UNIVERSITÄT SALZBURG

Parallel Code for One-sided Jacobi-Method

Martin Bečka Gabriel Okša Marián Vajteršic

Technical Report 2015-02 April 2015

Department of Computer Sciences

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

Technical Report Series

Parallel Code for One-sided Jacobi-Method

Martin Bečka* Gabriel Okša* Marián Vajteršic†

February 27, 2015

Abstract. *One sided block Jacobi algorithm for the singular value decomposition (SVD) of matrix can be a method of choice to compute SVD efficiently and accurately in parallel. A given matrix is logically partitioned into block columns and is subjected to an iteration process. In each iteration step, for given two block columns, their Gram matrix is generated, its symmetric eigenvalue decomposition (EVD) is computed and the update of block columns by matrix-matrix multiplication is performed. Another possibility is to omit the generation of Gram matrix and the update, so that there is no matrix-matrix multiplication at all. A local matrix is formed by the two block columns. We can compute its QR decomposition first to reduce the dimension and decrease the off-diagonal norm, and the one-sided serial Jacobi SVD is called (either for a local matrix or its R-factor). Since the matrix coming out from the one-sided Jacobi SVD algorithm is the same as the original matrix after update, we are done. Crucial for this new approach is an efficient implementation of the QR decomposition for tall and skinny matrices as well as a fast and accurate (serial) one-sided Jacobi SVD algorithm. Another improvement of the algorithm would be to replace the static (fixed) local stopping criterion in the inner EVD or SVD computations by the dynamic (flexible) one to reduce the work done by these routines in one parallel iteration step. Since the orthogonality of columns is crucial at the end of the algorithm, one could progressively set the local stopping criterion as the computation proceeds. We tried to implement the proposed approaches and compare the achieved results with the standard algorithm.*

1 Introduction

The one-sided Jacobi methods for computing the singular value decomposition (SVD) of a rectangular matrix are more efficient than their two-sided counterparts mainly due to the halving of the number of matrix-matrix products needed for updating the left and right singular vectors; see [1, 22, 23] Moreover, some of them were also proved to be accurate in the relative sense and at least as accurate as the two-sided Jacobi methods; see [6, 7, 8, 9, 10, 23]. When the preprocessing for positive definite matrices is used in form of the Cholesky decomposition,

*Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic, email: Martin.Becka@savba.sk, Gabriel.Oksa@savba.sk

†Department of Computer Sciences, University of Salzburg, Austria, email: marian@cosy.sbg.ac.at

the one-sided Jacobi methods are very accurate eigensolvers [6]. Recently, it has been reported in [9, 10] that the clever implementation of a serial one-sided Jacobi algorithm reached the efficiency of the QR method. However, the QR method uses a bidiagonalization as a pre-processing step, which means that the relative accuracy is lost and cannot be recovered using the subsequent one-sided Jacobi method. Consequently, the QR method should not be used in such applications where the high relative accuracy of computed singular values is required (e.g., the computation of energies of atoms and molecules in quantum physics and quantum chemistry), and the one-sided Jacobi method is the choice.

Parallel block version of the algorithm was discussed in previous research reports together with a new parallel dynamic ordering based on the estimation of principle angles between two block columns which represent two linear subspaces. Our original idea was based on the parallel implementation of the Lanczos process. This approach was extended in [5] where four new dynamic parallel orderings were designed, implemented and tested.

Next section contains a detailed description of the one-sided block-Jacobi algorithm (OSBJA) for computing the SVD of a general rectangular matrix A. We mention two interesting approaches for the reduction of each local matrix. The first approach consists of computing the symmetric, positive definite Gram matrix from two block columns in each processor and then to use some EVD procedure for symmetric matrices for its reduction to diagonal form. Another approach uses the direct SVD of two matrix blocks inside each processor using the serial Jacobi procedure from LAPACK. We also describe the most efficient dynamic parallel ordering which was used in this report. For local SVD computations, we propose the dynamic local stopping criterion which enables to accelerate the computations in a parallel iteration step.

The last section is devoted to numerical experiments and comparison of five variants of the OSBJA which differ in using the static or dynamic local stopping criterion and in QR pre- and post-processing.

