Proceedings of ALGORITMY 2012
pp. 132–140

PARALLEL ONE-SIDED BLOCK-JACOBI SVD ALGORITHM[*]

MARTIN BEČKA[†], GABRIEL OKŠA[‡], AND MARIÁN VAJTERŠIC[§]

Abstract. A new dynamic ordering is presented for the parallel one-sided block Jacobi SVD algorithm. Similarly to the two-sided variant, which has been analyzed and implemented in last 10 years, the dynamic ordering takes into account the actual status of a matrix—this time of its block columns with respect to their mutual orthogonality. Using p processors, in each parallel iteration step the p mostly inclined pairs of block columns are made orthogonal, whereby their inclination is measured by an estimation of principal angles between subspaces generated by those block columns. It is shown that principal angles can be estimated using a set of parallel Lanczos processes applied to special Wielandt-Jordan matrices. Only a limited number of iteration steps in each Lanczos process is needed for estimating a small number of smallest principal angles. Numerical experiments show that the proposed new parallel dynamic ordering can substantially decrease the number of parallel iteration steps needed for the convergence when compared to a parallel cyclic ordering. However, its more scalable implementation is desirable because currently it occupies a relatively high portion of the total parallel execution time.

Key words. Dynamic ordering, one-sided block-Jacobi method, Message Passing Interface

AMS subject classifications. 15A18, 15A23, 68W10

1. Background.

This section contains a very brief introduction into the mathematical background behind the Singular Value Decomposition (SVD) of a matrix. Full SVD theory can be found in many excellent books, e.g. [8, 13, 24]. Afterwards, some serial algorithms for the SVD computation, other than the Jacobi method, are briefly mentioned. We concentrate on dense matrices, so some projection methods that are well-suited for sparse matrices are omitted.

In what follows, A^H denotes the Hermitian operation over the elements of matrix A, i.e., their complex conjugation and transposition. Further, $\|A\|_F$ and $\|A\|_2$ are the Frobenius and spectral norms of matrix A, respectively.

1.1. Singular value decomposition.

The SVD of a complex matrix A of size $m \times n$, $(m \geq n)$, is defined by

$$A = U \Sigma V^H, \tag{1.1}$$

where U and V are unitary matrices of orders m and n, respectively, and Σ is an $m \times n$ diagonal matrix. The real, nonnegative diagonal elements $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$ of Σ are the singular values of A, and the columns of U and V are the left and right singular vectors, respectively. When $m > n$, the matrix Σ contains the zero block of size $(m-n) \times n$ at the bottom.

The decomposition $A = U\Sigma V^H$ can be also written as $AV = U\Sigma$ or $Av_i = \sigma_i u_i$ for $i = 1, 2, \ldots, n$. The alternative way of saying the same thing is $A^H U = V \Sigma^H$ or $A^H u_i = \sigma_i v_i$ for

[*]The authors were supported by the VEGA grant no. 2/0003/11 from the Scientific Grant Agency of the Ministry of Education and Slovak Academy of Sciences, Slovakia.

[†]Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic, (Martin.Becka@savba.sk).

[‡]Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic, (Gabriel.Oksa@savba.sk).

[§]Department of Computer Sciences, University of Salzburg, Salzburg, Austria, and Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic, (marian@cosy.sbg.ac.at).

$i = 1,2,\ldots,n$ and $A^H u_i = 0$ for $i = n+1,\ldots,m$. When $m > n$, the so called *thin* (or *economy-sized*) SVD is often computed in the form $A = U_n \Sigma_n V^H$, where $U_n = [u_1, u_2, \ldots, u_n]$ (i.e., only first n left singular vectors are computed), and $\Sigma_n = \mathrm{diag}(\sigma_1,\ldots,\sigma_n)$.

If $\mathrm{rank}(A) = r$ with $r < n$ then last $n-r$ singular values are zero and $A = U_r \Sigma_r V_r^H$ where $V_r = [v_1, v_2, \ldots, v_r]$. This is the so-called *compact* SVD of A where only first r left and right singular vectors play a role.

