Computing and Informatics, Vol. 44, 2025, 1254–1272, doi: 10.31577/cai_2025_5_1254

BLOCK-JACOBI SVD ALGORITHMS: A REVIEW

Gabriel Okša, Martin Bečka

Institute of Mathematics
Slovak Academy of Sciences
Štefánikova 49
814 73 Bratislava, Slovakia
e-mail: {Gabriel.Oksa, Martin.Becka}@savba.sk

Abstract. We discuss some progress in the design and implementation of block-Jacobi SVD algorithms in the scope of the last 25 years. Two ideas were crucial for enhancing the efficiency of two-sided and one-sided serial or parallel block-Jacobi methods: the so-called dynamic ordering of subproblems solved in each iteration step, and a suitable preconditioning of an original matrix. These two ideas led to a substantial decrease of (serial or parallel) iteration steps needed for the convergence. Consequently, especially the one-sided block-Jacobi SVD algorithm became competitive in speed with some SVD algorithms based on the matrix bi-diagonalization. We also discuss new theoretical results w.r.t. the asymptotic quadratic convergence of block-Jacobi SVD algorithms regardless of the distribution of singular values of the original matrix.

Keywords: Serial and parallel block-Jacobi algorithm, serial and parallel dynamic ordering, asymptotic quadratic convergence, preconditioning

Mathematics Subject Classification 2010: 65F30

1 INTRODUCTION

The computation of the Singular Value Decomposition (SVD) is an important and frequently demanding task in the area of scientific computing. The SVD can be counted among the core tasks of numerical linear algebra with many applications in science and engineering.

Given a matrix $A \in \mathbb{R}^{m\times n}$, $m \geq n$, its SVD is given as the decomposition $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m\times n}$ is the matrix of left singular vectors with orthonormal columns, $\Sigma \in \mathbb{R}^{n\times n}$ is the diagonal matrix of non-negative singular values, and $V \in \mathbb{R}^{n\times n}$ is the orthonormal matrix of right singular vectors.

Basic serial algorithms for the SVD computation have the time complexity $O(mn^2)$ [1]. They are all iterative, work with individual matrix elements and are based either on zeroing of the off-diagonal matrix elements in some cyclic order, or on the bi-diagonalization of A first and a subsequent special algorithm used for the computation of the SVD of a bidiagonal matrix [1].

Among the first group of algorithms, the most important ones are the one-sided and two-sided element-wise Jacobi SVD algorithms [2, 3, 4, 5, 6]. They can deliver very accurate singular values and vectors in finite arithmetic for certain classes of matrices [7, 8]. When applied to symmetric matrices, they can compute even the smallest eigenvalues with high relative accuracy, which is important, for example, in the computation of smallest energies of various quantum systems.

However, serial element-wise Jacobi algorithms need usually a very large number of serial steps for the convergence, so they are quite slow. To enhance their performance, one can parallelize them and work with matrix blocks instead of matrix elements. Both approaches were used since 1980's for various computer architectures – see, e.g., [9, 10, 11, 12].

Since the year 2000, further progress has been achieved by two basic improvements that were incorporated into two- and one-sided block-Jacobi SVD algorithms. The first one is so-called *dynamic ordering* [13, 14] when the zeroed (or orthogonalized) matrix blocks are not chosen according to a prescribed, fixed list, but according to the actual status of the iterated matrix w.r.t. its off-diagonal Frobenius norm. The second idea uses some sort of *preconditioning* [15, 16] where the block-Jacobi algorithm is applied to the preconditioned matrix $A\tilde{V}$ (instead of A) with some orthonormal matrix $\tilde{V}$. The purpose of both improvements is to substantially decrease the number of (serial or parallel) iteration steps needed for the convergence of block-Jacobi methods.

Besides these practical computational issues, some progress was achieved also in the theoretical analysis of the convergence behavior of the block-Jacobi SVD algorithms in exact arithmetic. For the two-sided method with dynamic ordering, the asymptotic *quadratic* convergence was proved [17] regardless of the distribution of singular values (simple, multiple, clusters) under some acceptable assumptions. This proof is valid also for the one-sided block-Jacobi SVD algorithm with a special dynamic ordering (with so-called *exact* weights).

The paper is organized as follows. Section 2 contains a short introduction into the two-sided and one-sided block-Jacobi SVD algorithms with some general ordering of block subproblems that are solved in each iteration step. The parallel block dynamic ordering, that can substantially reduce the number of parallel iteration steps needed for the convergence, is discussed in Section 3. Section 4 contains the description of theoretical results w.r.t. the asymptotic quadratic convergence of the parallel two-sided block-Jacobi SVD algorithm. These results show that the asymp-

totic quadratic convergence occurs not only in the element-wise Jacobi algorithms but also in their block variants. Finally, Section 5 is devoted to the preconditioning of the one-sided block-Jacobi SVD algorithm by using the eigenvalue decomposition (EVD) of the Gram matrix A^TA, or the EVD of the Hermitian factor of the polar decomposition of a given matrix A computed by the cubically convergent Halley's iterations. Section 6 concludes the paper.

2 BLOCK-JACOBI SVD ALGORITHMS

2.1 Two-Sided Method

When introducing the block-Jacobi SVD methods, it is sufficient to consider only square matrices. If an original matrix is of size $m \times n$, $m \geq n$, one can initially compute its thin QR decomposition and then apply the iterative SVD algorithm to the square factor R of size n. Then, the SVD of the original matrix can be reconstructed in an obvious way. In this paper, the assumption about a square matrix A is made only w.r.t. the two-sided block-Jacobi algorithm.

Let us divide a square matrix A of order n into a $\ell \times \ell$ block structure with ℓ blocks in each block row (column). Write the decomposition of n as $n = \lfloor n/\ell \rfloor \ell + r$ with $0 \leq r \leq \ell - 1$. If $r = 0$, ℓ divides n and nothing needs to be done. If $r > 0$, border a matrix A by adding $\ell - r$ zero rows to the bottom of A, $\ell - r$ zero columns to the right of A and $\ell - r$ ones to the lower part of the main diagonal of A. Hence, a bordered matrix is the direct sum $A \oplus I_{\ell-r}$ of size $(\lfloor n/\ell \rfloor + 1)\ell$, where $I_{\ell-r}$ is the identity of order $\ell - r$. Note that the SVD of A can be recovered from the SVD of $A \oplus I_{\ell-r}$ easily. To keep the following exposition simple, we assume that ℓ divides n.

