eliminating this block, we eliminate B_{yx} at the same time. But we will not mention this for simplicity.

[2] The definition of δ in Theorem 4.2 in [15] should be $\delta = \max_{I\neq J} \|A_{IJ}^{(m)}\|_F$ instead of $\delta = \sum_{I\neq J} \|A_{IJ}^{(m)}\|_F$.

(i) $B_{i_k,j_k}^{(r+k)}$ *is among the* $2L-2k$ *off-diagonal blocks with the largest Frobenius norm.*

(ii) $\|B_{i_k,j_k}^{(r+k)}\|_F > \frac{4L(k-1)\sqrt{q}}{d}\delta^2$.

Then the matrix $B^{(r+L)}$ *satisfies*

$$\|B_{i_k,j_k}^{(r+L)}\|_F \leq \frac{4L^2\sqrt{q}}{d}\,\delta^2 \quad (i \neq j), \tag{14}$$

that is, the Frobenius norms of the off-diagonal blocks converge to zero quadratically after every L *steps.*

Proof. Just as in the proof of Theorem 4.3 in [15], we show by induction that among the $2L$ off-diagonal blocks of $B^{(r+k)}$ $(k = 1, 2, \ldots, L)$, there are at least $2k$ blocks whose Frobenius norm is smaller than or equal to $\frac{4L(k-1)\sqrt{q}}{d}\delta^2$. Clearly, this statement is true for $k = 1$, because $B^{(r+1)}$ has ℓ zero blocks. Now assume that the statement is true for some k. We first consider case (i). If $\|B_{i_k,j_k}^{(r+k)}\|_F > \frac{4L(k-1)\sqrt{q}}{d}\delta^2$, the case reduces to case (ii), so we can safely assume that $\|B_{i_k,j_k}^{(r+k)}\|_F \leq \frac{4L(k-1)\sqrt{q}}{d}\delta^2$. But because $B_{i_k,j_k}^{(r+k)}$ is among the $2L-2k$ off-diagonal blocks with the largest Frobenius norm, there must be at least $2k+2$ off-diagonal blocks (including $B_{i_k,j_k}^{(r+k)}$ and $B_{j_k,i_k}^{(r+k)}$) with Frobenius norm smaller or equal to $\frac{4L(k-1)\sqrt{q}}{d}\delta^2$ in $B^{(r+k)}$. As shown in the proofs of Theorems 4.2 and 4.3 in [15], the Frobenius norms of these blocks increase at most by $\frac{4L\sqrt{q}}{d}\delta^2$ by the transformation from $B^{(r+k)}$ to $B^{(r+k+1)}$. Hence the statement holds also for $k+1$. Next we consider case (ii). In this case, $B_{i_k,j_k}^{(r+k)}$ is not included in the $2k$ off-diagonal blocks mentioned in the statement. By the transformation from $B^{(r+k)}$ to $B^{(r+k+1)}$, $B_{i_k,j_k}^{(r+k)}$ and $B_{j_k,i_k}^{(r+k)}$ become zero, while the Frobenius norms of the $2k$ blocks increase at most by $\frac{4L\sqrt{q}}{d}\delta^2$. Hence the statement holds for $k+1$ also in this case. Thus the statement holds for $k = 1, 2, \ldots, L$ and the theorem follows directly from the statement for $k = L$. □

From this result, we immediately have the corresponding theorem for the OSBJ method with parallel dynamic ordering based on the exact weight $w_{ij}^{(r)} = \|(A_i^{(r)})^\top A_j^{(r)}\|_F$. We assume that all the singular values of A are simple and let $d = \min_{i\neq j} |\sigma_i^2 - \sigma_j^2|$.

Theorem 3.5. *Assume that at the* r*th step of the OSBJ method with parallel dynamic ordering, the condition* $\delta \equiv \max_{i\neq j} w_{ij}^{(r)} \leq \frac{d}{4\ell\sqrt{qL}}$ *is satisfied. Assume also that at the* $(r+k)$*th step* $(k = 0, 1, \ldots, L-1)$*, the set of* $\ell/2$ *block column pairs is chosen so that it contains at least one pair, say* $[A_{i_k}^{(r+k)}, A_{j_k}^{(r+k)}]$*, that satisfies either (i) or (ii) of the following conditions.*

(i) $[A_{i_k}^{(r+k)}, A_{j_k}^{(r+k)}]$ *is among the* $L-k$ *pairs with the largest weight.*

(ii) $w_{i_k,j_k}^{(r+k)} > \frac{4L(k-1)\sqrt{q}}{d}\delta^2$.

Then, for the matrix $A^{(r+L)}$ *obtained after* L *steps, the weights satisfy*

$$w_{ij}^{(r+L)} \leq \frac{4L^2\sqrt{q}}{d}\,\delta^2 \quad (i \neq j), \tag{15}$$

that is, the weights, or $\|(A_i^{(r)})^\top A_j^{(r)}\|_F$*, converge to zero quadratically after every* L *steps.*

Clearly, the assumption (i) of Theorem 3.5 is satisfied by the OSBJ method with parallel dynamic ordering, if the exact weights are used and the $\ell/2$ column pairs are chosen by the greedy algorithm. Hence, the method converges ultimately quadratically if all singular values of A are simple.