Taking only first t, $t < r$, left and right singular vectors and singular values, one obtains the rank-t approximation $A_t = U_t \Sigma_t V_t^H$, which is called the *rank-t truncated (partial)* SVD of A. Among all rank-t matrices B, $B = A_t$ is the unique minimizer of $\|A-B\|_F$. The truncated SVD is much smaller to store and cheaper to compute than the compact SVD when $t \ll r$ and it is the most often form of the SVD in applications where the small singular values are of no interest to the user (e.g. in signal and image filtration, data retrieval computations, etc.).

2. One-sided block-Jacobi SVD algorithm.

The one-sided block-Jacobi SVD algorithm is suited for the SVD computation of a general complex matrix A of order $m \times n$, $m \geq n$. However, we will restrict ourselves to real matrices with obvious modifications in the complex case.

We start with the block-column partitioning of A in the form

$$A = [A_1, A_2, \ldots, A_\ell],$$

where the width of A_i is n_i, $1 \leq i \leq \ell$, so that $n_1 + n_2 + \cdots + n_\ell = n$.

The serial algorithm can be written as an iterative process:

$$\begin{aligned} A^{(0)} &= A, \quad V^{(0)} = I_n, \\ A^{(k+1)} &= A^{(k)} U^{(k)}, \quad V^{(k+1)} = V^{(k)} U^{(k)}, \quad k \geq 0. \end{aligned} \tag{2.1}$$

Here the $n \times n$ orthogonal matrix $U^{(k)}$ is the so-called *block rotation* of the form

$$U^{(k)} = \begin{pmatrix} I & & & & \\ & U_{ii}^{(k)} & & U_{ij}^{(k)} & \\ & & I & & \\ & U_{ji}^{(k)} & & U_{jj}^{(k)} & \\ & & & & I \end{pmatrix}, \tag{2.2}$$

where the unidentified matrix blocks are zero. The purpose of matrix multiplication $A^{(k)}U^{(k)}$ in (2.1) is to mutually orthogonalize the columns between column-blocks i and j of $A^{(k)}$. The matrix blocks $U_{ii}^{(k)}$ and $U_{jj}^{(k)}$ are square of order n_i and n_j, respectively, while the first, middle and last identity matrix is of order $\sum_{s=1}^{i-1} n_s$, $\sum_{s=i+1}^{j-1} n_s$ and $\sum_{s=j+1}^{r} n_s$, respectively. The orthogonal matrix

$$\hat{U}^{(k)} = \begin{pmatrix} U_{ii}^{(k)} & U_{ij}^{(k)} \\ U_{ji}^{(k)} & U_{jj}^{(k)} \end{pmatrix} \tag{2.3}$$

of order $n_i + n_j$ is called the *pivot submatrix* of $U^{(k)}$ at step k. During the iterative process (2.1), two index functions are defined: $i = i(k)$, $j = j(k)$ whereby $1 \leq i < j \leq r$. At each step k, the pivot pair (i,j) is chosen according to a given *pivot strategy* that can be identified with a function $\mathscr{F}: \{0,1,\ldots\} \to \mathbf{P}_r = \{(l,m) : 1 \leq l < m \leq r\}$. If $\mathbf{O} = \{(l_1,m_1),(l_2,m_2),\ldots,(l_{N(\ell)},m_{N(\ell)})\}$ is some ordering of $\mathbf{P}_\ell$ with $N(\ell) = \ell(\ell-1)/2$, then the *cyclic* strategy is defined by:

If $k \equiv \ell - 1 \mod N(\ell)$ then $(i(k), j(k)) = (l_s, m_s)$ for $1 \le s \le N(\ell)$.