Denote by A_{ij} the $(i,j)^{\text{th}}$ block of order n/ℓ. Hence, there are $\ell(\ell-1)$ off-diagonal blocks in A. In a given iteration step k, the algorithm proceeds by zeroing a pair of off-diagonal blocks $(A_{ij}^{(k)}, A_{ji}^{(k)})$, $i \neq j$, where this pair is chosen according to some ordering. The zeroing is performed by a two-sided unitary transformation

$$(U^{(k)})^T A^{(k)} V^{(k)} = A^{(k+1)},$$

where the $n \times n$ orthogonal matrices $U^{(k)}$ and $V^{(k)}$ are the matrices of local left and right singular vectors, respectively, embedded into the identity matrix I_n of order n. Four blocks of $U^{(k)}$ and $V^{(k)}$, each of order n/ℓ, that are different from blocks of I_n can be chosen so that

$$\begin{pmatrix} U_{ii}^{(k)} & U_{ij}^{(k)} \\ U_{ji}^{(k)} & U_{jj}^{(k)} \end{pmatrix}^T \begin{pmatrix} A_{ii}^{(k)} & A_{ij}^{(k)} \\ A_{ji}^{(k)} & A_{jj}^{(k)} \end{pmatrix} \begin{pmatrix} V_{ii}^{(k)} & V_{ij}^{(k)} \\ V_{ji}^{(k)} & V_{jj}^{(k)} \end{pmatrix} = \begin{pmatrix} A_{ii}^{(k+1)} & 0 \\ 0 & A_{jj}^{(k+1)} \end{pmatrix}, \qquad (1)$$

where the diagonal blocks $A_{ii}^{(k+1)}$ and $A_{jj}^{(k+1)}$ are square, diagonal matrices of order n/ℓ with non-negative diagonal elements (local singular values).

Let us define

$$\tilde{U}^{(k)} \equiv \begin{pmatrix} U_{ii}^{(k)} & U_{ij}^{(k)} \\ U_{ji}^{(k)} & U_{jj}^{(k)} \end{pmatrix}, \quad \tilde{V}^{(k)} \equiv \begin{pmatrix} V_{ii}^{(k)} & V_{ij}^{(k)} \\ V_{ji}^{(k)} & V_{jj}^{(k)} \end{pmatrix}, \tag{2}$$

and

$$\tilde{A}^{(k)} \equiv \begin{pmatrix} A_{ii}^{(k)} & A_{ij}^{(k)} \\ A_{ji}^{(k)} & A_{jj}^{(k)} \end{pmatrix}, \quad \tilde{\Sigma}^{(k)} \equiv \begin{pmatrix} A_{ii}^{(k+1)} & 0 \\ 0 & A_{jj}^{(k+1)} \end{pmatrix}. \tag{3}$$

Because (1) is the SVD of the matrix $\tilde{A}^{(k)}$, the matrix $\tilde{U}^{(k)}$ and $\tilde{V}^{(k)}$ is the unitary matrix of left and right singular vectors of $\tilde{A}^{(k)}$, respectively.

To prove the global convergence of the serial two-sided block-Jacobi SVD algorithm, let us define the square of the off-diagonal Frobenius norm of $A^{(k)}$ by

$$\|\mathrm{off}(A^{(k)})\|_F^2 \equiv \sum_{i \neq j} \|A_{ij}^{(k)}\|_F^2.$$

Let us assume that the off-diagonal blocks $A_{ij}^{(k)}$ and $A_{ji}^{(k)}$ are zeroed at the iteration step $k+1$. Hence, after the iteration step $k+1$, one has

$$\|\mathrm{off}(A^{(k+1)})\|_F^2 = \|\mathrm{off}(A^{(k)})\|_F^2 - \left(\|A_{ij}^{(k)}\|_F^2 + \|A_{ji}^{(k)}\|_F^2\right) \leq \|\mathrm{off}(A^{(k)})\|_F^2.$$

If, in each iteration step, one chooses the off-diagonal blocks for zeroing in such a way that the sum $\left(\|A_{ij}^{(k)}\|_F^2 + \|A_{ji}^{(k)}\|_F^2\right)$ is *maximal* among all pairs of off-diagonal blocks $(A_{rs}^{(k)}, A_{sr}^{(k)})$, $r \neq s$, one can prove the stronger upper bound ([17]):

$$\|\mathrm{off}(A^{(k+1)})\|_F^2 \leq \left(1 - \frac{2}{\ell(\ell-1)}\right) \|\mathrm{off}(A^{(k)})\|_F^2. \tag{4}$$

Hence, under the so-called *dynamic ordering* of subproblems ([17]), the sequence $\|\mathrm{off}(A^{(k)})\|_F^2$ decreases at least as fast as the geometric sequence with the quotient $q = (w-1)/w$, $w = \ell(\ell-1)/2$, and therefore converges to zero. Consequently, the iterative algorithm stops when the off-diagonal Frobenius norm of an iterated matrix $A^{(k)}$ is 'sufficiently small'.

The speed of convergence of the two-sided method is equivalent to the speed of decline of the off-diagonal Frobenius norm. A predefined, *static cyclic* ordering of subproblems that are solved in each iteration step of a sweep, is, so to say, 'blind' – it just combines some blocks A_{ij} and A_{ji} that are zeroed according to a fixed, prescribed list. Hence, the convergence can be very slow, because one can spend too much time in zeroing matrix off-diagonal blocks with very small Frobenius norms.

2.2 One-Sided Method

In the one-sided block-Jacobi algorithm (OSBJA), a matrix A is divided into ℓ block columns that are mutually orthogonalized during iterations according to some (serial or parallel) ordering. The OSBJA is listed as Algorithm 1 below for a general real matrix of size $m \times n$, $m \geq n$.