2 One-Sided Block-Jacobi Algorithm

2.1 Computation of Gram matrices

The OSBJA 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

$$A = [A_1, A_2, \ldots, A_r],$$

where the width of A_i is $n_i,\ 1 \leq i \leq r$, so that $n_1 + n_2 + \cdots + n_r = n$. The most natural choice is $n_1 = n_2 = \cdots = n_{r-1} = n_0$, so that $n = (r-1)n_0 + n_r,\ n_r \leq n_0$. Here n_0 can be chosen according to the available cache memory, which is up to 10 times faster than the main memory; this connection will be clear later on.

The OSBJA can be written as an iterative process:

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

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

$$U^{(k)} = \begin{pmatrix} I & & & \\ & U_{ii}^{(k)} & & U_{ij}^{(k)} \\ & & I & \\ & U_{ji}^{(k)} & & U_{jj}^{(k)} \\ & & & & I \end{pmatrix}, \tag{2}$$

where the unidentified matrix blocks are zero. The purpose of matrix multiplication $A^{(k)}U^{(k)}$ in (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

$$\hat{U}^{(k)} = \begin{pmatrix} U_{ii}^{(k)} & U_{ij}^{(k)} \\ U_{ji}^{(k)} & U_{jj}^{(k)} \end{pmatrix} \tag{3}$$

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

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

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 $N(r)$ iterations constitute the first *sweep* of the OSBJA. 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 (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.

One (serial) step of the OSBJA can be described in three parts:

1. For the given pivot pair (i, j), the symmetric, positive semidefinite Gram matrix is computed:

$$\hat{A}_{ij}^{(k)} = [A_i^{(k)} \; A_j^{(k)}]^T \, [A_i^{(k)} \; A_j^{(k)}] = \begin{pmatrix} A_i^{(k)T} A_i^{(k)} & A_i^{(k)T} A_j^{(k)} \\ A_j^{(k)T} A_i^{(k)} & A_j^{(k)T} A_j^{(k)}. \end{pmatrix} \tag{4}$$

   This requires $(n_i + n_j)(n_i + n_j - 1)/2$ dot products or $m\,(n_i + n_j)(n_i + n_j - 1)/2$ flops. As will be soon clear, except for a part of the first sweep, the two diagonal blocks of $\hat{A}^{(k)}$ will

be always diagonal. This reduces the flop count to $m\,(n_i\,n_j + n_i + n_j)$ where $m\,(n_i + n_j)$ comes from the computation of the diagonal elements of $\hat{A}^{(k)}$.

2. $\hat{A}_{ij}^{(k)}$ is diagonalized, i.e., the eigenvalue decomposition of $\hat{A}_{ij}^{(k)}$ is computed:

$$\hat{U}^{(k)T}\,\hat{A}_{ij}^{(k)}\,\hat{U}^{(k)} = \hat{\Lambda}_{ij}^{(k)} \tag{5}$$

and the eigenvector matrix $\hat{U}^{(k)}$ is partitioned according to (3). The matrix $\hat{U}^{(k)}$ defines the orthogonal transformation $U^{(k)}$ in (2) and (1), which is then applied to $A^{(k)}$ and $V^{(k)}$. Notice that the explicit diagonalization of $\hat{A}^{(k)}$ is equivalent to the implicit mutual orthogonalization of columns between column blocks i and j in $A^{(k)}$, i.e., in $(A_i^{(k)}, A_j^{(k)})$. This diagonalization requires (as will be discussed later) on average around $8(n_i + n_j)^3$ flops.

3. Finally, an updating of two block-columns of $A^{(k)}$ and $V^{(k)}$ is required, which requires $2m(n_i + n_j)^2$ flops.

In summary, the kth step of the standard OSBJA requires

$$\begin{aligned} N_{\text{flop}}(k) &\approx m(n_i n_j + n_i + n_j) + 8(n_i + n_j)^3 + 2m(n_i + n_j)^2 \\ &= 64n_0^3 + (9n_0^2 + 2n_0)n \quad (\text{if } m = n = n_0 r) \end{aligned} \tag{6}$$

flops.