The most common cyclic strategies are the *row-cyclic* one and the *column-cyclic* one, where the orderings are given row-wise and column-wise, respectively, with regard to the upper triangle of A. The first $N(\ell)$ iterations constitute the first *sweep*. When the first sweep is completed, the pivot pairs (i, j) are repeated during the second sweep, and so on, up to the convergence of the entire algorithm.

Notice that in (2.1) only the matrix of right singular vectors $V^{(k)}$ is iteratively computed by orthogonal updates. If the process ends at iteration t, say, then $A^{(t)}$ has mutually highly orthogonal columns. Their norms are the singular values of A, and the normalized columns (with unit 2-norm) constitute the matrix of left singular vectors.

The parallel version of the one-sided block-Jacobi SVD algorithm implemented on p processors with the blocking factor $\ell = 2p$ is given below.

Parallel one-sided block-Jacobi SVD algorithm

```
1:  $V = I_n,\ \ell = 2 * p$
2:  $\triangleright$ each processor has 2 block columns of $A$ : $A_L$ and $A_R$
3:  $G = \begin{pmatrix} G_{LL} & G_{LR} \\ G_{LR}^T & G_{RR} \end{pmatrix} = \begin{pmatrix} A_L^T A_L & A_L^T A_R \\ A_R^T A_L & A_R^T A_R \end{pmatrix}$
4:  $\triangleright$ *global convergence criterion with a constant* $\varepsilon$, $0 < \varepsilon \ll 1$
5:  **while** $(F(A, \ell) \ge \varepsilon)$ **do**
6:      $\triangleright$ *local convergence criterion with a constant* $\delta$, $0 < \delta \ll 1$
7:      **if** $(F(G, \ell) \ge \delta)$ **then**
8:          $\triangleright$ *diagonalization of G*
9:          EVD($G, X$)
10:         $\triangleright$ *update of block columns*
11:         $(A_L, A_R) = (A_L, A_R) * X$
12:         $(V_L, V_R) = (V_L, V_R) * X$
13:     **end if**
14:     $\triangleright$ *parallel ordering–choice of p independent pairs* $(i, j)$ *of block columns*
15:     ReOrderingComp($p$)
16:     Send-Receive($A_k, V_k, G_{kk}$), where $k$ is either $L$ or $R$
17: **end while**
18: $sv_L$ : square roots of diagonal elements of $G_{LL}$
19: $sv_R$ : square roots of diagonal elements of $G_{RR}$
20: $\triangleright$ *two block columns of left singular vectors*
21: $U_L = A_L * \mathrm{diag}(1/sv_L),\ U_R = A_R * \mathrm{diag}(1/sv_R)$
```

Note that the diagonalization of the auxiliary matrix G is equivalent to the mutual orthogonalization of block columns A_L and A_R of matrix A. Some parallel ordering is required in the procedure ReOrderingComp that defines p independent pairs of block columns of A which are simultaneously mutually orthogonalized in a given parallel iteration step by computing p eigenvalue decompositions EVD(G, X) of p auxiliary matrices G. Up to now, some cyclic (static) parallel ordering (see [1, 2]) has been used. In next subsection, we describe a new *dynamic* ordering that takes into account the actual status of matrix A with respect to the mutual inclination of its block columns.

2.1. Dynamic ordering. A big disadvantage of any fixed ordering is the fact that the actual status of orthogonality is usually checked only after a whole sweep and one has no information about the quality of this process at the beginning of a parallel iteration step. In other words, in a given parallel iteration step one can try to orthogonalize some mutually 'almost orthogonal' block columns while neglecting pairs that are far from being orthogonal. It is clear, at least intuitively, that orthogonalizing block columns with small mutual angles first would mean to eliminate the 'worst' pairs first, and this would mean (hopefully) the faster convergence of the whole algorithm as compared with any fixed, cyclic ordering.