Algorithm 1 One-sided block-Jacobi SVD algorithm

1: Input: $A = (A_1, A_2, \ldots, A_\ell)$, each block column is $m \times n/\ell$
2: Set: $V = I_n$
3: Choose the pair (i, j) of block columns according to some ordering
4: **while** $\max_{i,j}(\|A_i^T A_j\|_F) > (n/\ell)\,\varepsilon_M$ **do**
5: $\quad G_{ij} = \begin{pmatrix} A_i^T A_i & A_i^T A_j \\ A_j^T A_i & A_j^T A_j \end{pmatrix}$
6: $\quad [X_{ij}, \Lambda_{ij}] = \mathrm{EVD}(G_{ij})$
7: $\quad$ Update of block columns: $(A_i, A_j) = (A_i, A_j) * X_{ij}$
8: $\quad$ Update of right singular vectors: $(V_i, V_j) = (V_i, V_j) * X_{ij}$
9: $\quad$ Choose the pair of block columns (i, j)
10: **end while**
11: σ_r: norms of columns of A_r
12: $U_r = A_r * \mathrm{diag}(\sigma_r^{-1})$

Concerning the choice of pivot (i, j), we can use any static, cyclic ordering with a prescribed list of two column blocks that are orthogonalized in a given iteration step. After computing the local Gram matrix G_{ij}, such an orthogonalization is achieved by using the EVD of G_{ij} and using its eigenvectors X_{ij} in updating the chosen pair of block columns (A_i, A_j) and the corresponding block columns of the right singular vectors (V_i, V_j). The iterations continue until the maximal deviation from orthogonality of any two block columns is sufficiently small. Details can be found in [14].

In exact arithmetic, the one-sided block-Jacobi SVD algorithm can be understood as the two-sided block method implicitly applied to the Gram matrix $A^T A$, which inherits the $\ell \times \ell$ block structure from the one-sided block method. Hence, the one-sided block-Jacobi SVD algorithm converges, because the implicit decrease of the off-diagonal Frobenius norm of the Gram matrix corresponds directly to the increasing degree of orthogonality between any two block columns of a matrix A. However, as with the two-sided block method, the convergence can be very slow when using any *cyclic* ordering with a prescribed, static list of block columns that are mutually orthogonalized in each iteration step. The reason is that one can spend too much computational time in orthogonalizing the block columns that are already nearly orthogonal.

In the next section, we describe new variants of parallel ordering for both two- and one-sided block-Jacobi SVD algorithm called the *dynamic* ordering. The aim is

to design an *optimal strategy* in zeroing the off-diagonal matrix blocks in the two-sided method, and in mutual orthogonalization of block columns in the one-sided method.

3 DYNAMIC BLOCK ORDERING

3.1 Two-Sided Method

The rate of convergence of the two-sided block-Jacobi SVD algorithm is equivalent to the rate of decrease of the Frobenius norm of the off-diagonal matrix blocks. In each iteration step, some blocks are zeroed and other blocks are orthogonally updated. However, a static, cyclic ordering is 'blind' with regard to the individual Frobenius norms of the off-diagonal blocks brought to annihilation. Hence, it can happen that during a static ordering one zeroes many off-diagonal blocks with relatively small individual Frobenius norms, and this leads to many sweeps required for the convergence of the algorithm.

To achieve a faster convergence, we should maximize the off-diagonal norm that is zeroed in each iteration step. Jacobi's approach [5] is optimal for the scalar case, because it annihilates the element with a maximum absolute value in each serial iteration step. We extended his idea to the parallel, block formulation in paper [13].

For a given blocking factor ℓ, let us find in each serial iteration step a pair of indices (i, j) that maximizes $\|A_{rs}\|_F^2 + \|A_{sr}\|_F^2$ over all matrix blocks A_{rs}, $r \neq s$. Subsequently, the matrix blocks A_{ij} and A_{ji} are zeroed. If the cost of this search is reasonably small, one can benefit from the smaller number of iteration steps needed for the convergence of algorithm. Note that in this *dynamic* ordering of subproblems the notion of a sweep is not appropriate anymore. Instead, one is working with the individual iteration steps.

Technically, the above described search can be realized as follows. Let us keep an $\ell \times \ell$ matrix of the off-diagonal block norms and denote it by W. It was shown in [13] that the update of W after an iteration step takes $O(n^2/\ell)$ steps, and the searching for an optimal pair (i, j), which is given by the maximum element of $W + W^T$, will take another $O(\ell^2)$ steps. When comparing the complexity of searching with that of the dominant part of the whole algorithm – namely, with matrix-matrix multiplications, one can see that the cost of finding the optimal pair of indices (i, j) in each iteration step is higher only in the case when $\ell > O(n^{3/4})$. However, such a large blocking factor is not advisable from the point of view of the convergence of algorithm (see the statistical arguments in [13]).

In the parallel computation with p processors, the situation is more complex. In one parallel iteration step that represents p serial iteration steps, one would like to decrease the norm of the off-diagonal matrix blocks as much as possible. This task was formulated in [13] as the *maximum-weight perfect matching problem*. The numerical experiments have shown that the new approach is better in

all analyzed parameters than the implementation of the static cyclic ordering. Especially, the number of iteration steps needed for the convergence was improved significantly.

Notice that the dynamic ordering has to be found at the beginning of each parallel iteration step. Therefore, to be efficient w.r.t. the total parallel execution time, this operation must not occupy a significant portion of the computational time. Fortunately, the update of the off-diagonal Frobenius norm can be performed, to a large extent, in parallel so that the computation of dynamic ordering usually occupies less than 10 % of the total parallel execution time of the whole algorithm.

When turning to the one-sided Jacobi variant with approximately one half of matrix-matrix multiplications as compared to the two-sided method, we would like to design and implement some variant of dynamic ordering, too. This task is discussed next.

3.2 One-Sided Method

Having p processors, the OSBJA (see Algorithm 1) can be parallelized with the blocking factor $\ell = 2p$ and, for simplicity, let us assume that $n/(2p)$ is an integer. 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 consists of orthogonal columns. Let us suppose that all n/ℓ columns in each block column are *normalized* to the unit Euclidean norm, so that each block column is the *orthonormal basis* of some subspace of dimension n/ℓ.

The main idea is to mutually orthogonalize those block columns first which are *minimally inclined* to each other, i.e., their mutual position differs maximally from the orthogonal one. In [14], 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 n/ℓ. Here we mention the most efficient variant.

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

$$c_j \equiv \frac{A_j\, e}{\|e\|}, \quad 1 \le j \le \ell. \tag{5}$$

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 n/ℓ one-dimensional subspaces, which constitute $\mathrm{span}(A_j)$, in the definition of c_j.

