UNIVERSITÄT SALZBURG

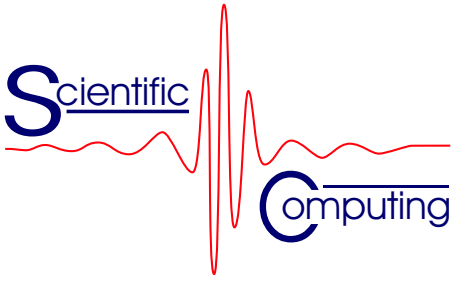
Efficient Preprocessing in the Parallel Block-Jacobi SVD Algorithm

Gabriel Okša[a] Marián Vajteršic

[a]Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic

Technical Report 2005-02 June 2005

Department of Scientific Computing



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

Technical Report Series

Efficient Preprocessing in the Parallel Block–Jacobi SVD Algorithm

Gabriel Okša* and Marián Vajteršic†

Abstract. *One way, how to speed up the computation of the singular value decomposition of a given matrix $A \in \mathbb{C}^{m\times n}$, $m \geq n$, by the parallel two-sided block-Jacobi method, consists of applying some pre-processing steps that would concentrate the Frobenius norm near the diagonal. Such a concentration should hopefully lead to fewer outer parallel iteration steps needed for the convergence of the entire algorithm. It is shown experimentally, that the QR factorization with the complete column pivoting, optionally followed by the LQ factorization of the R-factor, can lead to a substantial decrease of the number of outer parallel iteration steps, whereby the details depend on the condition number and on the distribution of singular values including their multiplicity. A subset of ill-conditioned matrices has been identified, for which the dynamic ordering becomes inefficient. Best results in numerical experiments performed on the cluster of personal computers were achieved for well-conditioned matrices with a multiple minimal singular value, where the number of parallel iteration steps was reduced by two orders of magnitude. However, the gain in speed, as measured by the total parallel execution time, depends decisively on the implementation of the distributed QR and LQ factorizations on a given parallel architecture. In general, the reduction of the total parallel execution time up to one order of magnitude has been achieved.*

1 Introduction

The two-sided serial Jacobi method is a numerically reliable algorithm for the computation of the eigenvalue/singular value decomposition (EVD/SVD) of a general matrix $A \in \mathbb{C}^{m\times n}$, $m \geq n$ [1]. For certain classes of matrices [6], it can achieve a high *relative* accuracy in computing the tiniest singular values (or eigenvalues), which is of great importance in such applications as quantum physics or chemistry.

Unfortunately, the Jacobi method – and especially its two-sided variant – belongs to the slowest known algorithms for computing the EVD/SVD. Our experiments have shown that the *dynamic* parallel ordering, which was proposed and implemented in [2], typically reduces the number of outer parallel iteration steps in the two-sided block-Jacobi algorithm by 30 – 40 per cent for

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

†Institute for Scientific Computing, University of Salzburg, Salzburg, Austria, email: marian@cosy.sbg.ac.at.

random, dense matrices of orders $10^3 - 10^4$. In general, however, this is not enough to make the method competitive with faster (albeit less accurate) algorithms based on the bi-diagonalization (cf. [2, 3]).

One way, how to further decrease the number of outer parallel iteration steps, can be based on applying an appropriate *preconditioner* to the original matrix A at the beginning of iteration process. Ideally, such a preconditioner should concentrate the Frobenius norm of A towards diagonal as much as possible. Notice that if A were a block-diagonal matrix, and if its partition covered all diagonal blocks, only one outer parallel iteration step would be required for the whole SVD. Hence, it is hoped that the concentration of the Frobenius norm towards the diagonal might substantially decrease the number of outer iteration steps.

The connection between diagonal elements of R- or L-factor of a general matrix A and its singular values (SVs) was studied by Stewart in [11]. He has shown experimentally that after the QR factorization with column pivoting (QRFCP), followed optionally by the LQ factorization (LQF) of the R-factor with or without column pivoting, the absolute values of diagonal elements in resulting upper or lower triangular matrix (so called *R-values* or *L-values*) are, in general, very good approximations of SVs of A. Since the sum of squares of SVs is equal to the square of the Frobenius norm of A, this also means that the Frobenius norm is concentrated on or near its diagonal. Moreover, R-values (or L-values) also reveal possible absolute gaps in the distribution of SVs and provide a substantial information needed in solving rank-revealing (cf. [5, 10]) or (total) least squares problems (cf. [4, 8, 12]).

For the serial Jacobi method, the idea of the pre-processing of matrix A (prior to its SVD) by the QRFCP, optionally followed by the LQF of R-factor, was tested by Drmač and Veselić in [7]. Together with some other techniques (e.g., by sophisticated monitoring of the size of off-diagonal elements for deciding when *not* to apply the Jacobi rotations), they were able to speed up the one-sided serial Jacobi EVD/SVD algorithm significantly.

We extend the idea of a serial preconditioner to the parallel case. We show that its combination with dynamic ordering can lead to a substantial decrease of the number of parallel iteration steps, at least for certain matrices. The best results were achieved for well-conditioned matrices with a multiple minimal SV, where the reduction can be as large as two orders of magnitude. However, due to an inefficient implementation of the QRFCP (LQF) in the current ScaLAPACK library, the reduction of the total parallel execution time is about one order of magnitude (which is certainly still quite promising).

The report is organized as follows. In Section 2 we briefly introduce the parallel two-sided block-Jacobi SVD algorithm with the dynamic ordering. Section 3 is devoted to the variants of pre- and post-processing based on the QRF with CP, optionally followed by the LQF of R-factor. Experimental results on a cluster of personal computers (PCs) are described in Section 4. Here we also discuss the efficiency of dynamic ordering with respect to the reduction of the number of outer parallel iteration steps needed for the convergence. We show experimentally and explain theoretically (at least to some degree), when the dynamic ordering can become inefficient. Finally, Section 5 summarizes achieved results and proposes lines for further research.