3.4 Effects of approximate weight computation

So far, we have assumed that the weights are computed by $w_{ij}^{(r)} = \|(A_i^{(r)})^\top A_j^{(r)}\|_F$. However, in our implementation described in subection 2.1, we use approximate weights defined by (4) to reduce the computational cost. In this subsection, we study the effect of this approximation on the theoretical convergence properties.

As a preparation, we point out a relationship between $\|C\|_F$ and $\|C\mathbf{e}\|_2/\|\mathbf{e}\|_2$, where $\mathbf{e} = (1, 1, \ldots, 1)^\top$, for a general $p \times q$ matrix C. In a sense, the latter quantity can be viewed as a stochastic approximation to $\|C\|_F$. To be specific, let us consider a set of $p \times q$ matrices with Frobenius norm f. Then, if we draw a matrix C randomly from this set and compute $(\|C\mathbf{e}\|_2/\|\mathbf{e}\|_2)^2$, its expectation value is proportional to f^2.

To make this statement more precise, we define a set of $p \times q$ matrices (assuming $p \geq q$ for simplicity) with prescribed Frobenius norm f by

$$\mathcal{M}_f = \left\{ U \begin{bmatrix} \Sigma \\ O_{(p-q)\times q} \end{bmatrix} V^\top \middle| U \in \mathcal{O}(p), V \in \mathcal{O}(q), \mathrm{Tr}(\Sigma^2) = f^2 \right\}. \tag{16}$$

Here, $\mathcal{O}(p)$ and $\mathcal{O}(q)$ denote the set of $p \times p$ and $q \times q$ orthogonal matrices, respectively. Furthermore, we introduce a probability measure into $\mathcal{O}(q)$ to compute the expectation value. We use the *Haar measure* [7] $\mu_1(V)$, which is a unique measure on $\mathcal{O}(q)$ that is invariant under multiplication by any element of $\mathcal{O}(q)$, that is, for any measurable subset $\mathcal{V} \subseteq \mathcal{O}(q)$ and any element $V_0 \in \mathcal{O}(q)$,

$$\mu_1(\mathcal{V}V_0) = \mu_1(\mathcal{V}). \tag{17}$$

Similarly, we assume the Haar measure $\mu_2(U)$ for $\mathcal{O}(p)$. As for the set of diagonal matrices $\{\Sigma \,|\, \mathrm{Tr}(\Sigma^2) = f^2\}$, we write its measure as $\mu_3(\Sigma)$, but will not make any assumption on it, because as we will see, the expectation value of $(\|C\mathbf{e}\|_2/\|\mathbf{e}\|_2)^2$ does not depend on it.

Now we are ready to state the theorem concerning the relationship between $\|C\|_F$ and $\|C\mathbf{e}\|_2/\|\mathbf{e}\|_2$.

Theorem 3.6. *Assume that C is a $p \times q$ matrix randomly chosen from $\mathcal{M}_f$ with respect to the probability measures defined above. Then*

$$\mathcal{E}[(\|C\mathbf{e}\|_2/\|\mathbf{e}\|_2)^2] = cf^2, \tag{18}$$

where $\mathcal{E}[\cdot]$ is the expectation value operator in $\mathcal{M}_f$ and c is a constant that depends only on q.

Proof. First, we fix the singular values $\sigma_1, \ldots, \sigma_q$ subject to the constraint $\sum_{i=1}^q \sigma_i^2 = f^2$ and regard only U and V as random variables. Let E_i be a $q \times q$ matrix whose (i, i) element is one and all the other elements are zero. Then we can rewrite $\|C\mathbf{e}\|_2^2$ as

$$\|C\mathbf{e}\|_2^2 = \mathbf{e}^\top V \Sigma^2 V^\top \mathbf{e} = \sum_{i=1}^q \sigma_i^2 \mathbf{e}^\top V E_i V^\top \mathbf{e}. \tag{19}$$

Hence, the expectation value under the fixed Σ can be computed as

$$\mathcal{E}[\,\|C\mathbf{e}\|_2^2 \,|\, \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_q)] = \sum_{i=1}^q \sigma_i^2 \mathcal{E}[\mathbf{e}^\top V E_i V^\top \mathbf{e}]. \tag{20}$$