Let us discuss shortly these three parts in terms of the CPU time needed for their computation on a serial computer. The first part needs the computation of dot products of length m, which are fast in comparison with matrix multiplications required in the third part. The second part, the eigendecomposition of $\hat{A}^{(k)}$, will be fast provided that we can choose the block width small enough to perform all needed computations in the cache memory (notice that $\hat{A}^{(k)}$ is of the order only $n_i + n_j$). So, the third part of each step in the OSBJA seems to be the most demanding one.

Notice that the Gram matrix $\hat{A}^{(k)}$ in the second phase is symmetric and positive definite. Hence, its SVD is equal to its EVD, and we can use, for example, the LAPACK procedure for the EVD of symmetric matrices. Of course, also a Jacobi procedure (two-sided or one-sided) can be a choice.

2.2 Direct SVD of (A_i, A_j)

Instead of computing the local Gram matrix, one can proceed by direct SVD of (A_i, A_j), i.e. of a local matrix of order $m \times (n_i + n_j)$. Notice that for $m \gg n_i + n_j$ this matrix is tall and skinny and it can be advantageous to perform its QR factorization first and subsequently to work with a 'small' square R-factor of order $n_i + n_j$. However, then the post-processing step is required which consists of the update of left singular vectors by the Q-factor.

For this local SVD computation, one can use the one-sided Jacobi procedure from LAPACK. Moreover, it is *not necessary* to accumulate the individual rotations for local right singular

vectors, and the computation can proceed *without* any updates of block columns. At the end of the global iteration process, say, at iteration k, the columns of $[A_1^{(k)}, A_2^{(k)}, \ldots, A_r^{(k)}]$ are all mutually orthogonal and after normalization represent the left singular vectors. Their norms are equal to singular values. If one has a copy of the original matrix A, the right singular vectors V can be computed *a posteriori* by solving the linear system

$$AV = [A_1^{(k)}, A_2^{(k)}, \ldots, A_r^{(k)}], \tag{7}$$

where the right hand side is equal to $U\Sigma$.

The problem with this approach lies in the accuracy of computed U and Σ at the end of the global iteration process. This accuracy is directly connected to the local stopping criterion of the inner Jacobi SVD procedure for each local SVD of (A_i, A_j) in each parallel iteration step.

The computational complexity per one parallel iteration step is given essentially by the complexity of the inner SVD of (A_i, A_j) which is of order $O(m(n_i + n_j)^2)$ for a version with no QR pre-processing and subsequent post-processing consisting of a matrix-matrix multiplication $Q(A_i, A_j)$.

2.3 Dynamic parallel ordering

Having p processors, the above OSBJA can be parallelized with the blocking factor $r = 2p$ and, for simplicity, assume $n_1 = n_2 = \ldots = n_{2p} = n/(2p)$. Hence, each processor contains two block columns and a parallel dynamic ordering has to define which pairs of block columns will meet in a given processor in each parallel iteration step.

The computation can be organized in such a way that after the first parallel iteration step (initialization), each block column contains inside orthogonal columns. Let us suppose that all $k = n/(2p)$ columns in each block column are *normalized* to the unit Euclidean norm. Hence, each block column is the *orthonormal basis* of the k-dimensional subspace which is spanned by the column vectors of a given block column.

The main idea is to mutually orthogonalize those block columns first which are maximally *inclined* to each other, i.e., their mutual position differs maximally from the orthogonal one. In [5], we have described four new variants of dynamic ordering that are based on estimates of principle angles between two linear subspaces of the same dimension k. Here we mention the most efficient variant 3.

Assume that the original matrix A is of full column rank. Let $e \equiv (1, 1, \ldots, 1)^T \in \mathbb{R}^{k\times 1}$, and for each column block A_j define its *representative vector*,

$$c_j \equiv \frac{A_j\, e}{\|e\|},\ 1 \le j \le 2p. \tag{8}$$

Recall that all block columns of A have linearly independent (orthogonal) columns. Hence, $A_j\, e \neq 0$ for all j throughout the computation. Moreover, assume that the columns in each A_j are orthonormalized so that $\|c_j\| = 1$ for all j. The choice of e ensures the uniform participation of all k one-dimensional subspaces, which constitute $\mathrm{span}(A_j)$, in the definition of c_j.