Throughout the report, $\|A\|_{\mathrm{F}}$ denotes the Frobenius norm of a matrix A, $a_{:j}$ is the jth column of A, $\|a_{:j}\|$ is its Euclidean norm, and κ is the condition number of A defined as the ratio of

its largest and smallest SV. By $A_{i_1:i_2,\,j_1:j_2}$ we denote the sub-matrix of A consisting of rows $i_1, \ldots, i_2$, and columns $j_1, \ldots, j_2$.

2 Parallel algorithm with dynamic ordering

We mention only briefly basic constituents of the parallel two-sided block-Jacobi SVD algorithm (PTBJA) with dynamic ordering; details can be found in [2]. The parallel algorithm for processor me, $me = 0, 1, \ldots, p-1$, can be written in the form of Algorithm 1.

Algorithm 1 *Parallel block-Jacobi SVD algorithm with dynamic ordering*

1: $U = I_m$
2: $V = I_n$
3: $(i, j) = (2\,me, 2\,me + 1)$
4: **while** $F(A, \ell) \geq \epsilon$ **do**
5: `update`(W)
6: `ReOrderingComp`$(i, j, W, me) \;\rightarrow\; dest1, dest2, tag1, tag2$
7: `copy`$(A_i, U_i, V_i, i) \;\rightarrow\; A_r, U_r, V_r, r$
8: `copy`$(A_j, U_j, V_j, j) \;\rightarrow\; A_s, U_s, V_s, s$
9: `send`$(A_r, U_r, V_r, r, dest1, tag1)$
10: `send`$(A_s, U_s, V_s, s, dest2, tag2)$
11: `receive`$(A_i, U_i, V_i, i, 1)$
12: `receive`$(A_j, U_j, V_j, j, 2)$
13: **if** $F(S_{ij}, \ell) \geq \delta$ **then**
14: ▷ *computation of* X_{ij} *and* Y_{ij} *by SVD of* S_{ij}
15: `SVD`$(S_{ij}) \;\rightarrow\; X_{ij}, Y_{ij}$
16: ▷ *update of block columns*
17: $(A_i, A_j) = (A_i, A_j) \cdot Y_{ij}$
18: $(U_i, U_j) = (U_i, U_j) \cdot X_{ij}$
19: $(V_i, V_j) = (V_i, V_j) \cdot Y_{ij}$
20: **else**
21: $X_{ij} = I_{(m/p)}$
22: **end if**
23: `AllGather`$(X_{ij}, i, j) \;\rightarrow\; XX(t) = (X_{rs}, r, s), t = 0, 1, \ldots, p-1$
24: ▷ *update of block rows*
25: **for** $t = 0$ to $p - 1$ **do**
26: $\begin{pmatrix} A_{ri} & A_{rj} \\ A_{si} & A_{sj} \end{pmatrix} = X^H_{rs,t} \cdot \begin{pmatrix} A_{ri} & A_{rj} \\ A_{si} & A_{sj} \end{pmatrix}$
27: **end for**
28: **end while**

end

When using p processors and the blocking factor $\ell = 2p$, a given matrix A is cut column-wise and row-wise into an $\ell \times \ell$ block structure. Each processor contains exactly two block columns of dimensions $m \times n/\ell$ so that $\ell/2$ SVD subproblems of block size 2×2 are solved in parallel in each iteration step. This tight connection between the number of processors p and the blocking factor ℓ can be released (see [3]). However, our experiments have shown that using $\ell = 2p$ ensures the least total parallel execution time in most cases.

The procedure `ReOrderingComp` (Algorithm 1, step 6) computes the optimal reordering destinations of all block columns residing in a given processor ($dest1$ and $dest2$) and their locations at new position ($tag1$ and $tag2$). The so-called *dynamic* reordering is based on the maximum-weight perfect matching that operates on the $\ell \times \ell$ updated weight matrix W using the elements of $W + W^T$, where $(W + W^T)_{ij} = \|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$. Details concerning the dynamic ordering can be found in [2]. The argument tag provides the matching between the corresponding `send` and `receive` calls.

The kernel operation is the SVD of 2×2 block subproblems

$$S_{ij} = \begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix}, \tag{1}$$

where, for a given pair (i, j), $i, j = 0, 1, \ldots, \ell - 1,\ i \neq j$, the unitary matrices X_{ij} and Y_{ij} are generated such that the product

$$X_{ij}^H \, S_{ij} \, Y_{ij} = D_{ij}$$

is a block diagonal matrix of the form

$$D_{ij} = \begin{pmatrix} \hat{D}_{ii} & 0 \\ 0 & \hat{D}_{jj} \end{pmatrix},$$

where $\hat{D}_{ii}$ and $\hat{D}_{jj}$ are diagonal.

The termination criterion of the entire process is

$$F(A, \ell) = \sqrt{\sum_{i,j=0,\, i \neq j}^{\ell-1} \|A_{ij}\|_{\mathrm{F}}^2} < \epsilon\,, \quad \epsilon \equiv \mathit{prec} \cdot \|A\|_{\mathrm{F}}\,, \tag{2}$$

where ϵ is the required accuracy (measured relatively to the Frobenius norm of the original matrix A), and $prec$ is a chosen small constant, $0 < \mathit{prec} \ll 1$.

The subproblem (1) is solved only if

$$F(S_{ij}, \ell) = \sqrt{\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2} \geq \delta\,, \quad \delta \equiv \epsilon \cdot \sqrt{\frac{2}{\ell\,(\ell - 1)}}\,, \tag{3}$$

where δ is a given subproblem accuracy. It is easy to show that if $F(S_{ij}, \ell) < \delta$ for all $i \neq j$ then $F(A, \ell) < \epsilon$, i.e., the entire algorithm has converged.