Now we show that $\mathcal{E}[\mathbf{e}^\top V E_i V^\top \mathbf{e}]$ does not depend on i. Noting that the argument of the expectation operator does not depend on U and that for any i and j, $E_i = P_{ij} E_j P_{ij}^\top$ for some $P_{ij} \in \mathcal{O}(q)$ (actually, P_{ij} is a permutation matrix), we have

$$\begin{aligned}
\mathcal{E}[\mathbf{e}^\top V E_i V^\top \mathbf{e}] &= \int_{V \in \mathcal{O}(q)} \mathbf{e}^\top V E_i V^\top \mathbf{e}\, d\mu_1(V) \\
&= \int_{V \in \mathcal{O}(q)} \mathbf{e}^\top V P_{ij} E_j P_{ij}^\top V^\top \mathbf{e}\, d\mu_1(V) \\
&= \int_{VP_{ij} \in \mathcal{O}(q)} \mathbf{e}^\top (VP_{ij}) E_j (VP_{ij})^\top \mathbf{e}\, d\mu_1(VP_{ij}) \\
&= \int_{V' \in \mathcal{O}(q)} \mathbf{e}^\top V' E_j V'^\top \mathbf{e}\, d\mu_1(V') = \mathcal{E}[\mathbf{e}^\top V E_j V^\top \mathbf{e}],
\end{aligned} \tag{21}$$

where we used the property of the Haar measure (Eq. (17)) in the third equality. By letting $c' = \mathcal{E}[\mathbf{e}^\top V E_i V^\top \mathbf{e}]$, which is a constant that depends only on q, and inserting this into (20), we have

$$\mathcal{E}[\,\|C\mathbf{e}\|_2^2 \,|\, \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_q)] = c' \sum_{i=1}^{q} \sigma_i^2 = c' f^2. \tag{22}$$

Finally, we take the expectation value of (22) regarding Σ as a random variable with measure $\mu_3(\Sigma)$ subject to the constraint $\mathrm{Tr}(\Sigma^2) = f^2$. But the result is clearly the same because the right-hand side depends only on f^2 and not on the individual singular values. Thus we have

$$\mathcal{E}[(\|C\mathbf{e}\|_2/\|\mathbf{e}\|_2)^2] = c' f^2/\|\mathbf{e}\|_2^2 \equiv c f^2. \tag{23}$$

□

Now we discuss the implications of Theorem 3.6 on the convergence properties of the OSBJ method. In our situation, we have $C = (A_i^{(r)})^\top A_j^{(r)}$, $f = \|(A_i^{(r)})^\top A_j^{(r)}\|_F = w_{ij}^{(r)}$ and $\|C\mathbf{e}\|_2/\|\mathbf{e}\|_2 = \|(A_i^{(r)})^\top A_j^{(r)}\mathbf{e}\|_2/\|\mathbf{e}\|_2 = \hat{w}_{ij}^{(r)}$. Thus Theorem 3.6 asserts that for a randomly chosen $(A_i^{(r)})^\top A_j^{(r)}$ (under the probability measure specified in Theorem 3.6), the expectation value of the squared approximate weight $(\hat{w}_{ij}^{(r)})^2$ is proportional to the squared exact weight $(w_{ij}^{(r)})^2$. In this sense, $\hat{w}_{ij}^{(r)}$ can be regarded as an approximation to $w_{ij}^{(r)}$.

In Theorem 3.3, we showed that the method is globally convergent if at least one of the $\ell/2$ block column pairs chosen at each step has a squared weight that is greater than or equal to the average squared weight. This is a rather loose condition and is likely to be satisfied in most cases even when $w_{ij}^{(r)}$ is replaced by $\hat{w}_{ij}^{(r)}$. Even if this condition fails to be satisfied at some step, it only delays the convergence by one step.

According to Theorem 3.5, the OSBJ method converges ultimately quadratically if at least one of the $\ell/2$ block column pairs chosen at each step satisfies either condition (i) or (ii). When k is small, say $k < L/2$, condition (i) is a loose condition because more than half of the L possible pairs satisfy this condition. As k increases, the number of block column pairs whose weight is $O(\delta^2)$ increases. On the other hand, the block column pairs which have never been orthogonalized after the rth step have in general weights of $O(\delta)$. Thus it seems possible to distinguish them even with the approximate weights $\hat{w}_{ij}^{(r)}$. Even if neither condition (i) nor (ii) is satisfied at some step, it only results in extending the period L of quadratic convergence. In conclusion, the OSBJ method with parallel dynamic ordering is likely to converge ultimately quadratically even if the approximate weights are used in the greedy algorithm.

4 Performance results on the Fujitsu FX10 and SGI Altix ICE

For the performance evaluation, we varied the number of nodes p from 8 to 144, by running one MPI process per processor (hence 1 process/node on the FX10 and 2 processes/node on the Altix ICE). Each MPI process consisted of 16 or 12 threads, on the FX10 and the Altix ICE, respectively, which were used by multi-threaded BLAS routines called from ScaLAPACK and PBLAS. To construct the test matrices, we used ScaLAPACK test routine pdlagge [10], which generates a random matrix with specified singular values. The singular values were drawn from the normal distribution $N(0, 1)$. The order of matrices was $n = 2160$, 4320 and 8640. The blocking factor ℓ was chosen from an even divisor of p greater than or equal to 4. For example, in the case of $p = 36$, $\ell = 4, 6, 12$ and 36 was used. The corresponding number of nodes in charge of one block column pair was $p/(\ell/2) = 72/\ell$, i.e. 18, 12, 6 and 2 respectively. For each of the cases, the process grid used in PDSYEVD to distribute the matrix data in 2-dimensional block cyclic fashion was chosen so that it is as close to square as possible.