In variant 3 (see [5]), the weight $w_{ij}^{(3)}$ describes the mutual position of the whole subspace $\mathrm{span}(A_i)$ with respect to the representative vector c_j defined in previous subsection. Hence,

$$w_{ij}^{(3)} \equiv \|A_i^T c_j\| = \frac{\|A_i^T A_j e\|}{\|e\|}. \tag{9}$$

Notice that the orientation of c_j with respect to the *whole* orthonormal basis of $\mathrm{span}(A_i)$ is taken into account.

There is a simple upper bound for $w_{ij}^{(3)}$:

$$w_{ij}^{(3)} = \frac{\|A_i^T A_j e\|}{\|e\|} \leq \|A_i^T A_j\|_2.$$

Therefore, if the global Jacobi process converges with respect to the iteration number r then the positive sequence $\{\max_{i,j} w_{ij,r}^{(3)}\}_{r\geq 1}$ converges to zero.

Conversely, if $w_{ij}^{(3)} = 0$, the representative c_j is perpendicular to *all* basis vectors stored in A_i, i.e., it is perpendicular to the whole subspace $\mathrm{span}(A_i)$. Moreover, since $\sigma_k(A_i^T A_j) = \min_{\|x\|=1} \|A_i^T A_j x\| \geq 0$, this also means that at least the largest principal angle is $\pi/2$.

2.4 Stopping criteria

Each weight $w_{ij}^{(3)}$ needs k scalar products of length m for its computation. Neglecting the length m, we propose the global stopping criterion for variant 3 as

$$\max_{i,j} w_{ij}^{(3)} < k\,\epsilon. \tag{10}$$

Locally, two block columns are not mutually orthogonalized if

$$w_{ij}^{(3)} < k\,\epsilon. \tag{11}$$

In the case of the inner (local) computation *without* Gram matrices, i.e., when the SVD of (A_i, A_j) is computed by the one-sided Jacobi procedure `DGESVJ` from LAPACK, one can use a *static or dynamic local* stopping criterion inside this procedure. Note that the stronger local stopping criterion, the longer time spent inside the local SVD. Therefore, one can accelerate the whole computation if, at the beginning of the global iteration process, local SVDs will be computed with *less* accuracy, but this accuracy will increase towards the end of the global iteration process. The switch to higher local accuracy can be controlled by values of weights $w_{ij}^{(3)}$.

When Gram matrices are computed locally, one can use any procedure from LAPACK designed for the EVD of symmetric matrices. However, in our implementation we use again the one-sided Jacobi procedure `DGESVJ` which does not take into account the matrix symmetry. On the other side, using `DGESVJ` one has control over the local stopping criterion that can be again either static or dynamic.

Let us denote a local matrix by B_{ij}, i.e. $B_{ij} = [A_i, A_j]$ in the case of direct computation or $B_{ij} = [A_i, A_j]^T[A_i, A_j]$ in the case of the Gram matrix. At the end of local SVD, the columns of B_{ij} should be orthogonal. Write $B_{ij} = \hat{B}_{ij} D_{ij}$ where D_{ij} is a diagonal matrix containing the norms of columns in B_{ij}. Hence, $\hat{B}_{ij}$ is orthonormal.

The static local stopping criterion tests the orthogonality of computed columns of $\hat{B}_{ij}$ against the fixed value. The local computation is finished if

$$\frac{\|\hat{B}_{ij}^T \hat{B}_{ij} - I\|_{\mathrm{F}}}{\sqrt{n_i + n_j}} \leq \frac{\sqrt{m}}{5}\, \epsilon_{\mathrm{M}}. \tag{12}$$

Here, $(n_i + n_j)$ is the order of $\hat{B}_{ij}^T \hat{B}_{ij}$ and I, m is the number of rows in the original matrix A and ϵ_{M} is the machine precision.

A dynamic version of the local stopping criterion takes into account the maximal weight encountered in a given parallel iteration step. Denote by RHS the right-hand side of (12) and define

$$maxw \equiv \max_{i,j} w_{ij}^{(3)}$$

in each parallel iteration step. Then we use the following rule for computing RHS:

$$\begin{aligned} \textbf{if} \quad (maxw \geq 10^{-6}) \qquad & \textbf{then} \qquad \mathrm{RHS} = 10^{-4} \times \sqrt{m} \times maxw \\ & \textbf{else} \qquad \mathrm{RHS} = \frac{\sqrt{m}}{5}\, \epsilon_{\mathrm{M}}. \end{aligned} \tag{13}$$

Hence, when the weights are 'large' at the beginning of the OSBJA, RHS depends only on *maxw* and some constants but it does not depend on the machine precision at all. Conversely, towards the end of computation when the weights converge to zero, the local stopping criterion becomes 'rigid' and depends on ϵ_{M} so that the local SVDs are computed practically to full machine precision.