After the embedded SVD is computed, the matrices X_{ij} and Y_{ij} of local left and right singular vectors, respectively, are used for the local update of block columns. Then each processor sends

its matrix X_{ij} to all other processors, so that each processor maintains an array of p matrices. These matrices are needed in the orthogonal updates of block rows.

From the implementation point of view, the embedded SVD is computed using the procedure `ZGESVD` from the LAPACK library while matrix multiplications are performed by the procedure `ZGEMM` from the BLAS (Basic Linear Algebra Subroutines). The point-to-point as well as collective communications are realized by the Message Passing Interface (MPI).

3 Variants of pre-processing and post-processing

3.1 QR factorization with column pivoting

As mentioned above, the main idea of pre-processing is to concentrate the Frobenius norm of the whole matrix A towards its diagonal. For this purpose, the QRFCP is applied to A at the beginning of computation. This pre-processing step can be written in the form

$$AP = Q_1 R, \tag{4}$$

where $P \in \mathbb{R}^{n\times n}$ is the permutation matrix, $Q_1 \in \mathbb{C}^{m\times n}$ has unitary columns and $R \in \mathbb{C}^{n\times n}$ is upper triangular. Notice that this is a so-called *economy-sized* QRFCP, where only n unitary columns of orthogonal matrix are computed.

In the second step, the SVD of the matrix R is computed by the PTBJA with dynamic ordering. Since R is upper triangular, one could use here some parallel variant of the Kogbetliantz method (cf. [9]), which preserves the upper triangular form through the whole Jacobi process. However, at this stage, our PTBJA does not include this option, and the upper triangular form of R is lost, in general, after first update of block rows and block columns. Let us denote the SVD of R by

$$R = U_1 \Sigma V_1^H, \tag{5}$$

where $U_1 \in \mathbb{C}^{n\times n}$ and $V_1 \in \mathbb{C}^{n\times n}$ are left and right singular vectors, respectively, and the diagonal matrix $\Sigma \in \mathbb{R}^{n\times n}$ contains n SVs, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$, that are the same as those of A.

In the final step, some post-processing is required to obtain the SVD of A, $A \equiv U\Sigma V^H$. Using Eq. (5) in Eq. (4), one obtains

$$AP = (Q_1 U_1)\Sigma V_1^H,$$

so that

$$U = Q_1 U_1 \quad \text{and} \quad V = PV_1. \tag{6}$$

As can be seen from Eq. (6), the post-processing step consists essentially of one distributed matrix multiplication. Here we assume that the permutation of rows of V_1 can be done without a distributed matrix multiplication, e.g., by gathering V_1 in one processor and exchanging its rows.

3.2 Optional LQ factorization of the R-factor

The second variant of pre- and post-processing starts with the same first step as above, i.e., with the QRFCP of A.

However, in the second step, the LQF of R-factor is computed (without column pivoting), i.e.,

$$R = LQ_2, \tag{7}$$

where $L \in \mathbb{C}^{n\times n}$ is the lower triangular matrix and $Q_2 \in \mathbb{C}^{n\times n}$ is the unitary matrix. This step helps to concentrate the Frobenius norm of R towards the diagonal even more (cf. [7, 11]).

Next, the SVD of L is computed in the third step by our parallel PTBJA with dynamic ordering,

$$L = U_2 \Sigma V_2^H, \tag{8}$$

and, finally, the SVD of the original matrix $A \equiv U\Sigma V^H$ is assembled in the post-processing step, where

$$U = Q_1 U_2 \quad \text{and} \quad V = P(Q_2^H V_2). \tag{9}$$

Hence, the post-processing consists essentially of two distributed matrix multiplications.

To illustrate the effect of pre-processing, Figure 1 depicts the relative block distribution of $\|A\|_{\mathrm{F}}^2$ for a random dense matrix A before and after both pre-processing steps. The QRFCP together with the LQF are clearly able to concentrate more that 99 per cent of $\|A\|_{\mathrm{F}}^2$ into diagonal blocks.

Clearly, the time and space complexity of the second pre-processing variant is higher than of the first one. In general, one can expect some trade-off between the parallel Jacobi algorithm applied to the original matrix A and to the R (L) factor after one (two) distributed factorization(s). If the reduction of the number of outer iteration steps were not large enough, and if the computation of one (two) factorization(s) were not very efficient on a given parallel architecture, it could easily happen that the total parallel execution time needed for the SVD of A would be higher for variants with pre- and post-processing than for the Jacobi algorithm applied directly to A. To test the behavior of both distributed factorizations, we have conducted some numerical experiments that are described next.

4 Implementation and experimental results

We have implemented three variants of the parallel two-sided block-Jacobi SVD algorithm on the cluster of PCs named 'Gaisberg' at the University of Salzburg. The first variant, denoted by [SVD], simply applies the PTBJA to an original matrix A without any pre-processing. The second method, denoted by [QRCP, SVD(R)], first computes the QRF with CP of A and then applies the PTBJA to the R-factor. The computation ends by the post-processing according to Eq. (6). Finally, the third variant, denoted by [QRCP, LQ, SVD(L)], computes the QRF with CP of A, then the LQF (without CP) of the R-factor, and applies the PTBJA to the L-factor that comes out from the second factorization. To get the SVD of an original matrix A, the post-processing step according to Eq. (9) is required.