Hence, the main question is how to choose p pairs of block columns with smallest principal angles among all $\ell(\ell-1)/2 = p(2p-1)$ pairs. The obvious, but very naive way is to compute, for each column block X, all possible matrix products X^TY, then to compute the SVD of X^TY and look at the singular values, which are the cosines of acute principal angles (the smaller angle, the larger cosine). When the block columns are distributed in processors, to compute matrix products X^TY for each two different block columns X and Y means to move block columns across processors, i.e., it leads to heavy communication at the beginning of each parallel iteration step. Besides that, one needs to compute many matrix products and SVDs. Moreover, when p pairs of column blocks with smallest principal angles are chosen, they must meet in processors, which means yet another communication.

Our idea is different. After the first parallel iteration step, the block columns inside contain mutually orthogonal columns. Suppose that each processor contains exactly two block columns (this is not substantial for the following discussion). Moreover, suppose that $k \equiv n/2p$ columns in each block column are *normalized* so that each has the unit Euclidean norm. Hence, each column block is the *orthonormal basis* of the k-dimensional subspace which is spanned by the column vectors of a given block column.

Consider now the block column partitioning $A = [A_1, A_2, \ldots, A_\ell]$ of matrix A. Take two block columns A_i, A_j which should be orthogonalized in a given parallel iteration step. Having p processors, our goal is to choose p pairs of those block columns that are maximally *inclined* to each other, i.e., their mutual position differs maximally from the orthogonal one. Mathematically, this goal can be described by using the notion of *principal angles* between two k-dimensional subspaces spanned by two block columns A_i, A_j. Since A_i and A_j are orthonormal bases of two subspaces with the equal dimension, the cosines of principal angles are defined as the singular values of the matrix X^TY. Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k$ be k singular values of the $k \times k$ matrix $A_i^TA_j$. Then the principal angles $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_k$, $\theta_i \in [0, \pi/2]$, $1 \leq i \leq k$, are defined as:

$$\theta_i = \arccos(\sigma_i), \ 1 \leq i \leq k. \tag{2.4}$$

Since A_i and A_j have orthonormal columns, all singular values of $A_i^TA_j$ are in the interval $[0,1]$, so that the relation (2.4) is well defined.

We are interested in, say, L *smallest* principal angles, i.e., in L *largest* cosines (largest singular values) $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_L$. When $\sigma_1 = 0$, then all $\sigma_i = 0$, $2 \leq i \leq k$, and two block columns X and Y are perfectly orthogonal; we do not need to orthogonalize them explicitly. On the other hand, when all σ_k are significantly greater than 0, column blocks X and Y are certainly far from the mutual orthogonality.

However, this approach means that we must explicitly compute the matrix $A_i^TA_j$. When two block columns A_i and A_j are placed in two different processors, we can either compute this matrix product in parallel (but for each pair of block columns), or store both blocks in one processor and compute the matrix product locally using the LAPACK library. Afterwards, we must compute (or at least somehow estimate) the largest L singular values and afterwards compute some function of them (e.g., the sum of their squares) to get our weight w_{ij} for the maximal perfect matching. In both cases we need again too much communication at the beginning of each parallel iteration step to construct the actual parallel ordering for that step.

To estimate L largest singular value of the $k \times k$ matrix $A_i^T A_j$, we suggest to use the Lanczos process applied to the symmetric Jordan-Wielandt matrix C,

$$C \equiv \begin{pmatrix} 0 & A_i^T A_j \\ A_j^T A_i & 0 \end{pmatrix}. \tag{2.5}$$

It is well known that the eigenvalues of the $2k \times 2k$ matrix C are $\pm\sigma_1, \pm\sigma_2, \ldots, \pm\sigma_k$. Notice that there are k pairs of eigenvalues with the same absolute value.

It follows from the theory of Krylov space methods that the Lanczos algorithm applied to a symmetric matrix is the good iterative method for estimating its largest (in absolute value) eigenvalues. This algorithm, applied to the symmetric Jordan-Wielandt matrix C, is listed below for a *fixed* number of iteration steps L.