To evaluate the performance of the parallel OSBJ method with dynamic ordering and variable blocking, we implemented the algorithm using FORTRAN and MPI. As a substance for that, we used a code which was presented in [3]. In the GRAM, EVD and MM tasks, we used ScaLAPACK and PBLAS routines PDSYRK, PDSYEVD and PDGEMM, respectively. All experiments were executed on the following two distributed-memory parallel computers.

Table 2: Number of iterations and sweeps of the OSBJ method.

n		$\ell = 144$	$\ell = 72$	$\ell = 36$	$\ell = 24$	$\ell = 12$	$\ell = 8$	$\ell = 6$	$\ell = 4$
4096	# of iterations	854	390	178	113	49	28	20	10
	# of sweeps	5.97	5.49	5.09	4.91	4.45	4	4	3.33
8192	# of iterations	846	383	177	111	50	29	22	10
	# of sweeps	5.92	5.39	5.06	4.83	4.55	4.14	4.4	3.33

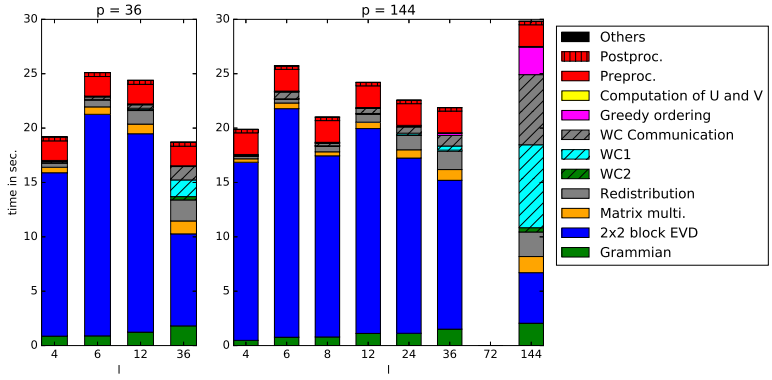

Figure 1: Execution time for the case of $n = 4320$ and $p = 36$ or 144 on the FX10 (before optimization).

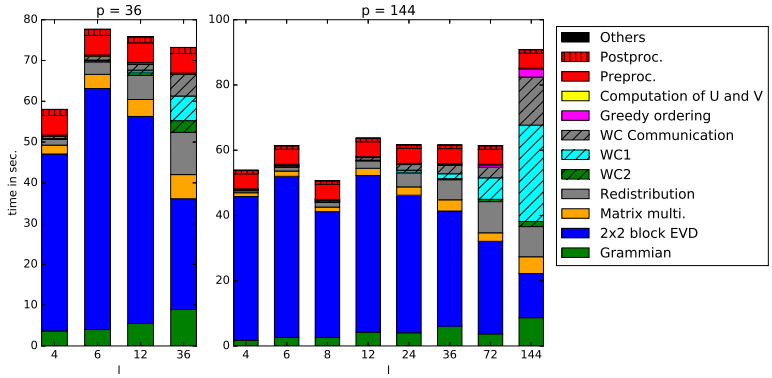

Figure 2: Execution time for the case of $n = 8640$ and $p = 36$ or 144 on the FX10 (before optimization).

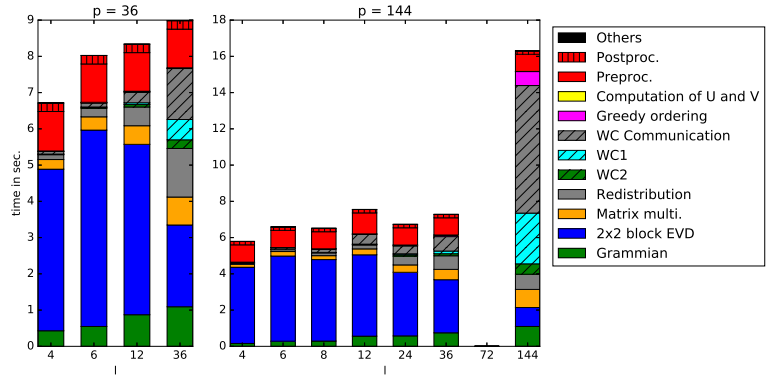

Figure 3: Execution time for the case of $n = 4320$ and $p = 36$ or 144 on the Altix ICE (before optimization).