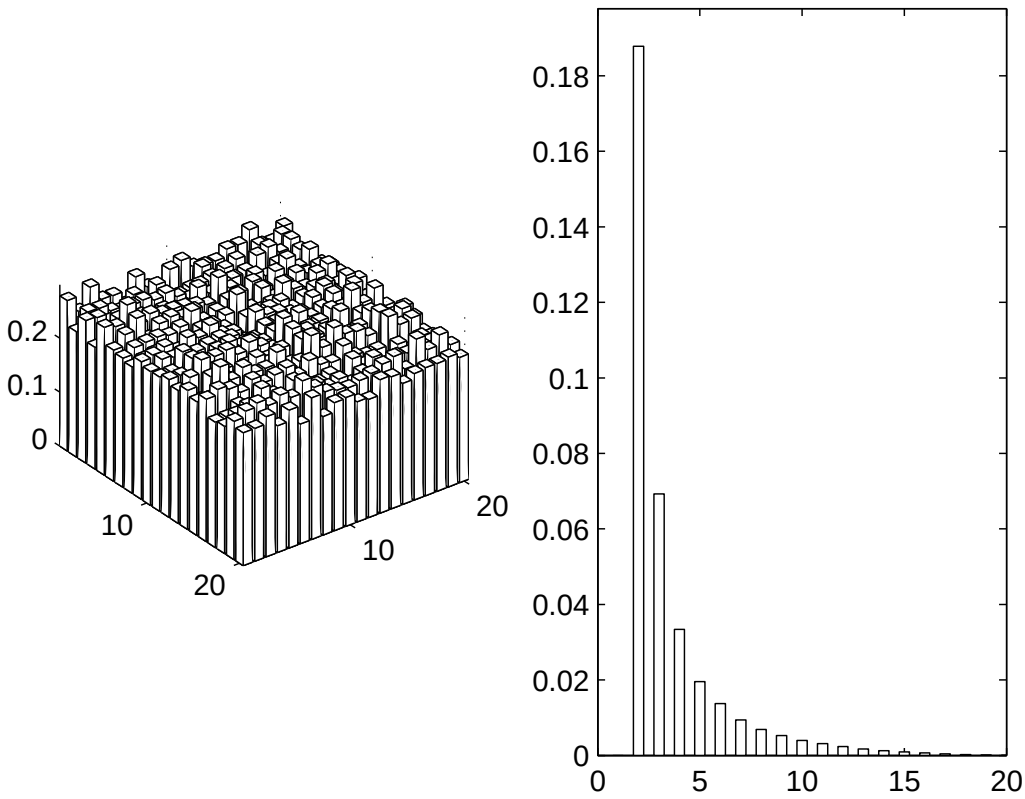


Figure 1: *Left*, relative block distribution (in per cent) of $\|A\|_{\mathrm{F}}^2$. *Right*, the same distribution w.r.t. the individual block sub-diagonals of L-factor after the QRFCP + LQF. The main block diagonal contains 99.6 per cent of $\|A\|_{\mathrm{F}}^2$ (not shown). Random matrix A with $n = 600$, $\ell = 20$, $\kappa = 10$, and with a multiple minimal SV.

The cluster of PCs consisted of 25 nodes arranged in a 5×5 two-dimensional torus. Nodes were connected by the Scalable Coherent Interface (SCI) network; its bandwidth was 385 MB/s and latency $< 4\mu$s. Each node contained 2 GB RAM with two 2.1 GHz ATHLON 2800+ CPUs, while each CPU contained a two-level cache organized into a 64 kB L1 instruction cache, 64 kB L1 data cache and 512 kB L2 data cache.

All computations were performed using the IEEE standard double precision floating point arithmetic with the machine precision $\epsilon_M \approx 1.11 \times 10^{-16}$. By default, the constant $prec = 10^{-13}$ was used for the computation of ϵ and δ (see Eqs. (2) and (3)). The number of processors p was variable, $p = 4$, 8, 24, 40, and depended on the order n of a square real test matrix A, whereby the values $n = 2000$, 4000, 6000 and 8000 have been used.

Matrix elements in all cases were generated randomly, with a prescribed condition number κ and a known distribution of SVs $1 = \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n = 1/\kappa$. More precisely, $A = YDZ^T$, where Y and Z were random orthogonal matrices with their elements from the Gaussian distribution $N(0, 1)$, and D was a diagonal matrix with a prescribed distribution of SVs on its main diagonal.

With respect to κ, there were well-conditioned matrices with $\kappa = 10$ and ill-conditioned matrices

with $\kappa = 10^8$. In all cases, the SVs were contained in the closed interval $[\kappa^{-1}, 1]$, and two types of their distribution were used. In the first distribution, a matrix had a multiple minimal SV with $\sigma_1 = 1$ and $\sigma_2 = \sigma_3 = \cdots = \sigma_n = \kappa^{-1}$. In the second case, the SVs were distributed in the form of a geometric sequence with $\sigma_1 = 1$ and $\sigma_n = \kappa^{-1}$ (i.e., all SVs were distinct, but clustered towards σ_n). It is well known that the SVD of matrices with multiple or clustered SVs, especially in an ill-conditioned case, is harder to compute as compared to the case of well-separated SVs.

Numerical computations were performed using standard numerical libraries, either from local (LAPACK) or distributed (ScaLAPACK) software packages. In particular, the QRFCP and the LQF was implemented by the ScalAPACK's routine `PDGEQPF` and `PDGELQF`, respectively. Point-to-point and collective communication between processors was performed using the communication libraries BLACS (Basic Linear Algebra Communication Subroutines) and MPI.

Experimental results are presented in subsequent tables, the format of which is common for all of them. The first column contains the order of a (square) matrix while the second one denotes the number of processors used in an experiment. Afterwards, the results for individual methods are depicted in the format of two sub-columns per method. The first sub-column contains the number of parallel iteration steps n_{iter} needed for the convergence at given accuracy, and the second sub-column contains the total parallel execution time T_p. Best values of T_p for each matrix order n are depicted in bold.