Lanczos algorithm for the symmetric Jordan-Wielandt matrix C

```
1: Choose integer $L = 2s$ and the vector $x_0$ of length $2k$, and compute
   $\beta_1 = \|x_0\|$; $v_1 = x_0/\beta$
2: **for** $s = 1$ to $L$ **do**
3:    $w_s = Cv_s$
4:    **if** $(s \neq 1)$ **then**
5:       $w_s = w_s - \beta_s v_{s-1}$
6:    **end if**
7:    $\alpha_s = w_s^T v_s$
8:    $w_s = w_s - \alpha_s v_s$
9:    $\beta_{s+1} = \|w_s\|$
10:   **if** $(\beta_{s+1} \neq 0)$ **then**
11:      $v_{s+1} = w_s/\beta_{s+1}$
12:   **end if**
13:   **if** $(\beta_{s+1} == 0)$ **then**
14:      $s = L$
15:   **end if**
16: **end for**
17: Set: $T_L = \text{tridiag}(\beta_i, \alpha_i, \beta_{i+1})$
18: Compute the Frobenius norm of $T_L$.
```

Steps 2–9 constitute an adaptation of the Arnoldi method for a symmetric matrix. Due to the special structure of C (see Eq. (2.5)), the matrix-vector product in step 2 is applied in two substeps: $w_s^1 = A_i^T A_j v_s^1$, $w_s^2 = A_j^T A_i v_s^2$, where $v_s = (v_s^{1T}, v_s^{2T})^T$ and $w_s = (w_s^{1T}, w_s^{2T})^T$.

The result is the orthonormal basis of the Krylov subspace $\mathscr{K}_L(C, x_0)$ formed by vectors v_s, $1 \leq s \leq L$. Besides that, the coefficients α_s and β_s are computed that are stored in the symmetric, tri-diagonal matrix T_L (step 10).

In our application, the orthonormal vectors v_s are not important (they are used, for example, in the solution of a linear system of equations). What is most important, is the square of the Frobenius norm of T_L written in terms of its eigenvalues ω_s, $1 \leq s \leq L$ (they are known as Ritz values):

$$\|T_L\|_F^2 = \sum_{s=1}^{L} \omega_s^2.$$

As already mentioned, the L Ritz values approximate reasonably well L largest (in the absolute value) eigenvalues λ_s of the Jordan-Wielandt matrix C. However, in our application, there are exactly two eigenvalues of C with the same absolute value (with opposite signs) and they are related to the squares of singular values of $A_i^T A_j$. Therefore,

$$\|T_m\|_F^2 = \sum_{s=1}^{L} \omega_s^2 \approx \sum_{s=1}^{L} \lambda_s^2 = 2\sum_{s=1}^{L/2} \sigma_s^2 = 2\sum_{s=1}^{L/2} \cos^2(\theta_s),$$

i.e., the Frobenius norm of T_L can be used as the (good) approximation for the sum of $L/2$ largest cosines defining $L/2$ *smallest* principal angles between subspaces span(A_i) and span(A_j). In other words, we have found an easily computable weight w_{ij} for the maximum perfect matching in the one-sided block-Jacobi method. We stress that we do *not* need to compute the Ritz values (i.e., the EVD of T_L) - the Frobenius norm squared is enough.

Moreover, note that in our application T_L is not needed in its explicit form. All that is needed is the square of its Frobenius norm. Since

$$w_{ij} = \|T_L\|_F^2 = \sum_{s=1}^{L} \alpha_s^2 + 2\sum_{s=2}^{L} \beta_s^2,$$

$\|T_L\|_F^2$ can be updated recursively immediately after computing α_s and β_{s+1} in the sth iteration step of the Lanczos algorithm.