3 Numerical experiments

Five variants of the OSBJA with dynamic ordering were implemented on the Doppler Cluster at the University of Salzburg, Austria. This system consists of 32 nodes, where each node has 16 or 64 cores of type Opteron Series 6200, 2.2 GHz, and with 2–8 GB RAM per core. For our experiments, we used one node in a stand-alone mode of computation, so that measured execution times are quite reliable. All computations were performed using the IEEE standard double precision floating point arithmetic with the machine precision $\epsilon \approx 2.22 \times 10^{-16}$.

The blocking factor $r = 2p$ was used so that all block columns were of the same width, $n_i = n/(2p)$, $1 \leq i \leq 2p$. One quality measure was consistently computed to estimate the accuracy of all algorithms that describes the relative error in the orthogonality of computed left singular vectors:

$$\tilde{Q} \equiv \frac{\|U^T U - I\|_{\mathrm{F}}}{\sqrt{n}}.$$

The results are presented in Tab. 1 and Tab. 2 for a square, random matrix of order $n = 4000$ and $n = 8000$, respectively. In both cases, the matrix singular values were randomly distributed. Five different variants of the OSBJA with dynamic ordering were tested and the abbreviations

Table 1: $n = 4000$

p		NGS	NGD	NGQS	NGQD	G
8	n_{it}	81	107	79	107	82
	T_{p}	1040	695	622	770	671
	$T_{(\text{SVD,PP})}$	(701,-)	(458,-)	(84,428)	(61,590)	(121,301)
	$\tilde{Q}$	3e–13	7e–14	3e-13	1e–13	4e–14
16	n_{it}	181	219	177	220	177
	T_{p}	704	425	455	641	247
	$T_{(\text{SVD,PP})}$	(440,-)	(246,-)	(28,281)	(18,412)	(32,131)
	$\tilde{Q}$	4e–13	7e–14	4e-13	9e–14	5e–14
32	n_{it}	722	770	724	778	724
	T_{p}	859	427	674	817	343
	$T_{(\text{SVD,PP})}$	(383,-)	(191,-)	(14,362)	(8,375)	(17,144)
	$\tilde{Q}$	7e–13	8e-14	8e-13	2e–13	8e–14

Table 2: $n = 8000$

p		NGS	NGD	NGQS	NGQD	G
8	n_{it}	81	119	81	120	80
	T_{p}	7708	6472	6634	6912	3848
	$T_{(\text{SVD,PP})}$	(5533,-)	(4484,-)	(692,3792)	(589,5417)	(856,2036)
	$\tilde{Q}$	6e–13	1e–13	7e-13	5e–13	7e–14
16	n_{it}	178	235	176	236	177
	T_{p}	5401	3263	4706	5604	2711
	$T_{(\text{SVD,PP})}$	(3494,-)	(2070,-)	(227,2955)	(150,3578)	(249,1393)
	$\tilde{Q}$	7e–13	1e–13	7e-13	4e–13	8e–14
32	n_{it}	394	480	384	477	384
	T_{p}	3138	2193	2739	3747	1532
	$T_{(\text{SVD,PP})}$	(1827,-)	(1087,-)	(639,1565)	(39,2010)	(78,808)
	$\tilde{Q}$	8e–13	1e–13	8e-13	4e–13	8e–14

in tables have the following meaning:

- NGS: computation without Gram matrices, static local stopping criterion.
- NGD: computation without Gram matrices, dynamic local stopping criterion.
- NGQS: computation without Gram matrices, QR pre- and post-processing, static local stopping criterion,
- NGQD: computation without Gram matrices, QR pre- and post-processing, dynamic local stopping criterion,

- G: computation with Gram matrices, static local stopping criterion. This is our original implementation from previous papers.