4.1 Multiple minimal singular value

We begin with results for *well-conditioned matrices with a multiple minimal SV*, which are summarized in Table 1. Its last two columns contain ratios of n_{iter} and T_p for two methods

Table 1: Performance for $\ell = 2p$, $prec = 10^{-13}$, $\kappa = 10$, multiple minimal SV

		[SVD]		[QRCP, SVD(R)]			
n	p	n_{iter}	T_p[s]	n_{iter}	T_p[s]	[Ratio n_{iter}]	[Ratio T_p]
2000	4	170	1778.5	3	**91.0**	56.7	19.5
4000	8	452	6492.5	4	**307.7**	113.0	21.1
6000	24	1817	5367.6	6	**369.3**	302.8	14.5
8000	40	3289	7709.9	7	**1273.2**	469.0	6.1

studied – namely, [SVD] and [QRCP, SVD(R)]. The reduction of n_{iter} using the QRF with CP is enormous (two orders of magnitude), and the value of n_{iter} for the [QRCP, SVD(R)] method increases only slowly with an increasing n. Thus, considering the reduction of n_{iter} alone, the QRF with CP plays the role of an almost ideal preconditioner in this case. It is also clear that employing the additional LQF of R-factor is not necessary because the QRF with CP has already reduced n_{iter} substantially.

An interesting observation is that of *decrease* of T_p when going from $n = 4000$ to $n = 6000$ while n_{iter} increases. This can be explained as follows. For $n = 4000$ and $p = 8$, the size of

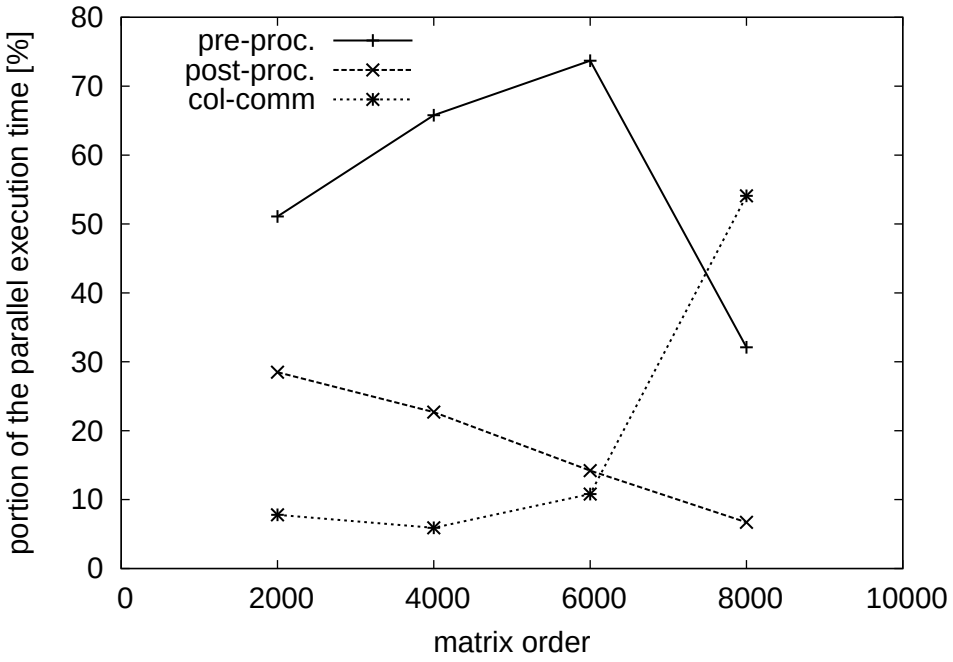


Figure 2: Portion of T_p needed in the pre-processing, post-processing and collective communication using the [QRCP, SVD(R)] for $\kappa = 10$ and the multiple minimal SV.

a 2×2 block SVD subproblem solved in each processor is 500×500, whereas for $n = 6000$ and $p = 24$ it is 250×250. Since the time complexity of the SVD is $O(n^3)$, halving the dimension results in eight-fold speedup of the SVD computation in each parallel iteration step. However, since the blocking factor ℓ increases from $\ell = 16$ to $\ell = 48$, the number of parallel iteration steps also increases (cf. [2, 3]); in our case approximately four times, from $n_{\text{iter}} = 452$ to $n_{\text{iter}} = 1817$. Hence, the total time spent in the SVD of sub-problems is approximately halved. This leads, together with the increasing cost of pre- and post-processing (see Figure 2 below) and of collective as well as point-to-point communication, to the observed reduction of T_p. We note that similar reduction of T_p by transition from $n = 4000$ to $n = 6000$ has been observed also for other condition numbers and/or distributions of SVs in tables below.

In contrast to n_{iter}, savings in T_p are of one order of magnitude less. The reason of this behavior can be deduced from Figure 2. For all matrix orders, the pre-processing step (the QRF with CP) takes more than 30 per cent of T_p, for matrix orders up to 6000 even more than 50 per cent. This means that the QRF with CP, as currently implemented in the ScaLAPACK library, is not very efficient (at least for our cluster of PCs). In other words, the substantial decrease of n_{iter} is not sufficient for a comparable decrease of T_p when another portion of parallel computation is not implemented efficiently.

The portion of T_p spent in collective communication includes the gathering of matrices U, Σ and V on one processor after finishing the computation. For $n = 8000$ and the number of processors $p = 40$ this gathering alone suddenly jumps in time complexity, so that the whole collective communication takes more than 50 per cent of T_p. It is possible that the operating system takes another algorithm for gathering columns of double precision floats of length 8000 than for smaller vectors. On the other hand, the distributed matrix multiplication needed in the post-processing step is implemented quite efficiently. Its time complexity actually *decreases*

with the matrix order, and only about 7 per cent of T_p is needed for its completion for $n = 8000$. Similar results regarding the profiling of pre- and post-processing steps were observed also in other experiments.