Note that the weight w_{ij} takes into account the *actual* mutual position of two subspaces span(A_i) and span(A_j). Therefore, we can simply choose the 'worst' p pairs of column blocks for their parallel orthogonalization by choosing the pairs with highest values of w_{ij}. This is an analogy to the two-sided dynamic ordering where the actual Frobenius norm of the off-diagonal blocks was taken into account. Therefore, the above described ordering can be defined as the *one-sided dynamic ordering*. To choose the p 'worst' block columns for the parallel orthogonalization, the same maximum-weight perfect matching algorithm on the complete graph with r vertices and weights w_{ij} can be used as in the two-sided case (see [3]).

We have just described, how we can quite cheaply compute the weight w_{ij} that is the function of $L/2$ (estimated) largest cosines of principal angles between subspaces span(A_i) and span(A_j). The larger the weight, the lower the degree of mutual orthogonality between these two subspaces. However, at the beginning of each parallel iteration step we have to compute those weights for *all* pairs of block columns of matrix A. Next we describe how this computation can be done in parallel *without* sending/receiving whole block columns and *without* computing explicitly the matrix products $A_i^T A_j$.

In a parallel environment with p processors and the blocking factor $\ell = 2p$, these computations must be done for all $2(p-1)$ Lanczos processes for which each processor P_j is the master and this work is serialized inside processors. Each processor stores the information about two block columns that it currently overviews stores, and about all Lanczos processes for which it serves as the master. Therefore, each processor can read/write from/to the data structure the data/results of its own computations for all Lanczos processes for which it is the master (matrix-vector products, updates of Frobenius norms). To communicate data between all processors, the MPI collective communication `ALLTOALL` is used. Two such communications are needed per one parallel iteration step, i.e., together $2L$ collective communications are needed. These communications serve also like the global synchronization steps in the whole computation.

At the end of computation with Lanczos processes, all processors contain all weights w_{ij} for all block column pairs (excluding those residing in p processors), which are simply the squares of Frobenius norms of all matrices T_L produced in all Lanczos processes. Therefore, each processor can compute the maximum-weight perfect matching and the resulting parallel ordering; the algorithm is the same as for the parallel two-sided block-Jacobi method (see [3]). For transferring the chosen pairs in processors, the optimal parallel scheduling is used (see [4]).

The global stopping criterion of the iteration process is based on the maximum value of currently computed weights w_{ij}. When using a computer with machine precision ε, the convergence is reached when

$$\max_{i,j} w_{ij} < nL\varepsilon, \tag{2.6}$$

where n is the matrix order and L is the number of steps in Lanczos processes. In other words, the computation is finished when the cosines of $L/2$ largest principal angles between all column blocks are 'sufficiently' small. The local stopping criterion is similar: A given pair (i, j) of block columns is *not* orthogonalized when

$$w_{ij} < nL\varepsilon. \tag{2.7}$$

In following tables, we present first numerical results comparing the behavior of the parallel one-sided block-Jacobi SVD algorithm with dynamic ordering with two different cyclic (static) orderings, `static1` (the odd-even ordering CO(0), see [1]) and `static2` (the robin-round ordering DO(0), see [1]). Computations were performed on the Woodcrest Cluster at Nuernberg-Erlangen University for random matrices with six various distributions of SVs defined by the variable `mode`. `mode` $= 1$ corresponds to a multiple minimal singular value, `mode` $= 2$ to a multiple maximal SV, `mode` $= 3$ describes a geometric sequence of SVs, `mode` $= 4$ defines an arithmetic sequence of SVs, `mode` $= 5$ defines the SVs as random numbers such that their logarithms are uniformly distributed, and, finally, `mode` $= 6$ sets the SVs to random numbers from the same distribution as the rest of a matrix (i.e., in our case they were normally distributed).