The weight w_{ij} describes the mutual position of the whole subspace $\mathrm{span}(A_i)$ with respect to the representative vector c_j defined by Equation (5) (see [14, var3]). Hence,

$$w_{ij} \equiv \|A_i^T c_j\| = \frac{\|A_i^T A_j e\|}{\|e\|}. \tag{6}$$

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}:

$$w_{ij} = \frac{\|A_i^T A_j e\|}{\|e\|} \leq \|A_i^T A_j\|_2,$$

where $\|\cdot\|_2$ is the spectral norm of a given matrix. Therefore, if the global Jacobi process converges with respect to the iteration number r then the positive sequence $\{\max_{i,j} w_{ij,r}\}_{r\geq 1}$ converges to zero.

Conversely, if $w_{ij} = 0$, then 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 the minimal singular value $\sigma_k(A_i^T A_j) = \min_{\|x\|=1} \|A_i^T A_j x\| \geq 0$, this also means that at least the largest principal angle is $\pi/2$.

Note that the weights w_{ij} are computed (updated) at the beginning of each iteration step (serial or parallel), so that this part of computation must be efficient. The clever and fast update of weights was designed and tested in [18]. It is based on the matrix-matrix multiplication, i.e. on a BLAS Level 3 operation, which is crucial for the efficiency of the whole algorithm.

3.2.1 Stopping Criteria

In practical computations, an important question arises: *When* should we stop an iterative algorithm? Since the one-sided method uses about half as many matrix-matrix multiplications per one iteration step compared to the two-sided one, it is clear that the one-sided variant will be used almost exclusively. Next we discuss the appropriate stopping criteria for it.

For the matrix A of size $m \times n$, $m \geq n$, each weight w_{ij} needs n/ℓ scalar products of length m for its computation. Neglecting the length m, we proposed in [14] the global stopping criterion as

$$\max_{i,j} w_{ij} < (n/\ell)\, \epsilon_M, \tag{7}$$

where ϵ_M is the machine precision. Locally, two block columns are *not* mutually orthogonalized if

$$w_{ij} < (n/\ell)\, \epsilon_M. \tag{8}$$

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 [19], one can use a *static or dynamic local* stopping criterion inside this

procedure. Note that the stronger local stopping criterion leads to 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 using the weights w_{ij}, and we tested this approach in [20].

When the 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 hand, using the procedure `DGESVJ` one has control over the local stopping criterion that can be again either static or dynamic [19].

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 a local Gram matrix. At the end of a 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, $\hat{B}_{ij}$ should be orthonormal.

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

$$\frac{\|\hat{B}_{ij}^T \hat{B}_{ij} - I\|_{\mathrm{F}}}{\sqrt{n/p}} \leq \frac{\sqrt{m}}{5}\,\epsilon_M. \tag{9}$$

Here, n/p is the order of $\hat{B}_{ij}^T \hat{B}_{ij}$ and the identity matrix I, and m is the number of rows in the original matrix A.

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 (9) and define

$$maxw \equiv \max_{i,j} w_{ij}$$

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

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

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 ϵ_M 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 will be computed practically to full machine precision.

Although the two-sided method is not widely used in practice, the theoretical analysis of its rate of convergence may help to better understand the properties of

block-Jacobi algorithms in general and those of the one-sided method in particular. This issue is discussed in the next section.

4 ASYMPTOTIC QUADRATIC CONVERGENCE

Using the dynamic ordering of subproblems described above, the asymptotic quadratic convergence of the serial and parallel two-sided block-Jacobi SVD algorithm has been proved in [17] under some assumptions, regardless of the distribution of singular values of a matrix A (simple, multiple, clusters). Below we report the results for the parallel algorithm running on p processors with the greedy implementation of dynamic ordering (see [13] for details about the greedy implementation of dynamic ordering).

4.1 Parallel Two-Sided Block-Jacobi Algorithm

Let us divide a square matrix A of order n into a $\ell \times \ell$ block structure using the blocking factor $\ell = 2p$, $\ell \geq 4$, where p is the number of processors. Here we assume that ℓ divides n. Thus, ℓ denotes the number of blocks in each block row (column) and each block is square of order n/ℓ.

At the beginning of a parallel iteration step k, $2p$ off-diagonal blocks of $A^{(k)}$ with block indices $(X_{k,1}, Y_{k,1})$, $(Y_{k,1}, X_{k,1})$, $\ldots$, $(X_{k,p}, Y_{k,p})$, $(Y_{k,p}, X_{k,p})$, $X_{k,i} < Y_{k,i}$ for all i, are annihilated using the greedy implementation of parallel dynamic ordering (GIPDO). The pairs of off-diagonal blocks are ordered decreasingly with respect to their weights $w_{ij}^{(k)}$ measured by the sum of squares of their Frobenius norms. After choosing the first pair, additional $p-1$ pairs are chosen for annihilation with a decreasing weight in a compatible way, i.e., the block indices of each new pair must be different from those of all already chosen blocks. This ensures the selection of p 2×2 block sub-problems that can be solved in parallel. More details about the communication and computational complexity of GIPDO can be found in [13].

It was shown in [17] that the algorithm converges globally, because the decrease of the off-diagonal Frobenius norm in one parallel iteration step has an upper bound

$$\|\mathrm{off}(A^{(k+1)})\|_F^2 \leq \left(1 - \frac{1}{2\ell - 3}\right) \|\mathrm{off}(A^{(k)})\|_F^2. \tag{11}$$

Hence, $\|\mathrm{off}(A^{(k+1)})\|_F^2$ converges to zero at least as fast as the geometric sequence with the quotient $q = 1 - 1/(2\ell - 3)$.

Writing

$$A^{(k)} = \mathrm{diag}(A^{(k)}) + \mathrm{off}(A^{(k)}), \tag{12}$$

where $\mathrm{diag}(A^{(k)})$ is the diagonal matrix with diagonal elements of the iterated matrix $A^{(k)}$, we can make the following assumptions at some iteration step k_1:

A1. The off-diagonal Frobenius norm of $A^{(k_1)}$ is small enough:

$$\|\mathrm{off}(A^{(k_1)})\|_F = \sqrt{\sum_{i\neq j} \|A_{ij}^{(k)}\|_F^2} < \frac{\min(d_s, d_c)}{8}, \tag{13}$$

where the absolute gap d_s between r different simple (or multiple) singular values σ_{s_i}, $1 \leq i \leq r$, is defined as $d_s \equiv \min_{i\neq j} |\sigma_{s_i} - \sigma_{s_j}|$, $1 \leq i, j \leq r$, and the absolute gap for q clusters of singular values with centers $c_1, \ldots, c_q$ is defined by $d_c \equiv \min_{i\neq j} |c_i - c_j|$, $1 \leq i, j \leq q$.