Results for *ill-conditioned matrices with a multiple minimal SV* are depicted in Table 2. When

Table 2: Performance for $\ell = 2p$, $prec = 10^{-13}$, $\kappa = 10^8$, multiple minimal SV

		[SVD]		[QRCP, SVD(R)]		[QRCP, LQ, SVD(L)]	
n	p	n_{iter}	T_p[s]	n_{iter}	T_p[s]	n_{iter}	T_p[s]
2000	4	59	832.9	11	163.5	7	**153.1**
4000	8	191	3308.8	26	**547.9**	15	609.8
6000	24	819	2791.8	72	**632.6**	47	842.9
8000	40	1512	5169.6	125	1811.0	79	**1782.5**

compared with well-conditioned matrices (see Table 1), one can conclude that for the [QRCP, SVD(R)] method the number of parallel iteration steps n_{iter} depends much more strongly on n. The additional LQF of the R-factor helps to decrease further the number of parallel iteration steps, but savings in the total parallel execution time are not proportional due to the large time complexity of *two* distributed factorizations during pre-processing.

4.2 Geometric sequence of singular values

In the following experiments, the SVs were distributed in the form of a *geometric* sequence in the interval $[\kappa^{-1}, 1]$ with $\sigma_1 = 1$ and $\sigma_n = \kappa^{-1}$, i.e., they were distinct but clustered towards σ_n.

Results for *well-conditioned matrices* are depicted in Table 3. As can be seen, neither the

Table 3: Performance for $\ell = 2p$, $prec = 10^{-13}$, $\kappa = 10$, geometric sequence of SVs

		[SVD]		[QRCP, SVD(R)]		[QRCP, LQ, SVD(L)]	
n	p	n_{iter}	T_p[s]	n_{iter}	T_p[s]	n_{iter}	T_p[s]
2000	4	41	**665.5**	39	699.3	36	758.1
4000	8	102	**2486.6**	93	2594.4	84	2580.8
6000	24	356	**1570.0**	323	1866.2	283	1828.5
8000	40	621	2785.2	565	2827.4	492	**2566.9**

[QRCP, SVD(R)] method nor the [QRCP, LQ, SVD(L)] one can reduce substantially the total parallel execution time T_p, since n_{iter} is reduced by at most 10 – 20 per cent, which is not enough.

Table 4 depicts the experimental results for *ill-conditioned* matrices. Using the [QRCP, SVD(R)] method in this case leads to the reduction of n_{iter} by only 5 – 30 per cent, which is *not* enough

Table 4: Performance for $\ell = 2p$, $prec = 10^{-13}$, $\kappa = 10^8$, geometric sequence of SVs

		[SVD]		[QRCP, SVD(R)]		[QRCP, LQ, SVD(L)]	
n	p	n_{iter}	T_p[s]	n_{iter}	T_p[s]	n_{iter}	T_p[s]
2000	4	44	520.6	43	565.9	19	**315.0**
4000	8	137	2132.9	114	2042.7	45	**1067.2**
6000	24	559	1883.4	527	2136.7	154	**1186.7**
8000	40	1025	3583.9	969	3929.3	260	**2034.4**

to reduce the total parallel execution time T_p. In fact, for $n = 6000$ and $n = 8000$, the total parallel execution time is even higher than for the SVD of A alone. Therefore, the application of the [QRCP, LQ, SVD(L)] method is required to decrease n_{iter} further. Consequently, T_p is decreased albeit the savings, as compared to the direct SVD of A, are only around 40 – 50 per cent.

When comparing the results in Tables 3 and 4, it turns out that the [QRCP, LQ, SVD(L)] method is far more successful in reducing n_{iter} and T_p for ill-conditioned matrices than for well-conditioned ones. Since in both cases the SVs decrease *gradually* from 1 to κ^{-1} (i.e., there are no large absolute gaps in the distribution of SVs), there exists no significant layered structure in the L-factor in any of these two cases (with respect to the layered structure of the R- or L-factor, see the discussion in subsection 4.3 below). Hence, the main reason for the faster convergence in the case of ill-conditioned matrices seems to be the more concentrated Frobenius norm near the diagonal of L at the beginning of iterations. This somewhat surprising fact is explained in next subsection.

4.3 Structure of R-factor (L-factor) and its impact on convergence rate

There are two deciding parameters that influence the convergence rate of the PTBJA after applying the QRFCP, optionally followed by the LQF of R-factor. The first parameter is a degree of concentration of the overall Frobenius norm of A towards the diagonal of R (L), while the second one is the distribution of the remaining off-diagonal Frobenius norm in R (L). To simplify discussion, the case of R-factor is treated in detail, and some remarks with respect to the L-factor are added when appropriate.

Let us consider first a concentration of the Frobenius norm towards the diagonal of R. Recall that the QRFCP is computed by applying the unitary Householder transformations to the columns of A from left to right that do not change the Frobenius norm of any column. Since $\|A\|_{\text{F}}^2 = \|R\|_{\text{F}}^2 = \sum_{j=1}^{n} \sigma_j^2$, the degree of concentration depends on how close the absolute values of diagonal elements of R approximate the SVs. As shown in [11], the R-values *underestimate* and *overestimate* the largest and smallest SVs, respectively.

For a well-conditioned matrix A, *all* its SVs contribute to $\|A\|_{\text{F}}^2$ significantly. In contrast, taking an ill-conditioned matrix A, it can happen that only σ_1 together with few largest SVs play an

important role in computing $\|A\|_{\mathrm{F}}^2$. Consequently, underestimating σ_1 (or few largest SVs) by top R-values means that the Frobenius norm of A can be concentrated on diagonal of R to a *lesser* degree for the ill-conditioned matrices than for the well-conditioned ones. Thus, one can expect a greater number of parallel iteration steps in the former case than in the latter.