Table 2.1 contains the results for the SVD of well-conditioned matrices (with the condition number $\kappa = 10^1$) of order $n = 4000$ with a variable number of Lanczos steps L. For both static cyclic orderings, the number of sweeps is given by $n_{\text{it}}/15$ where n_{it} is the number of parallel iterations needed for the convergence of the whole algorithm. The total parallel execution time T_{p} is given in seconds. For `mode` $= 1$ and 2, our dynamic ordering needs about *five times less* parallel iterations than a static ordering. For harder cases, with `mode` ≥ 3, the ratio is about $2-3$. But notice, that the decrease of T_{p} is much less. The dynamic ordering is about 2.5 times faster for `mode` $= 1$ and 2, but only about 1.5 faster for other modes. Also, T_{p} increases with L, the number of Lanczos steps, suggesting that the estimation of weights at the beginning of each parallel iteration step is quite time-demanding.

This conclusion is confirmed in Table 2.2 with results for ill-conditioned matrices (with $\kappa = 10^8$) where the last row depicts the average time T_{WC} of weight computations for a given number of Lanzcos steps for `mode` $= 5$. With respect to n_{it}, the situation is similar to well-conditioned matrices. However, it is clearly seen that our current implementation of the dynamic ordering is not very efficient. For example, in the case of $L = 6$ Lanczos steps the time spent in the computation of weights is 60 per cent of T_{p}. If this portion of algorithm were faster, one would substantially decrease T_{p} and be even more efficient as compared to the static ordering.

3. Conclusions. Recent progress in the parallel block-Jacobi SVD algorithm has been achieved by applying two ideas: i) the new parallel dynamic ordering of subproblems, and ii)

TABLE 2.1
Performance for $n = 4000,\ p = 8,\ \kappa = 10^1$

mode		$L=1$	$L=2$	$L=4$	$L=6$	static1	static2
1	n_{it}	4	4	4	4	30	30
	T_{p} [s]	5	6	9	11	13	13
2	n_{it}	4	4	4	4	30	30
	T_{p} [s]	5	6	9	11	13	12
3	n_{it}	108	99	98	99	240	270
	T_{p} [s]	225	234	292	354	354	390
4	n_{it}	103	97	98	97	225	240
	T_{p} [s]	212	228	289	345	327	354
5	n_{it}	109	103	99	100	255	285
	T_{p} [s]	230	242	293	360	367	409
6	n_{it}	108	107	106	103	270	285
	T_{p} [s]	226	252	315	371	395	420

TABLE 2.2
Performance for $n = 4000,\ p = 8,\ \kappa = 10^8$

mode		$L=1$	$L=2$	$L=4$	$L=6$	static1	static2
2	n_{it}	19	19	19	19	45	45
	T_{p} [s]	21	26	38	50	25	24
3	n_{it}	226	205	169	183	780	795
	T_{p} [s]	515	535	552	696	1233	1252
4	n_{it}	111	105	103	101	240	270
	T_{p} [s]	225	241	303	358	347	390
5	n_{it}	219	208	184	177	795	795
	T_{p} [s]	501	538	597	688	1243	1266
	T_{WC} [s]	159	250	345	413		
6	n_{it}	219	208	184	177	795	795
	T_{p} [s]	501	538	597	688	1243	1266

the matrix pre-processing by QR iterations. For the parallel two-sided block-Jacobi method, these ideas were implemented and tested on various parallel platforms during last 10 years and results were published in papers [3, 4, 5, 22].

In this paper, we present a new dynamic ordering of subproblems for the parallel one-sided variant. First numerical results are quite promising, but a more efficient implementation of the estimation of principal angles between any two block matrix columns is needed. In other words, one should spend much less portion of the total parallel execution time in the computation and distribution of weights for the dynamic ordering.

However, the new dynamic ordering alone can not make the parallel one-sided block-Jacobi SVD algorithm competitive to the ScaLAPACK routine `PDGESVD`. Again, some sort of matrix pre-processing has to be included similarly as it was the case in the two-sided variant [22]. A concentration of the Frobenius norm near the main matrix diagonal is not enough. It has to be coupled with a special ordering inside EVDs of 2×2 subproblems computed in each processor within a given parallel iteration step (see [11, 12]). We plan to investigate and implement these ideas in the near future.