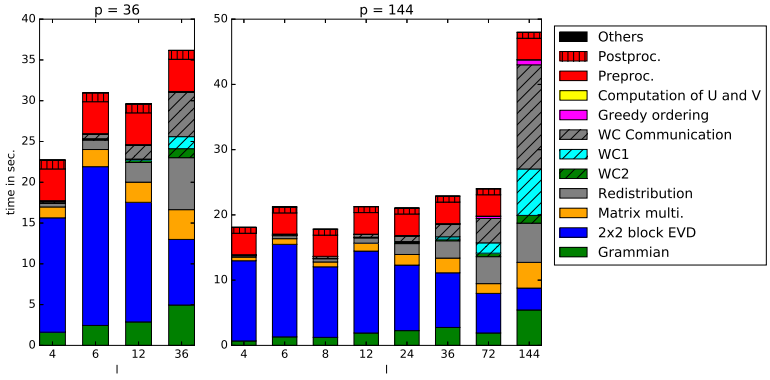

Figure 4: Execution time for the case of $n = 8640$ and $p = 36$ or 144 on the Altix ICE (before optimization).

- The Fujitsu PRIMEHPC FX10 installed at the Information Technology Center of the University of Tokyo. It has 4800 computing nodes connected via a 6-dimensional torus network and each node consists of a SPARC IXfx processor with 16 cores, 32GB of main memory and 12MB of secondary cache.
- The SGI Altix ICE installed at the Supercomputing Center of the Institute for Solid State Physics, The University of Tokyo. It has 1920 computing nodes connected via an enhanced hypercube and each node consists of two Xeon processors with 12 cores, 128GB of main memory and 512KB of secondary cache.

In Figs. 1 and 2, we show the execution times on the FX10 for the test matrices of order 4320 and 8640, respectively. In both cases, the number of nodes p was fixed to 36 or 144 and ℓ varied from 4 to p. The execution times on the Altix ICE are shown in Figs. 3 and 4 for the test matrices of order 4320 and 8640, respectively. Note that on both machines the data for $n = 4320$, $p = 144$ and $\ell = 72$ is missing (see Figs. 1 and 3). This is because an error occurred during execution in the ScaLAPACK routine PDSYEVD[3]. The number of iterations and sweeps (defined as the number of iterations divided by $\ell - 1$) for each case are shown in Table 2. These values depend only on n and ℓ and not on p, as mentioned at the beginning of Section 3.

5 Possible improvements and their effectiveness

5.1 Improvements

Figs. 1 through 4 show that a considerable part of the computation time is spent for two tasks, namely for the weight computation (the sum of three parts, WC1, WC2 and WC Communication) and 2×2 block EVD. This is the same for other cases, which are omitted due to space limitations. As expected from Table 1, the time for WC increases with ℓ, while that for EVD decreases with ℓ. On the other hand, the number of sweeps increases with ℓ, as can be seen from Table 2. The optimal value of ℓ for given n and p are determined from these factors.

To further speed up the OSBJ method, we propose improvements on the implementations of WC and EVD in the following.

Improvement of weight computation In the current implementation [3], the weights $\{w_{ij}\}$ in (4) are computed in the form of dot products. However, it is well known that this operation cannot use the cache memory efficiently. As can be seen from (4), the computation of $\{w_{ij}\}$ can be divided into three parts: computation of $\mathbf{b}_j \equiv A_j^{(r)}\mathbf{e}/\|\mathbf{e}\|_2$ $(1 \le j \le \ell)$, computation of $\mathbf{c}_{ij} = (A_i^{(r)})^\top \mathbf{b}_j$ $(1 \le i < j \le \ell)$ and computation of $w_{ij} = \|\mathbf{c}_{ij}\|$

[3]A close investigation reveals that the error was caused by shortage of the workspace used by PDSYEVD. However, we had provided sufficient workspace specified by ScaLAPACK, so we have not been able to resolve the problem. Note that this error does not occur in the optimized version of our code to be described in Section 5, which computes the 2×2 block on one node without using PDSYEVD.

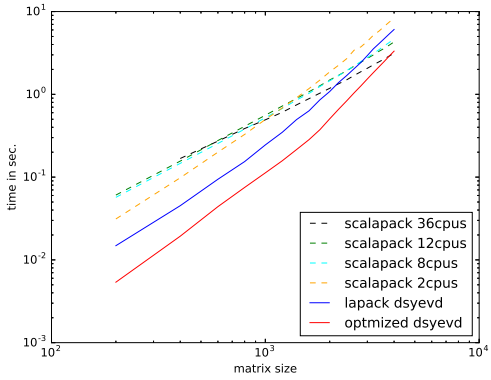

Figure 5: Performance comparison of ScaLAPACK PDSYEVD, LAPACK DSYEVD and our optimized DSYEVD on the FX10.

$(1 \le i < j \le \ell)$. Among them, the second part is the heaviest, since it requires $O(n^2\ell/(2p))$ work per iteration. We therefore propose to aggregate these computations for $1 \le j \le \ell$ and compute them using matrix-matrix multiplication as

$$[\mathbf{c}_{i1}, \mathbf{c}_{i2}, \ldots, \mathbf{c}_{i\ell}] = (A_i^{(r)})^\top [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_\ell] \quad (1 \le i \le \ell). \tag{24}$$

Thus the cache usage of this part is greatly enhanced. In the actual implementation, the matrix multiplication (24) is distributed among several nodes, since each node has only a subset of $\{\mathbf{b}_j\}_{j=1}^{\ell}$.