However, the situation may change when going from the R- to L-factor, because $|l_{11}|$ usually approximates σ_1 much more exactly than $|r_{11}|$ (and this is also true for few other SVs next to σ_1 – cf. [11]). Thus it follows that after the LQF of R-factor the Frobenius norm of A can be concentrated on the diagonal of L to a *greater* degree for the ill-conditioned matrices than for the well-conditioned ones. This helps to explain the experimental results for geometrically distributed SVs, where the application of the [QRCP, LQ, SVD(L)] method was more successful for ill-conditioned matrices (compare Tables 3 and 4).

Second important parameter for the rate of convergence of the PTBJA is a distribution of the remaining off-diagonal Frobenius norm in R-factor. As is well known, after performing the QRF with column pivoting on matrix A, each diagonal element r_{ii}, $1 \leq i \leq n$, of resulting R-factor satisfies the set of inequalities

$$|r_{ii}|^2 \geq \sum_{k=i}^{j} |r_{kj}|^2, \quad j = i+1,\, i+2, \ldots, n.$$

Now assume that the R-values approximate the SVs of A reasonably well, and that there is a large absolute gap in the distribution of SVs – i.e., there exists an index i_1 such that

$$\sigma_1 \geq \ldots \geq \sigma_{i_1} \gg \sigma_{i_1+1} \geq \sigma_{i_1+2} \geq \ldots \geq \sigma_n.$$

Since the values $|r_{ii}|$ are ordered non-increasingly, $|r_{i_1,\,i_1}| \gg |r_{i_1+1,\,i_1+1}|$, and

$$\|R\|_{\mathrm{F}}^2 = \sum_{j=1}^{n} \|r_{:j}\|^2 = \sum_{j=1}^{n} \sigma_j^2 \geq \sigma_{i_1}^2 \gg \sigma_{i_1+1}^2 \approx |r_{i_1+1,\,i_1+1}|^2.$$

Consequently, the *bottom* sub-matrix $R_{i_1+1:n,\,i_1+1:n}$ can contain only 'small' elements (i.e., those in absolute value smaller or equal to $|r_{i_1+1,\,i_1+1}|$), while the *top* sub-matrix $R_{1:i_1,\,1:n}$ must contain 'large' elements in almost all sub-columns (in order to preserve $\|R\|_{\mathrm{F}} = \|A\|_{\mathrm{F}}$). Here the ratio 'large/small' can be estimated by $|r_{i_1,\,i_1}|/|r_{i_1+1,\,i_1+1}| \approx \sigma_{i_1}/\sigma_{i_1+1}$.

Thus the R-factor can be partitioned row-wise in two non-balanced parts. When imposing any block structure on R using the blocking factor ℓ of the block-Jacobi method, it follows that any off-diagonal block from the upper part will have much larger Frobenius norm than any off-diagonal block from the lower part.

Notice that the above considerations can be easily extended (by induction from the bottom of R to the top) to any number of large gaps in the distribution of SVs.

This *layered* block-row distribution of the off-diagonal Frobenius norm of R-factor causes the loss of efficiency in the dynamic ordering. Consider results from Table 2 for *ill-conditioned* matrices with a *multiple minimal* SV. Since $\sigma_1 = 1$ and $\sigma_i = 10^{-8}$ for all i, $2 < i \leq n$, we have $i_1 = 1$, and the off-diagonal Frobenius norm of R will be concentrated in its first block row *regardless* to n and ℓ. This situation is depicted in Figure 3. The complete graph with

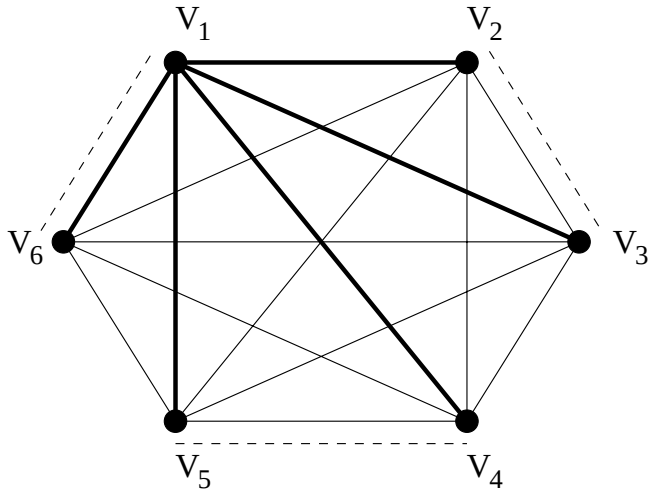


Figure 3: Structure of the complete edge-weighted graph when the dynamic ordering is inefficient. Major part of the off-diagonal Frobenius norm of R is concentrated in the first block-row, so that all edges incident with the vertex V_1 are 'heavy' (thick lines), other are 'light' (thin lines). Since a perfect matching (dashed lines) contains exactly one edge incident with each vertex, the decrease of the off-diagonal Frobenius norm in this parallel iteration step is very small relatively to its overall actual value.

six vertices corresponds to the 6×6 block partition of R. Its edge $[V_i, V_j]$ is weighted by $\|R_{ij}\|_{\mathrm{F}}^2 + \|R_{ji}\|_{\mathrm{F}}^2$. All edges incident with the vertex V_1 are 'heavy' (i.e., their weights are large), while other edges are 'light' (i.e., their weights are small). Since the maximum-weight perfect matching can only use exactly one edge incident with V_1, the reduction of the off-diagonal Frobenius norm will be relatively small, and a greater number of iteration steps may be required for the convergence of the whole algorithm.