REFERENCES

[1] M. Bečka and M. Vajteršic, Block-Jacobi SVD algorithms for distributed memory systems: I. Hypercubes and rings, Parallel Algorithms Appl. 13 (1999) 265–287.

[2] M. Bečka and M. Vajteršic, Block-Jacobi SVD algorithms for distributed memory systems: II. Meshes, Parallel Algorithms Appl. 14 (1999) 37–56.

[3] M. Bečka, G. Okša and M. Vajteršic, Dynamic ordering for a parallel block-Jacobi SVD algorithm, Parallel Computing 28 (2002) 243–262.

[4] M. Bečka and G. Okša, On variable blocking factor in a parallel dynamic block-Jacobi SVD algorithm, Parallel Computing 28 (2003) 1153–1174.

[5] M. Bečka, G. Okša, M. Vajteršic and L. Grigori, On iterative QR pre-processing in the parallel block-Jacobi SVD algorithm, Parallel Computing 36 (2010) 297–307.

[6] J. Demmel and W. Kahan, Accurate singular values of bidiagonal matrices, SIAM J. Sci. Statist. Comp. 11 (1990) 873–912.

[7] J. Demmel and K. Veselić, Jacobi's method is more accurate than QR, SIAM J. Matrix Anal. Appl. 13 (1992) 1204–1245.

[8] J. Demmel, Applied numerical linear algebra, First ed., SIAM, Philadelphia, 1997.

[9] Z. Drmač, Implementation of Jacobi rotations for accurate singular value computation in floating-point arithmetic, SIAM J. Sci. Comp. 18 (1997) 1200–1222.

[10] Z. Drmač, A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm, IMA J. Numer. Anal. 19 (1999) 191–213.

[11] Z. Drmač and K. Veselić, New fast and accurate Jacobi SVD algorithm: I., LAPACK Working Note 169, August 2005.

[12] Z. Drmač and K. Veselić, New fast and accurate Jacobi SVD algorithm: II., LAPACK Working Note 170, August 2005.

[13] G. H. Golub and C. F. Van Loan, Matrix computations, Third ed., The Johns Hopkins University Press, Baltimore, 1996.

[14] V. Hari and J. Matejaš, Accuracy of the Kogbetliantz method, preprint, University of Zagreb, 2005.

[15] V. Hari and V. Zadelj-Martič, Parallelizing Kogbetliantz method, accepted for publication at Int. Conf. on Numerical Analysis and Scientific Computation, Rhodos, Greece, September 2006.

[16] V. Hari, Accelerating the SVD block-Jacobi method, Computing 75 (2005) 27–53.

[17] V. Hari, Convergence of a block-oriented quasi-cyclic Jacobi method, accepted for publication in SIAM J. Matrix Anal. Appl.

[18] M. R. Hestenes, Inversion of matrices by biorthogonalization and related results, J. SIAM 6 (1958) 51–90.

[19] N. J. Higham, Accuracy and stability of numerical algorithms, First ed., SIAM, Philadelphia, 1996.

[20] C. G. J. Jacobi, Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen, Crelle's Journal für Reine und Angewandte Mathematik 30 (1846) 51–95.

[21] MPI Programming Standard 2.0, http://www.mcs.anl.gov

[22] G. Okša and M. Vajteršic, Efficient preprocessing in the parallel block-Jacobi SVD algorithm, Parallel Computing 31 (2005) 166–176.

[23] P. M. de Rijk, A one-sided Jacobi algorithm for computing the singular value decomposition on a vector computer, SIAM J. Sci. Stat. Comp. 10 (1989) 359–371.

[24] G. W. Stewart, Matrix algorithms, Vol. II: Eigensystems, First ed., SIAM, Philadelphia, 2001.

[25] K. Veselić and V. Hari, A note on a one-sided Jacobi algorithm, Numer. Math. 56 (1989) 627-633.