A2. The main diagonal of $A^{(k)}$, $k \geq k_1$, is ordered (e.g., decreasingly) by suitable row and column permutations so that the diagonal elements of $A^{(k_1)}$ affiliated with the same multiple singular value occupy successive positions on the diagonal.

A3. The partition of $A^{(k)}$, $k \geq k_1$, is such that the diagonal elements affiliated with the same multiple singular value (or cluster) are confined to a single diagonal block.

Now consider the parallel two-sided block-Jacobi SVD algorithm with the GIPDO using the blocking factor $\ell = 2p$. Let k_1 be the first integer for which the assumptions A1.–A3. are met. Then the main result proved in [17] says that there exists the integer K, $\ell - 1 \leq K < 2\ell(\log \ell + 1)$, such that for all iteration steps $k \geq k_1$ one has

$$\|\mathrm{off}(A^{(k+K)})\|_F \leq \sqrt{12(\ell-2)}\frac{\|\mathrm{off}(A^{(k)})\|_F^2}{\delta} \tag{14}$$

with $\delta = \sqrt{2}\min(d_s, d_c)/4$. This means that the parallel two-sided block-Jacobi SVD algorithm with the GIPDO converges quadratically, and this behavior can be observed after the iteration step k_1 with the 'period' K w.r.t. the decrease of the off-diagonal Frobenius norm of the iterated matrix.

Since the one-sided block-Jacobi SVD algorithm in exact arithmetic with *exact* weights $w_{ij} = \|A_i^T A_j\|_F^2$ and dynamic ordering is implicitly the two-sided block-Jacobi SVD algorithm applied to the Gram matrix $A^T A$, its asymptotic quadratic convergence follows from the above result applied to $A^T A$. It should be noticed, however, that for *approximate weights*, which were defined in Subsection 3.2, the asymptotic quadratic convergence has not been proved yet.

Besides the asymptotic quadratic convergence, the convergence of the whole computed singular triplets (i.e., of the left and right singular vector corresponding to a given singular value) to the exact ones, as well as that of certain computed left and right subspaces to the given ones, was proved in [21]. It remains an open problem if at least some singular triplets can converge quadratically.

The second way of how to decrease the number of (serial or parallel) iteration steps needed for the convergence is to apply some *preconditioning* to an original matrix A before starting the Jacobi process. This issue is discussed in the next section.

5 PRECONDITIONING

Let us suppose that a matrix A has the SVD decomposition $A = U\Sigma V^T$, i.e. $AV = U\Sigma$. In other words, the columns of AV are contained in the vector space arising from the columns of $U\Sigma$. Now imagine that before the computation of the SVD one multiplies a matrix A with an orthogonal matrix $\tilde{V}$ such that $A\tilde{V}$, as the linear vector space of its columns, will be 'close' to the range of columns of $U\Sigma$. (If $A\tilde{V}$ were *exactly* in the range of $U\Sigma$, no further computation would be required.) Such a 'closeness' means that the subsequent application of any block-Jacobi SVD algorithm to $A\tilde{V}$ (instead of A) will require substantially less serial or parallel iteration steps for its convergence.

In the following, two methods of preconditioning are described for the serial one-sided block-Jacobi SVD algorithm with dynamic ordering. They differ in obtaining the suitable preconditioner $\tilde{V}$. Note that the preconditioning can be easily extended to the parallel algorithm, where one uses the parallel dynamic ordering.

5.1 Preconditioning Using the Gram Matrix

In the first approach, one computes the EVD of the Gram matrix $B = A^T A$ and the matrix $\tilde{V}$ is the orthonormal matrix of eigenvectors of B. The preconditioned one-sided block-Jacobi SVD algorithm (POSBJA) is listed below as Algorithm 2.

Algorithm 2 The POSBJA with the Gram matrix and dynamic ordering

Input: $A = (A_1, A_2, \ldots, A_\ell)$, each block column is $m \times n/\ell$
Compute the Gram matrix: $B = A^T A$
$[\tilde{V}, \Lambda] = \mathrm{EVD}(B)$
$A = A\tilde{V}$
Set: $V = \tilde{V}$
Compute the weights w_{rs}
Choose the pair (i, j) of block columns with the maximum weight *maxw*
while $maxw \leq (n/\ell)\varepsilon_M$ **do**
    $G_{ij} = \begin{pmatrix} A_i^T A_i & A_i^T A_j \\ A_j^T A_i & A_j^T A_j \end{pmatrix}$
    $[X_{ij}, \Lambda_{ij}] = \mathrm{EVD}(G_{ij})$
    $(A_i, A_j) = (A_i, A_j) * X_{ij}$
    $(V_i, V_j) = (V_i, V_j) * X_{ij}$
    Update the weights w_{rs}
    Choose the pair of block columns (i, j) with the maximum weight *maxw*
end while
σ_r : norms of columns of A_r
$U_r = A_r * \mathrm{diag}(\sigma_r^{-1})$

Algorithm 2 differs from Algorithm 1 in steps 2–5 where the preconditioning of A takes place, and in using the dynamic ordering that is defined by approximate weights (see Subsection 3.2).

The price for obtaining the preconditioner $\tilde{V}$ seems to be high, both in its computation and subsequent application. A necessary condition of its applicability is its high degree of orthogonality. To be efficient, we need a fast EVD algorithm, which generates a highly orthogonal of eigenvectors, and then a fast matrix-matrix multiplication.

The EVD of Gram matrix B at the beginning is computed by the LAPACK procedure `DSYEVD` [19], which implements a very fast Divide-and-Conquer algorithm. During the iteration process we hold the vector of column norms of A. From this vector we estimate the condition number of the Gram matrix G_{ij}. If it is below one hundred, the procedure `DSYEVD` is applied to compute the EVD of G_{ij}; otherwise the procedure `DGESVJ` is used, which uses the element-wise Jacobi algorithm [19]. Both methods compute the eigenvectors X_{ij} of the Gram matrix G_{ij} with a sufficiently high level of orthogonality.