However, ill-conditioning alone is *not* sufficient for a layered structure of R to occur. Consider an *ill-conditioned* matrix A (say, $\kappa = 10^8$,) with a *multiple maximal* SV, i.e., $\sigma_i = 1$ for all i, $1 \leq i \leq n-1$, and $\sigma_n = 10^{-8}$. In this case, σ_n alone is of no significant influence, and all off-diagonal blocks will have more or less the same Frobenius norm. This is advantageous from the point of view of dynamic ordering, because the maximum-weight perfect matching can substantially reduce the off-diagonal Frobenius norm in one iteration step. Hence, even for ill-conditioned matrices, the off-diagonal Frobenius norm of R-factor can be distributed uniformly and the PTBJA may converge in a relatively smaller number of parallel iteration steps. This conclusion has been confirmed by additional numerical experiments, where the results for ill-conditioned matrices with a multiple maximal SV were almost identical to those presented in Table 1 for well-conditioned matrices with a multiple minimal SV. In particular, n_{iter} was of order $O(1)$ over the whole range of n.

In the case of *well-conditioned* matrices, a relatively small value of κ precludes the existence of large gap(s) in the distribution of SVs, so that the off-diagonal Frobenius norm of R-factor is distributed more or less evenly. This helps to explain excellent convergence properties of the PTBJA after pre-processing for well-conditioned matrices with a multiple minimal SV (see results in Table 1).

After an optional LQF of R, the resulting L-factor is usually much closer to a diagonal matrix

than R (cf. [7, 11]). Despite a high concentration of the Frobenius norm towards the diagonal of L, the layered block structure may still be present, so that a faster convergence of the PTBJA is hindered. This effect is clearly observed for ill-conditioned matrices with a multiple minimal SV (see Table 2). Here n_{iter} did *not* decrease significantly after applying the LQF of R-factor, although, at the beginning of iterations, the diagonal of L contained about 99 per cent of $\|A\|_{\text{F}}^2$.

5 Conclusions

The traditional usage of the QRF before the EVD/SVD computation emphasizes the reduction of dimension of an original matrix A whenever $m \gg n$. Even more importantly, the QRF with column pivoting, optionally followed by the LQF of R-factor, concentrates the Frobenius norm towards a diagonal, and reveals the form of the distribution of SVs. While the former fact can be very advantageous from the point of dynamic ordering used in the parallel two-sided block-Jacobi SVD algorithm, a layered block-row (block-column) structure of the R-factor (L-factor) can hinder the efficiency of dynamic ordering. This is the case for ill-conditioned matrices with large absolute gap(s) between SVs *and* with a special distribution of SVs (including their multiplicity), when a vast majority of the off-diagonal Frobenius norm is concentrated in one or few block-rows of R (block-columns of L).

Our experiments have shown that the largest savings, both in n_{iter} and T_p, as compared to the simple block-Jacobi SVD, can be observed for well-conditioned matrices with a multiple minimal SV. In this case, the QRF with CP and the subsequent SVD of R-factor is the method of choice. For ill-conditioned matrices with a geometric distribution of SVs, the additional pre-processing step (the LQF of R-factor) is required to substantially reduce n_{iter}. Consequently, T_p is also reduced, but only mildly.

Further savings, both in memory and T_p, can be achieved by exploiting the Kogbetliantz-like strategy when computing the SVD of R-factor (L-factor). Recall that the Kogbetliantz algorithm preserves the upper (lower) triangular form of R (L) during the whole Jacobi process, so that the memory required for storing them is halved as compared to our current implementation. To decrease T_p, one can think of time spent in orthogonal updates of block rows and block columns, whereby only exactly one block row (block column) is of length n and all other are shorter. Using the currently implemented block column-wise data distribution of R (L), the orthogonal updates of block columns would remain local to processors (albeit non-balanced due to the variable lengths of block columns). However, the communication complexity in the orthogonal updates of block rows can be decreased, since not all processors store the corresponding parts of *all* block rows anymore. It is an interesting question if there exists another data distribution of the R-factor (L-factor) for a parallel, balanced implementation of the Kogbetliantz algorithm on a block level.

The current *main bottleneck* of the proposed preconditioning is the high time complexity of the distributed QRF with CP, and of the distributed LQF, as implemented in the current version of ScaLAPACK. This is plainly seen in the case of well-conditioned matrices with geometrically distributed SVs, where the reduction of n_{iter} is not sufficient for decreasing T_p. It is an open and interesting question whether this state of affairs can be improved. We also plan to extend numerical experiments to larger matrices of order 10^5–10^6.

References

[1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst, Templates for the solution of algebraic eigenvalue problems: A practical guide, First ed., SIAM, Philadelphia, 2000.

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

[3] M. Bečka and G. Okša, On variable blocking factor in a parallel dynamic block-Jacobi SVD algorithm, Parallel Computing 29 (2003) 1153-1174.

[4] Å. Björck, Numerical methods for least squares problems, First ed., SIAM, Philadelphia, 1996.

[5] T. F. Chan, Rank revealing QR factorizations, Linear Algebra Appl. 88/89 (1987) 67-82.

[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 and K. Veselić, New fast and accurate Jacobi SVD algorithm, 2004, in preparation.

[8] G. H. Golub, Numerical methods for solving least squares problems, Numer. Math. 7 (1965) 206-216.

[9] V. Hari and J. Matejaš, Scaled iterates by Kogbetliantz method, Proc. First Conf. Applied Mathematics and Computation, Dubrovnik, Croatia, September 13-18, 1999, 1-20.

[10] Y. P. Hong and C.-T. Pan, Rank-revealing QR factorizations and the singular value decomposition, Math. Comp. 58 (1992) 2 13-232.

[11] G. W. Stewart, The QLP approximation to the singular value decomposition, SIAM J. Sci. Comput. 20 (1999) 1336-1348.

[12] S. Van Huffel and J. Vandewalle, The total least squares problem, First ed., SIAM, Philadelphia, 1991.