Improvement of 2×2 block EVD In the current implementation, diagonalization of the Gram matrix G_s is executed by $2k$ nodes in charge of the block column pair $[A_{i_{r,s}}^{(r)}, A_{j_{r,s}}^{(r)}]$ using ScaLAPACK PDSYEVD. However, the tridiagonalization step used in PDSYEVD requires as many interprocessor communications as the order of the input matrix. Hence, when the size $2n/\ell$ of this matrix is small, the portion belonging to communication becomes to be prevailing in the total computational time, thus the performance of PDSYEVD can become lower than that of the sequential routine DSYEVD. Meanwhile, we recently developed an optimized version of DSYEVD for a *single* node of FX10 which adopts a new cache-efficient implementation of the tridiagonalization algorithm, SIMD optimization and a new cache-friendly matrix storage scheme [8]. Thus it might be advantageous to replace PDSYEVD with this routine. To this end, it is required to gather the matrix G_s distributed across $2k$ nodes to one node prior to the EVD, and distribute the eigenvector matrix V_s to the $2k$ nodes after the EVD.

To predict the performance gain to be obtained by this optimization, we compared the performance of ScaLAPACK PDSYEVD, LAPACK DSYEVD and our optimized DSYEVD on the FX10. The latter two routines were executed on one node of the FX10, while PDSYEVD was executed on 2, 8, 12 and 36 nodes. The results are shown in Fig. 5. It is clear from the graph that using ScaLAPACK is not profitable when the matrix size is small, say less than 2000. In fact, in this region our optimized DSYEVD is several times faster than PDSYEVD. Even the standard version of DSYEVD is about two times faster than PDSYEVD. This justifies adopting the improvement on both machines, at least when $2n/\ell$ is not too large.

5.2 Performance evaluation

We incorporated the two improvements proposed in the previous subsection into our parallel OSBJ code [3] and evaluated its performance on the FX10 and Altix ICE. The experimental conditions are the same as those explained in Section 4. The results on the FX10 for the matrices of order 4320 and 8640 are shown in Figs. 6 and 7, respectively. Figs. 8 and 9 show the results on the Altix ICE. It can be seen from the graphs that the times for both WC and EVD have been greatly reduced. Additionally, our experiments show new optimal values for ℓ. On both machines, when $p = 144$, the optimal value is around $\ell = 12$ and 24 for the cases of $n = 4320$ and 8640, respectively. When we compare the total execution time of the original and improved codes for the optimal ℓ, the speedup is 1.8 to 2.0 on the FX10 when using 144 nodes. On the Altix ICE, the corresponding speedup is

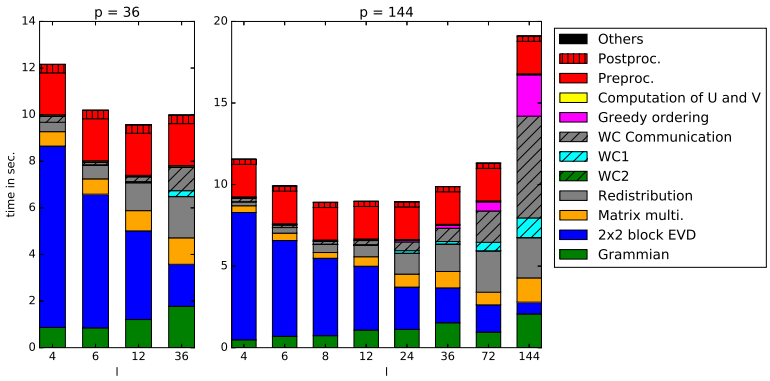

Figure 6: Execution time for the case of $n = 4320$ and $p = 36$ or 144 on the FX10 (after optimization).

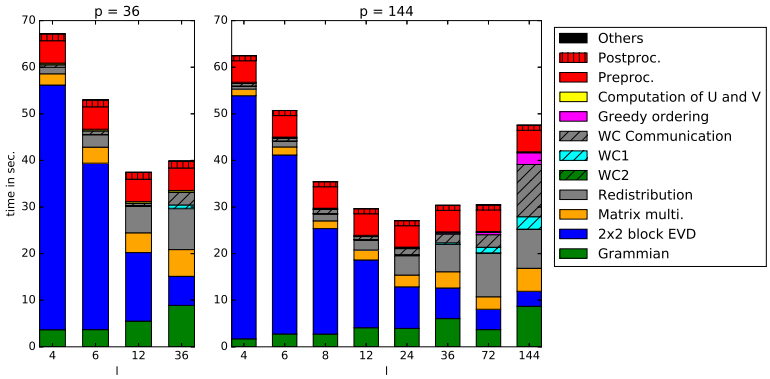

Figure 7: Execution time for the case of $n = 8640$ and $p = 36$ or 144 on the FX10 (after optimization).