As was shown by the analysis in [15], if A is very ill-conditioned, the use of the Gram matrix A^TA for the computation of the preconditioner $\tilde{V}$ can be numerically unreliable. Consequently, the matrix multiplication $A\tilde{V}$ can put the range of $A\tilde{V}$ very far from the range of $U\Sigma$ even for a highly orthogonal matrix $\tilde{V}$. Therefore, for a very ill-conditioned matrix A it is not advisable to compute the Gram matrix A^TA, and a different approach is discussed in the next subsection.

5.2 Preconditioning Using the Halley Iterations

Another way, how to find a preconditioner $\tilde{V}$ such that $A\tilde{V}$ will be close to $U\Sigma$ in the sense of linear subspaces, is to compute the polar decomposition of A, take its symmetric factor, compute its EVD and use the matrix of eigenvectors as $\tilde{V}$. Note that each matrix $A \in \mathbb{R}^{m\times n}$, $m \geq n$, has the polar decomposition $A = U_pH$, where $U_p \in \mathbb{R}^{m\times n}$ has orthonormal columns and $H \in \mathbb{R}^{n\times n}$ is symmetric and positive semidefinite. Let us write $A = U\Sigma V^T$, the 'thin' SVD of A, where U is of size $m \times n$ with orthonormal columns (the matrix of left singular vectors), Σ is the diagonal square matrix of order n with real nonnegative elements (singular values) and V is the unitary matrix of order n (right singular vectors). Then using the EVD of Hermitian factor, $H = \tilde{V}\Lambda\tilde{V}^T$, it follows immediately that, in *exact* arithmetic, $A = (U_p\tilde{V})\Lambda\tilde{V}^T$ is the SVD of A with $U = U_p\tilde{V}$, $\Sigma = \Lambda$ and $V = \tilde{V}$. In other words, $A\tilde{V} = U\Sigma$, so that the preconditioning with exact $\tilde{V}$ should lead to exact (scaled) U. However, in *finite* arithmetic, we can only hope for some degree of orthogonality of $A\tilde{V}$ and the 'closeness' of range($A\tilde{V}$) to range(U) in the sense of a 'small' distance between these two linear subspaces, so that the OSBJA applied to $A\tilde{V}$ will need substantially less iteration steps for convergence than in the case of no preconditioning. Note that the preconditioner $\tilde{V}$ is computed directly from A *without* using the Gram matrix. Hence, one can also hope that this approach will give a good preconditioner for an ill-conditioned matrix A, because

its large 2-norm condition number $\kappa(A)$ is never squared during the computation of $\tilde{V}$.

The main task is to efficiently compute the polar decomposition of A using some numerically stable and fast algorithm. In [22, 23, 24], the authors analyzed the behavior and performance of the Halley iterations for computing the polar decomposition. Hence, we have implemented and tested the subroutine $[U_p, H] =$ `Halley`(A) in [16], and the matrix of eigenvectors of the symmetric factor H was used as the preconditioner. The corresponding POSBJA is listed below as Algorithm 3.

Algorithm 3 The POSBJA with Halley's iterations and dynamic ordering

1: Input: ℓ, $A = (A_1, A_2, \ldots, A_\ell)$, each block column is $m \times n/\ell$
2: $[U_p, H] = \mathrm{Halley}(A)$ (polar decomposition of A)
3: $[\tilde{V}, \Lambda] = \mathrm{EVD}(H)$ (EVD of the Hermitian factor)
4: $A = A\,\tilde{V}$ (preconditioning)
5: Set: $V = \tilde{V}$ (initial matrix of right singular vectors)
6: Compute the weights w_{rs}
7: Choose block columns (i, j) with the maximum weight *maxw*
8: $iter = 0$
9: **while** $(maxw \geq (n/\ell)\,\varepsilon_{\mathrm{M}})$ **do**
10: $iter = iter + 1$
11: Compute: $G_{ij} = \begin{pmatrix} A_i^T A_i & A_i^T A_j \\ A_j^T A_i & A_j^T A_j \end{pmatrix}$ (local Gram matrix)
12: $[X_{ij}, \Lambda_{ij}] = \mathrm{EVD}(G_{ij})$
13: $(A_i, A_j) = (A_i, A_j)\,X_{ij}$
14: $(V_i, V_j) = (V_i, V_j)\,X_{ij}$
15: Update the weights w_{rs}
16: Choose block columns (i, j) with the maximum weight *maxw*
17: **end while**
18: $\sigma_r = \|A(:, r)\|_2$, $1 \leq r \leq n$ (Euclidean norm)
19: $U = A\,\mathrm{diag}(\sigma_r^{-1})$ (left singular vectors of A)

Its subroutine `Halley`(A) is given below as Algorithm 4.

It begins with a matrix normalization (step 2) and finding the upper bound for the minimal singular value $\sigma_{\min}(A)$ (steps 3–5). We take the largest square submatrix of A and estimate its 2-norm condition number. Then the upper bound for $\sigma_{\min}(A)$ is given by λ_0 (see [22, p. 2711]), and this bound is needed for the (sub)optimal behavior of the algorithm w.r.t. its rate of convergence ([22, 23]). It is known that the algorithm converges cubically ([24, p. A1328]), so that the number of needed iterations can be fixed to 6 (step 6) in finite arithmetic with $\varepsilon_{\mathrm{M}} \approx 2.22 \times 10^{-16}$. Each iteration starts with computation of some auxiliary variables that serve for the evaluation of dynamic parameters a_k, b_k and c_k in step 12. Then a special, 2×1 block matrix is formed using the 'old' scaled iterate X_{k-1} and the identity I_n of order n, and its QR decomposition is computed (step 13). Computationally, this is the

Algorithm 4 $[U_p, H] = \mathrm{Halley}(A)$: Halley's iterations for polar decomposition