The algorithms were run on $p = 8,\ 16$ and 32 cores and we provide the number of parallel iteration steps needed for the global convergence n_{it}, the total parallel execution time T_{p}, the time spent in local SVDs and pre- plus post-processing $T_{(\mathrm{SVD,PP})}$ and the relative error in the orthogonality of computed left singular vectors $\tilde{Q}$. Note that in the variant G the pre- and post-processing denotes the computation of a Gram matrix and the subsequent update of two block columns, respectively. All timings are in seconds.

Considering the NG variants, the implementation with the static local stopping criterion (LSC) requires always less n_{it} than the dynamic LSC. When no QR pre-processing is used, the complexity of a local SVD is $O(m(n/p)^2)$, whereas for NG variants with QR pre-processing it is $O((n/p)^3)$. This explains relatively small values of time spent in local SVDs for variants NGQS and NGQD, especially for larger values of p. However, the time spent in QR decompositions and subsequent updates is significantly larger than that of local SVDs. This is a drawback of all implementations which use the QR decomposition from LAPACK of tall, skinny matrices.

Our hope was to achieve a good performance for variant NGD. When compared to NGS, the dynamic LSC helps to decrease the time spent in local SVDs. The number of parallel iteration steps increases as compared to NGS but is practically equal to that of NGQD (the same is true for variants NGS and NGQS). In summary, variant NGD is almost always the best one among variants NG with respect to the total parallel execution time T_{p}. The only exception is for $n = 4000,\ p = 8$ when NGQS is better.

Now we compare the variant NGD with the variant G. Mainly due to the smaller value of n_{it}, variant G is always *faster* than the variant NGD despite the extra matrix-matrix multiplications required for computing Gram matrices and updates of two block columns per iteration. Note that square, symmetric Gram matrices are 'small', just of order n/p, and the inner SVD is computed again by the one-sided Jacobi procedure `DGESVJ` from LAPACK which does *not* utilize the symmetry [9, 10]. Our new variant NGD would be competitive with variant G if its values of n_{it} were only slightly decreased.

With respect to the orthogonality of computed left singular vectors, the variant G seems to be the best one. One reason could be that it works with small, symmetric matrices B_{ij} of order n/p in local SVDs. After finishing the inner SVD resulting in the non-normalized matrix of left singular vectors U_{ij}, the matrix of right singular vectors V_{ij} is computed *a posteriori* by solving the linear system $B_{ij}V_{ij} = U_{ij}$. Then, V_{ij} is used in orthogonalization of two local block columns (A_i, A_j) by matrix-matrix multiplication $(A_i, A_j)V_{ij}$ (this is an orthogonal update). It is remarkable that this approach gives practically the same values of $\tilde{Q}$ than the NGD variant which applies the procedure `DGESVJ` directly to (A_i, A_j). It would be interesting to explain this fact using the perturbation theory for both cases.

Notice that the static LSC in variants NGS and NGQS, which is quite stringent during the whole computation, results in equal (or very close) values of $\tilde{Q}$. On the other hand, variants NGD and NGQD with dynamic LSC (also mutually comparable) compute the left singular vectors with *better* level of orthogonality (up to one order of magnitude) although, at the beginning of the OSBJA, the local SVDs are not computed with high accuracy! We have no explanation for

this interesting behavior at this moment.

Next two tables depict very interesting results of how the length of one iteration step T_{it} (in seconds) and the number of parallel iterations steps n_{it} depend on the order of maximal weight *maxw* in the dynamic ordering. Tab. 3 and Tab. 4 contain results for $n = 8000,\ p = 32$ and $n = 8000,\ p = 16$, respectively. Some trends are interesting. First of all, we should mention