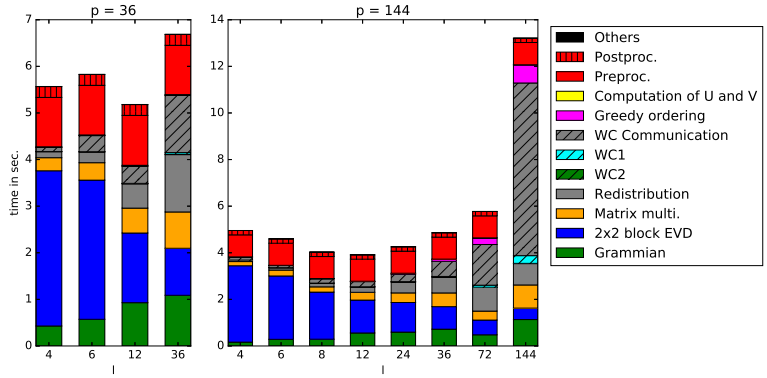

Figure 8: Execution time for the case of $n = 4320$ and $p = 36$ or 144 on the Altix ICE (after optimization).

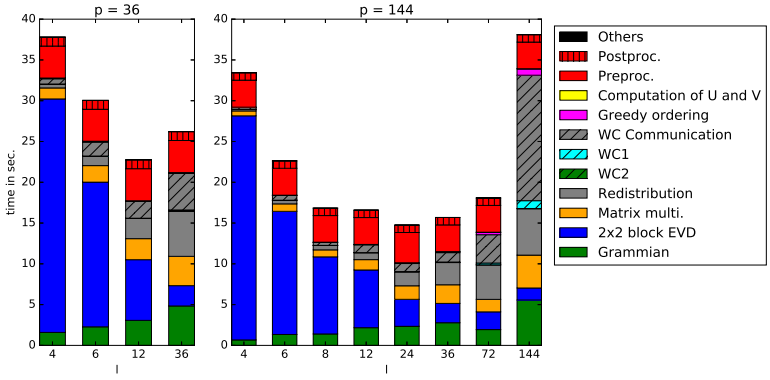

Figure 9: Execution time for the case of $n = 8640$ and $p = 36$ or 144 on the Altix ICE (after optimization).

Table 3: The process grid used for ScaLAPACK PDGESVD.

p	8	12	24	36	72	144
grid	2×4	3×4	4×6	6×6	8×9	12×12

1.2 to 1.5. Thus we can say that the proposed improvements contribute to reducing the total execution time of the SVD considerably.

Finally, we compare the performance of our parallel OSBJ code with that of ScaLAPACK SVD routine PDGESVD. For PDGESVD, we chose the best block size from $\{32, 64, 96, 128\}$ for each case. The process grid was chosen so that it is as close to square as possible under the condition that the number of columns is larger than or equal to the number of rows. The actual grid size is shown in Table 3. The execution times of the OSBJ method and ScaLAPACK PDGESVD for three matrix sizes, $n = 2160$, 4320 and 8640, are shown in Fig. 10 and Fig. 11, respectively, for the FX10 and Altix ICE. On the FX10, the OSBJ method was constantly faster than PDGESVD and the ratio was more than two in most cases. For example, when $n = 8640$ and $p = 72$, the execution time of the OSBJ method and PDGESVD was 31.16 seconds and 120.6 seconds, respectively, so the former was 4 times faster. On the Altix ICE, the OSBJ method was faster than PDGESVD except for the case of $n = 4320$ and $p = 144$.

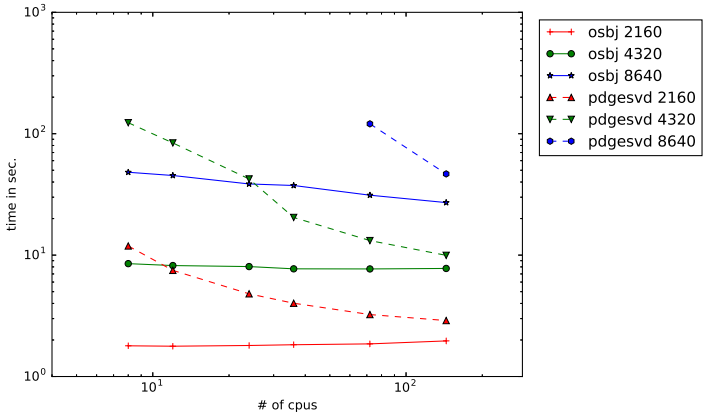

Figure 10: Comparison of the OSBJ method with ScaLAPACK PDGESVD on the FX10.