1: Input: $A \in \mathbb{R}^{m\times n}$, $m \geq n$
2: $\alpha = \|A\|_F$, $X_0 = \alpha^{-1}A$ (matrix normalization)
3: $C = A(1:n, 1:n)$ (largest square submatrix of A)
4: $\gamma = \mathrm{condest}(C)/\|A\|_1$, $\beta = 1/(\sqrt{n}\,\gamma)$
5: $\lambda_0 = \beta/\alpha$ (lower bound for the smallest singular value of A)
6: $iter = 6$ (fixed number due to the cubic convergence)
7: $k = 0$
8: **while** $k \leq iter$ and $\|X_k - X_{k-1}\|_F \geq \varepsilon_{\mathrm{M}}^{1/3}$ **do**
9: $k = k + 1$
10: $\gamma = 4(1 - \lambda_{k-1}^2)/\lambda_{k-1}^4$
11: $\varphi = 8(2 - \lambda_{k-1}^2)/(\lambda_{k-1}^2\sqrt{1+\gamma})$, $\delta = 8 - 4\gamma + \varphi$
12: $a_k = \sqrt{1+\gamma} + 0.5\sqrt{\delta}$, $b_k = (a_k - 1)^2/4$, $c_k = a_k + b_k - 1$
13: Form the matrix $\begin{pmatrix}\sqrt{c_k}X_{k-1}\\ I_n\end{pmatrix}$ of size $(m+n)\times n$, and compute its 'thin' QR decomposition: $\begin{pmatrix}\sqrt{c_k}X_{k-1}\\ I_n\end{pmatrix} = \begin{pmatrix}Q_1\\ Q_2\end{pmatrix} R$, where Q_1 is of size $m \times n$.
14: $X_k = (b_k/c_k)X_{k-1} + [a_k - (b_k/c_k)]\, c_k^{-1/2}\, Q_1 Q_2^H$ (new iterated matrix)
15: **end while**
16: $U_p = X_k$ (polar factor with orthonormal columns of size $m \times n$)
17: $H = 0.5 * \left(U_p^H A + (U_p^H A)^H\right)$ (Hermitian factor of size $n \times n$)

most demanding operation in each iteration step, and its efficient implementation is crucial for the efficiency of the whole preconditioning. Finally, new iterate X_k is computed in step 14 using only the partitioned Q-factor. After convergence, the newest iterate is equal to the polar factor U_p, while the Hermitian factor H is computed using one matrix multiplication U_pA and matrix addition (steps 16–17). Note that the EVD of H provides the unitary matrix of eigenvectors $\tilde{V}$ which is used in the preconditioning of the OSBJA (steps 3–4 of Algorithm 2).

The numerical results show [16] that the use of the *partial* polar decomposition consisting of only one iteration step in Algorithm 4 (instead of fixed 6 iteration steps due to its cubic convergence) can be efficient for very ill-conditioned matrices A. In this case, the computation of the Gram matrix A^TA would lead to numerical difficulties that would destroy the reliability of subsequent steps in preconditioning. For well-conditioned matrices A, the use of the preconditioner coming from the Gram matrix A^TA can be recommended.

For the QR decomposition in Halley's iterations, the special matrix structure can be exploited by using either the Householder vectors of the constant length $n+1$ or a composition of Householder reflections and Givens rotations. Two new variants were proposed and both of them require less flops for computing the R- and Q-factor than the standard QR decomposition of the whole $2n \times n$ matrix using Householder vectors. The minimal computation times both in the QR decomposition as well

as in the OSBJA for very ill-conditioned matrices were achieved by exploiting the special structure of a matrix X_k in each iteration step as well as the blocking in application of Householder vectors. It is interesting to note that even one iteration step of Halley's iterations delivers such a Hermitian factor H that its eigenvector matrix $\tilde{V}$ is a good preconditioner, i.e., the matrix $A\tilde{V}$ is quite close to $U\Sigma$ in the sense of column vector spaces. More details can be found in [16].

6 CONCLUSIONS

The above mentioned ideas of dynamic ordering and preconditioning have substantially enhanced the efficiency of the block-Jacobi SVD algorithms, so that especially the parallel one-sided block-Jacobi method with the dynamic ordering and preconditioning became competitive with the SVD algorithms based on the matrix bi-diagonalization [25]. For well-conditioned matrices, our implementation of the POSBJA outperforms the fastest serial algorithm from the LAPACK library, which is based on the Divide-And-Conquer algorithm [18]. In the parallel environment, the POSBJA is faster than the ScaLAPACK routine `PDGESVD` regardless of the condition number of A [15]. Together with their excellent numerical properties in computing the singular triplets with high relative accuracy for certain classes of matrices, the block-Jacobi SVD algorithms can be used on modern computer architectures also in processing (compressing) the higher dimensional data sets that are given in the form of tensors.

Acknowledgment

The authors were supported by the VEGA grant No. 2/0001/23.

REFERENCES

[1] Golub, G. H.—Van Loan, C. F.: Matrix Computations. Johns Hopkins University Press, 2013.

[2] Demmel, J. W.—Veselić, K.: Jacobi's Method Is More Accurate Than QR. SIAM Journal on Matrix Analysis and Applications, Vol. 13, 1992, No. 4, pp. 1204–1245, doi: 10.1137/0613074.

[3] Drmač, Z.: Implementation of Jacobi Rotations for Accurate Singular Value Computation in Floating-Point Arithmetic. SIAM Journal on Scientific Computing, Vol. 18, 1997, No. 4, pp. 1200–1222, doi: 10.1137/S1064827594265095.

[4] Drmač, Z.: A Posteriori Computation of the Singular Vectors in a Preconditioned Jacobi SVD Algorithm. IMA Journal of Numerical Analysis, Vol. 19, 1999, No. 2, pp. 191–213, doi: 10.1093/imanum/19.2.191.

[5] Jacobi, C. G. J.: Über ein Leichtes Verfahren die in der Theorie der Säcularstörungen Vorkommenden Gleichungen Numerisch Aufzulösen. Journal für die

Reine und Angewandte Mathematik, Vol. 30, 1846, pp. 51–94, http://eudml.org/doc/147275 (in German).

[6] Matejaš, J.—Hari, V.: Accuracy of the Kogbetliantz Method for Scaled Diagonally Dominant Triangular Matrices. Applied Mathematics and Computation, Vol. 217, 2010, No. 8, pp. 3726–3746, doi: 10.1016/j.amc.2010.09.020.