Table 3: $n = 8000,\ p = 32$

maxweight		NGS	NGD	QRS	QRD	G
10^{-1}	T_{it}	12	7	7	10	4
	n_{it}	31	28	32	28	30
10^{-2}	T_{it}	10	6	7	8	4
	n_{it}	157	158	150	157	159
10^{-3}	T_{it}	7	3	7	7	4
	n_{it}	51	51	53	55	47
10^{-4}	T_{it}	7	2	7	7	3
	n_{it}	27	45	31	44	30
10^{-5}	T_{it}	5	2	7	7	3
	n_{it}	24	43	21	39	24
10^{-6}	T_{it}	4	2	7	7	3
	n_{it}	20	50	19	48	15
10^{-7}	T_{it}	4	4	7	7	3
	n_{it}	14	35	13	33	16
10^{-8}	T_{it}	2	2	7	7	3
	n_{it}	12	33	15	35	12
10^{-9}	T_{it}	2	-	7	-	3
	n_{it}	8	0	8	0	11
10^{-10}	T_{it}	2	-	6	-	3
	n_{it}	6	0	11	0	9
10^{-11}	T_{it}	2	2	6	-	3
	n_{it}	16	1	11	0	10
10^{-12}	T_{it}	2	2	6	7	3
	n_{it}	5	7	5	7	7
10^{-13}	T_{it}	2	2	6	7	3
	n_{it}	7	17	6	19	9
10^{-14}	T_{it}	2	2	4	5	2
	n_{it}	16	17	9	12	5

that although the weight $w_{ij}^{(3)}$ is only an estimate of mutual geometric position of two linear subspaces we observe almost monotical convergence of *maxw* in all numerical experiments, regardless to the variant of OSBJA.

With respect to the length of one parallel iteration step T_{it}, there are clearly two types of behavior. For variants with no pre- and post-processing (QR decomposition or Gram matrices plus updates), i.e. for NGS and NGD, the decrease of T_{it} with the decrease of *maxw* is observed in the interval $10^{-1} \geq maxw \geq 10^{-8}$, whereas for smaller values of maximal weight T_{it} remains

Table 4: $n = 8000,\ p = 16$

maxweight		NGS	NGD	QRS	QRD	G
10^{-1}	$T_{\rm it}$	49	27	26	23	14
	$n_{\rm it}$	26	26	33	26	26
10^{-2}	$T_{\rm it}$	38	19	25	23	14
	$n_{\rm it}$	59	60	51	62	61
10^{-3}	$T_{\rm it}$	27	10	25	22	13
	$n_{\rm it}$	21	26	22	22	19
10^{-4}	$T_{\rm it}$	22	7	25	22	13
	$n_{\rm it}$	16	23	15	25	13
10^{-5}	$T_{\rm it}$	17	6	25	22	13
	$n_{\rm it}$	10	29	11	27	13
10^{-6}	$T_{\rm it}$	16	6	24	22	13
	$n_{\rm it}$	7	18	7	20	6
10^{-7}	$T_{\rm it}$	13	11	24	22	13
	$n_{\rm it}$	9	24	10	25	8
10^{-8}	$T_{\rm it}$	9	6	24	22	12
	$n_{\rm it}$	4	9	4	9	8
10^{-9}	$T_{\rm it}$	8	-	24	-	11
	$n_{\rm it}$	7	0	5	0	3
10^{-10}	$T_{\rm it}$	7	-	23	-	11
	$n_{\rm it}$	4	0	6	0	7
10^{-11}	$T_{\rm it}$	8	6	23	-	11
	$n_{\rm it}$	3	1	3	0	2
10^{-12}	$T_{\rm it}$	7	6	23	22	11
	$n_{\rm it}$	3	5	3	5	2
10^{-13}	$T_{\rm it}$	7	6	21	21	10
	$n_{\rm it}$	6	9	3	10	6
10^{-14}	$T_{\rm it}$	5	5	18	15	6
	$n_{\rm it}$	3	5	3	5	3

constant. Recall that for these two variants the computation in one parallel iteration step is dominated by the SVD of tall, skinny matrix constructed from two block columns. As *maxw* decreases all block columns are mutually more and more orthogonal so that the LAPACK procedure `DGESVJ` is faster up to some level. On the other hand, other three variants that use some sort of pre- and post-processing (NGQS, NGQD, G) have an almost constant length of $T_{\rm it}$ in the whole range of *maxw* because the amount of this extra work is *constant* and does not depend on *maxw*.