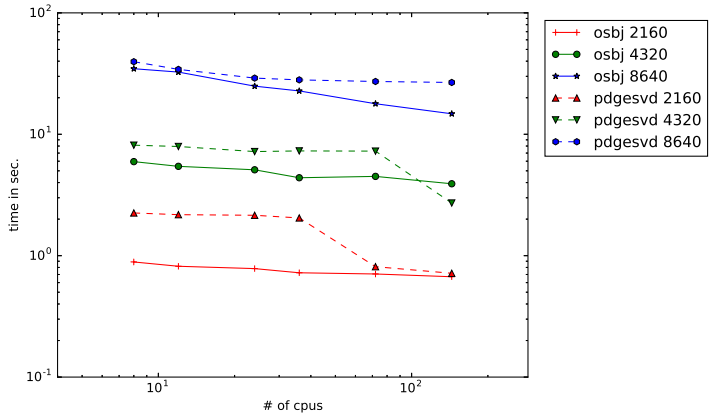

Figure 11: Comparison of the OSBJ method with ScaLAPACK PDGESVD on the Altix ICE.

6 Conclusion

In this paper, we analyzed the performance of the one-sided block Jacobi SVD algorithm with dynamic ordering and variable blocking both theoretically and experimentally. In the theoretical part, we analyzed its convergence behavior and showed that the algorithm has global convergence and asymptotic quadratic convergence properties when the mutual perpendicularity of all pairs of column blocks is computed exactly. In the actual algorithm, an approximate measure of perpendicularity is employed to reduce the computational cost. We also studied its effect on the convergence and showed that the convergence properties are likely to be retained even with the approximate measure. In the experimental part, we evaluated the performance of our one-sided block Jacobi code on the Fujitsu FX10 and SGI Altix ICE parallel computers. We identified two performance bottlenecks, namely, weight computation and 2×2 block EVD, proposed improvements for them and investigated their effectiveness. Thanks to these improvements, the bottlenecks were resolved and the total execution time was reduced by up to 2.0 times and 1.5 times on the FX10 and the Altix ICE, respectively. As a result, our OSBJ solver outperformed ScaLAPACK PDGESVD for almost all the cases when computing the SVD of matrices of order 2160 to 8640 on these machines using 8 to 144 nodes.

Acknowledgment

The authors thank Professor Gabriel Okša at the Mathematical Institute, Slovak Academy of Sciences, for providing valuable comments on the first version on this paper. The present study is supported in part by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (Nos. 26286087, 15H02708, 15H02709) and the Core Research for Evolutional Science and Technology (CREST) Program "Highly Productive, High Performance Application Frameworks for Post Petascale Computing" of Japan Science and Technology Agency (JST). This work is supported also by grant No. 2/0026/14 of the VEGA grant agency.

References

[1] Bečka, M., Okša, G., Vajteršic, M. Dynamic ordering for a parallel block-Jacobi SVD algorithm. *Parallel Computing* 2002; **28**:243–262.

[2] Bečka, M., Okša, G. Parallel one-sided Jacobi SVD algorithm with variable blocking factor. *Lecture Notes in Computer Science* 2014; **8384**:57–66.

[3] Bečka, M., Okša, G., Vajteršic, M. Parallel code for one-sided Jacobi-method. *Technical Report* 2015-02, Department of Computer Sciences, University of Salzburg (April 2015).

[4] Drmač, Z., Veselić, K. New fast and accurate Jacobi SVD algorithm: I. *SIAM J. Matrix Anal. Appl.* 2007; **29**:1322–1342.

[5] Golub, G. H., van Loan, C. F. *Matrix Computations*. 4th Ed. Johns Hopkins Univ. Press, 2012.

[6] Hyvärinen, A., Karhunen, J., Oja, E. *Independent Component Analysis*. John Wiley & Sons, 2001.

[7] Krantz, S. G, Parks, H. R. *Geometric Integration Theory*. Birkhäuser, Basel, 2008.

[8] Kudo, S., Yamamoto, Y., Yokokawa, M. Performance analysis and implementation method of Dongarra-Wilkinson tridiagonalization method. *Proceedings of HPCS2015* (in Japanese).

[9] Okša, G., Vajteršic, M. Efficient preprocessing in the parallel block-Jacobi SVD algorithm. *Parallel Computing* 2005; **31**:166–176.

[10] ScaLAPACK test routine pdlagge:
http://www.netlib.org/scalapack/explore-html/d7/da2/pdlagge_8f_source.html.

[11] Schönhage, A. Zur Konvergenz des Jacobi-Verfahrens. *Numerische Mathematik* 1961; **3**:374–380.

[12] Schönhage, A. Zur quadratischen Konvergenz des Jacobi-Verfahrens. *Numerische Mathematik* 1964; **6**:410–412.

[13] Strang, G. *Computational Science and Engineering*, Wellesley-Cambridge Press, 2007.

[14] Toledo, S., Rabani, E. Very large electronic structure calculations using an out-of-core filter-diagonalization method. *Journal of Computational Physics* 2002; **180**:256–269.

[15] Yamamoto, Y., Zhang, L., Kudo, S. Convergence analysis of the parallel classical block Jacobi method for the symmetric eigenvalue problem. *JSIAM Letters* 2014; **6**:57–60.