[7] Drmač, Z.—Veselić, K.: New Fast and Accurate Jacobi SVD Algorithm. I. SIAM Journal on Matrix Analysis and Applications, Vol. 29, 2008, No. 4, pp. 1322–1342, doi: 10.1137/050639193.

[8] Drmač, Z.—Veselić, K.: New Fast and Accurate Jacobi SVD Algorithm. II. SIAM Journal on Matrix Analysis and Applications, Vol. 29, 2008, No. 4, pp. 1343–1362, doi: 10.1137/05063920X.

[9] Bečka, M.—Vajteršic, M.: Block-Jacobi SVD Algorithms for Distributed Memory Systems I: Hypercubes and Rings. Parallel Algorithms and Applications, Vol. 13, 1999, No. 3, pp. 265–287, doi: 10.1080/10637199808947370.

[10] Bečka, M.—Vajteršic, M.: Block-Jacobi SVD Algorithms for Distributed Memory Systems II: Meshes. Parallel Algorithms and Applications, Vol. 14, 1999, No. 1, pp. 37–56, doi: 10.1080/10637199808947377.

[11] Bischof, C. H.: Computing the Singular Value Decomposition on a Distributed System of Vector Processors. Parallel Computing, Vol. 11, 1989, No. 2, pp. 171–186, doi: 10.1016/0167-8191(89)90027-6.

[12] Bremt, R. P.—Luk, F. T.: The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays. SIAM Journal on Scientific and Statistical Computing, Vol. 6, 1985, No. 1, pp. 69–84, doi: 10.1137/0906007.

[13] Bečka, M.—Okša, G.—Vajteršic, M.: Dynamic Ordering for a Parallel Block-Jacobi SVD Algorithm. Parallel Computing, Vol. 28, 2002, No. 2, pp. 243–262, doi: 10.1016/S0167-8191(01)00138-7.

[14] Bečka, M.—Okša, G.—Vajteršic, M.: New Dynamic Orderings for the Parallel One-Sided Block-Jacobi SVD Algorithm. Parallel Processing Letters, Vol. 25, 2015, No. 2, Art. No. 1550003, doi: 10.1142/S0129626415500036.

[15] Bečka, M.—Okša, G.: Preconditioned Jacobi SVD Algorithm Outperforms PDGESVD. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2019). Springer, Cham, Lecture Notes in Computer Science, Vol. 12043, 2020, pp. 555–566, doi: 10.1007/978-3-030-43229-4_47.

[16] Bečka, M.—Okša, G.: Preconditioning of the One-Sided Block-Jacobi SVD Algorithm by Polar Decomposition. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2024). Springer, Cham, Lecture Notes in Computer Science, Vol. 15581, 2025, pp. 205–216, doi: 10.1007/978-3-031-85703-4_14.

[17] Okša, G.—Yamamoto, Y.—Bečka, M.—Vajteršic, M.: Asymptotic Quadratic Convergence of the Two-Sided Serial and Parallel Block-Jacobi SVD Algorithm. SIAM Journal on Matrix Analysis and Applications, Vol. 40, 2019, No. 2, pp. 639–671, doi: 10.1137/18M1222727.

[18] Bečka, M.—Okša, G.—Vidličková, E.: New Preconditioning for the One-Sided

Block-Jacobi SVD Algorithm. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2017). Springer, Cham, Lecture Notes in Computer Science, Vol. 10777, 2018, pp. 590–599, doi: 10.1007/978-3-319-78024-5_51.

[19] Anderson, E.—Bai, Z.—Bischof, C.—Blackford, L. S.—Demmel, J.—Dongarra, J. et al.: LAPACK Users' Guide. SIAM, 1999.

[20] Bečka, M.—Okša, G.: New Approach to Local Computations in the Parallel One-Sided Jacobi SVD Algorithm. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9573, 2016, pp. 605–617, doi: 10.1007/978-3-319-32149-3_56.

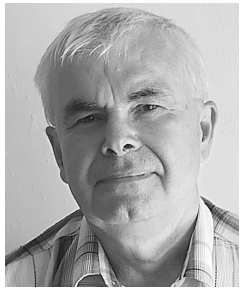
[21] Okša, G.—Yamamoto, Y.—Vajteršic, M.: Convergence to Singular Triplets in the Two-Sided Block-Jacobi SVD Algorithm with Dynamic Ordering. SIAM Journal on Matrix Analysis and Applications, Vol. 43, 2022, No. 3, pp. 1238–1262, doi: 10.1137/21M1411895.

[22] Nakatsukasa, Y.—Bai, Z.—Gygi, F.: Optimizing Halley's Iteration for Computing the Matrix Polar Decomposition. SIAM Journal on Matrix Analysis and Applications, Vol. 31, 2010, No. 5, pp. 2700–2720, doi: 10.1137/090774999.

[23] Nakatsukasa, Y.—Higham, N. J.: Backward Stability of Iterations for Computing the Polar Decomposition. SIAM Journal on Matrix Analysis and Applications, Vol. 33, 2012, No. 2, pp. 460–479, doi: 10.1137/110857544.

[24] Nakatsukasa, Y.—Higham, N. J.: Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD. SIAM Journal on Scientific Computing, Vol. 35, 2013, No. 3, pp. A1325–A1349, doi: 10.1137/120876605.

[25] Dongarra, J.—Gates, M.—Haidar, A.—Kurzak, J.—Luszczek, P.—Tomov, S.—Yamazaki, I.: The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale. SIAM Review, Vol. 60, 2018, No. 4, pp. 808–865, doi: 10.1137/17M1117732.



Gabriel Okša works in the Mathematical Institute of the Slovak Academy of Sciences as the Head of Department of Informatics. In the year 1995 he received the Royal Society Fellowship and spent one year at the Loughborough University, United Kingdom, as the researcher in the field of parallel numerical algorithms. He is interested in the numerical properties and efficient implementation of serial and parallel algorithms for the eigenvalue and singular value decomposition of large, dense matrices.

Martin Bečka works at the Mathematical Institute of the Slovak Academy of Sciencesin the Department of Informatics as a Senior Researcher. In the years 2003–2005, he completed a two-year postdoctoral study at ETH Zurich, Switzerland, at the Department of Computer Science. His main interest is the efficient parallelization and implementation of algorithms for the eigenvalue and singular value decomposition of large, dense matrices.