With respect of the number of parallel iteration steps $n_{\rm it}$ spent at given *maxw*, the general trend is again the decrease of $n_{\rm it}$ with decrease of *maxw*—see the results for variant G. However, there are some irregularities especially for smallest orders of *maxw*, *maxw* $\approx 10^{-13},\ 10^{-14}$. Here $n_{\rm it}$ can suddenly increase—see its values in Tab. 3 for all variants except G. Apparently, in this case there is a problem to meet the *global* convergence criterion (10). It seems that for very small values of *maxw* it can be a problem to push the LAPACK procedure `DGESVJ` 'behind its limit'

in requiring 'more orthogonal' left singular vectors—we are somewhere near the perturbation limit of computations with a limited (although double) precision.

Finally, notice the sudden jump of three orders of magnitude in the value of *maxw* in variants with the dynamic LSC (NGD, NGQD). Namely, there are no parallel iterations (or there is just one iteration) when $10^{-9} \geq maxw \geq 10^{-11}$. This is connected with the switch from dynamic to static LSC in (13) which occurs for $maxw < 10^{-6}$. Hence, for $maxw \approx 10^{-7}$, the more stringent, static LSC was applied for the first time according to (13). Then, for NGD with $maxw \approx 10^{-8}$ and $p = 16$ (Tab. 4), $T_{\rm it}$ was almost halved and $n_{\rm it}$ decreased almost three times. For $p = 32$, the same is true for $T_{\rm it}$ but $n_{\rm it}$ remained the same (see Tab. 3). For NGQD, there is no reduction of $T_{\rm it}$ due to the QR pre- and post-processing but for $p = 16$ the value of $n_{\rm it}$ decreases almost three times (see Tab. 4). However, there is no decrease of $n_{\rm it}$ for $p = 32$ (see Tab. 4). And then, suddenly, we have a jump in *maxw* downwards over three orders of magnitude for both dynamic algorithms and both values of p where there are *no* iterations (or just one iteration for NGD)! At the moment, this remarkable property is not well understood. Intuitively, the static LSC achieves 'more' orthogonal columns in p pairs of block columns at the end of each parallel iteration step than the dynamic LSC. However, it is 'the delay' between applying the static LSC at $maxw \approx 10^{-7}$ and the jump occurring at $maxw \approx 10^{-9}$ which is difficult to explain.

4 Conclusions

The idea of a dynamic LSC is quite interesting and results in the acceleration of the OSBJA as compared to the static LSC. However, when the local SVDs are computed with less accuracy at the beginning of a global iteration process, one needs more parallel iteration steps for a given global stopping criterion (notice that the global stopping criterion was *the same* for all algorithms). Perhaps one can gently tune the constants in (13) so that the variant NGD becomes competitive with variant G. At the moment, we have no guidelines for such tuning.

References

[1] A. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, *LAPACK Users' Guide*, Second ed., SIAM, Philadelphia, 1999.

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

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

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

[5] M. Bečka, G. Okša and M. Vajteršic, *New dynamic orderings for the parallel one–sided block–Jacobi SVD algorithm*, accepted for publication in Parallel Processing Letters, April 2014.

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

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

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

[9] Z. Drmač and K. Veselić, *New fast and accurate Jacobi SVD algorithm: I*, SIAM J. Matrix Anal. Appl. 29 (2007) 1322-1342.

[10] Z. Drmač and K. Veselić, *New fast and accurate Jacobi SVD algorithm: II*, SIAM J. Matrix Anal. Appl. 29 (2007) 1343-1362.

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

[12] 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.

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

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

[15] E. Kogbetliantz, *Diagonalization of general complex matrices as a new method for solution of linear equations*, Proc. Intern. Congr. Math. Amsterdam 2 (1954) 356-357.

[16] E. Kogbetliantz, *Solutions of linear equations by diagonalization of coefficient matrices*, Quart. Appl. Math. 13 (1955) 123-132.

[17] F. T. Luk and H. Park, *On parallel Jacobi orderings*, SIAM J. Sci. Statist. Comput. 10 (1989) 18-26.

[18] F. T. Luk and H. Park, *A proof of convergence for two parallel Jacobi SVD algorithms*, IEEE Trans. Comp. 38 (1989) 806-811.

[19] W. Mascarenhas, *On the convergence of the Jacobi method*, poster presentation, 4th SIAM Conference on Parallel Processing for Scientific Computing, Chicago, USA, December 1989.

[20] J. Matejaš, *Convergence of scaled iterates by Jacobi method*, Lin. Alg. Appl. 349 (2002) 17-53.

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

[22] 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.

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