Proceedings of ALGORITMY 2005
pp. 202–211

PRECONDITIONING 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 the form of spectrum. However, the gain in speed, as measured by the total parallel execution time, depends decisively on how efficient is the implementation of the distributed QR and LQ factorizations on a given parallel architecture.

Key words. QR factorization with column pivoting, LQ factorization, singular value decomposition, two-sided block-Jacobi method, parallel computation, message passing interface, cluster of personal computers

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

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 [4], it can achieve the 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 serial 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 random, dense matrices of orders 2000 – 10000. In general, however, this is not enough to make the method competitive with faster (albeit less accurate) algorithms based on the bi-diagonalization.

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 computation. Hence, it is hoped for that the concentration of the Frobenius norm towards the diagonal might decrease the number of outer iteration steps substantially.

In the serial case, the idea of a preconditioner was proposed and tested by Drmač and Veselić in [5]. Their preconditioner consists of the QR factorization (QRF) with

[*]This work was supported by VEGA Grant no. 2/4136/24 from the Scientific Grant Agency of Slovak Republic and Slovak Academy of Sciences.

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

[‡]Institute for Scientific Computing, University of Salzburg, Salzburg, Austria

the complete column pivoting (CP), optionally followed by the LQ factorization (LQF) of the R-factor. Both these methods help to concentrate the Frobenius norm towards the diagonal.

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 singular value.

The paper 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 the pre- and post-processing based on the QRF with CP, optionally followed by the LQF of the R-factor. Experimental results on a cluster of personal computers (PCs) are described in Section 4. Finally, Section 5 summarizes achieved results and proposes lines for further research.

2. Parallel algorithm with dynamic ordering.

For the sake of consistency, we briefly mention basic constituents of the parallel two-sided block-Jacobi SVD algorithm 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 2.

When using the blocking factor $\ell = 2p$, 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 2, step 6) computes the optimal reordering destinations of all block columns residing in a given processor (*dest*1 and *dest*2) and their locations at new position (*tag*1 and *tag*2). 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 $(W + W^T)_{ij} = \|A_{ij}\|_F^2 + \|A_{ji}\|_F^2$. 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

$$S_{ij} = \begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix}, \tag{2.1}$$

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

$$F(A, \ell) = \sqrt{\sum_{i,j=0,\, i \neq j}^{l-1} \|A_{ij}\|_F^2} < \epsilon\,, \quad \epsilon \equiv prec \cdot \|A\|_F\,, \tag{2.2}$$

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:        $\begin{pmatrix} A_{ri} & A_{rj} \\ A_{si} & A_{sj} \end{pmatrix} = X_{rs,t}^H \cdot \begin{pmatrix} A_{ri} & A_{rj} \\ A_{si} & A_{sj} \end{pmatrix}$
27:    **end for**
28: **end while**
```

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

A subproblem (2.1) is solved only if

$$F(S_{ij}, \ell) = \sqrt{\|A_{ij}\|_F^2 + \|A_{ji}\|_F^2} \geq \delta\,, \quad \delta \equiv \epsilon \cdot \sqrt{\frac{2}{\ell\,(\ell - 1)}}\,, \tag{2.3}$$

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

After the embedded SVD is computed (step 15), the matrices X_{ij} and Y_{ij} of local left and right singular vectors, respectively, are used for the local update of block columns (steps 16–22). In the procedure AllGather (step 23), each processor sends its matrix X_{ij} to all other processors, so that each processor maintains an array (denoted by XX) of p matrices. These matrices are needed in the orthogonal updates of block rows (steps 24–27).

From the implementation point of view, the embedded SVD is computed using the procedure ZGESVD from the LAPACK library while the matrix multiplications are performed by the procedure `ZGEMM` from the BLAS (Basic Linear Algebra Subroutines). The point-to-point (steps 9–12) as well as collective (step 23) communications are realized by the Message Passing Interface (MPI).

3. Variants of pre-processing and post-processing.

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

$$AP = Q_1 R, \tag{3.1}$$

where $P \in \mathbb{R}^{n\times n}$ is the permutation matrix, $Q \in \mathbb{C}^{m\times n}$ has unitary columns and $R \in \mathbb{C}^{n\times n}$ is upper triangular. Notice that this is a so-called *economy-sized* QRF with CP, where only n unitary columns of orthogonal matrix are computed. It follows from the definition of the complete column pivoting that the upper triangular matrix R is diagonally dominant, so that its Frobenius form is indeed concentrated near the diagonal.

In the second step, the SVD of the matrix R is computed by Algorithm 2. Since R is upper triangular, one could use here some parallel variant of the Kogbetliantz method, which preserves the upper triangular form through the whole Jacobi process. However, at this stage, our two-sided block-Jacobi algorithm does not include this option, and the upper triangular form of R is in general lost after first update of block rows and block columns. Let us denote the SVD of R by

$$R = U_1 \Sigma V_1^H, \tag{3.2}$$

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 singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ 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. (3.2) in Eq. (3.1), one obtains

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

so that

$$U = Q_1 U_1 \quad \text{and} \quad V = PV_1. \tag{3.3}$$

As can be seen from Eq. (3.3), 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 according to the permutation P.

3.2. Optional LQF of the R-factor. The second variant of pre- and post-processing starts with the same first step as above, i.e., with the QRF of A using the complete CP.

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

$$R = LQ_2, \tag{3.4}$$

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.

Next, the SVD of L is computed in the third step by our parallel two-sided block-Jacobi algorithm with dynamic ordering,

$$L = U_2 \Sigma V_2^H, \tag{3.5}$$

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

$$U = Q_1 U_2 \quad \text{and} \quad V = P(Q_2^H V_2). \tag{3.6}$$

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

To illustrate the effect of both steps of pre-processing, Figure 3.1 depicts the relative block distribution of the Frobenius norm of a random dense matrix A before and after both pre-processing steps. The QRF with CP and the LQF are clearly able

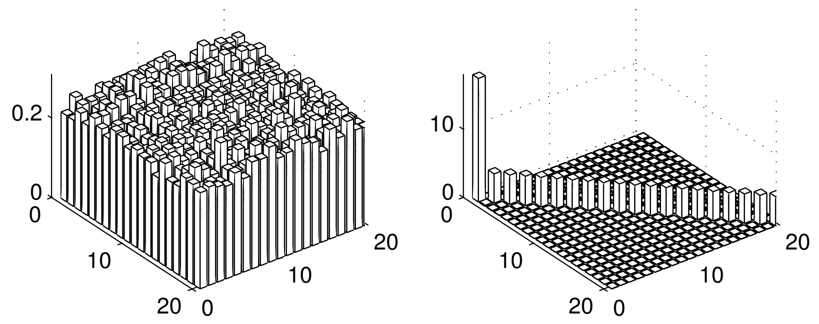

FIG. 3.1. *Relative block distribution (in per cent) of the Frobenius norm of an original matrix A (left) and after the QRF with CP + LQF (right). Random matrix A with $n = 600$, $\ell = 20$, $\kappa = 10$, and with a multiple minimal singular value.*

to concentrate more that 80 percent of the Frobenius norm into diagonal blocks.

Figure 3.2 brings the same information as Figure 3.1, but this time for the *geometrically* distributed singular values. When comparing both figures, it is clearly seen that the QRF with CP together with the subsequent LQF not only concentrate the Frobenius norm towards the diagonal, but also *reveal the spectral shape* of A. This can be very useful for matrices with their spectra not known *a priori*.

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

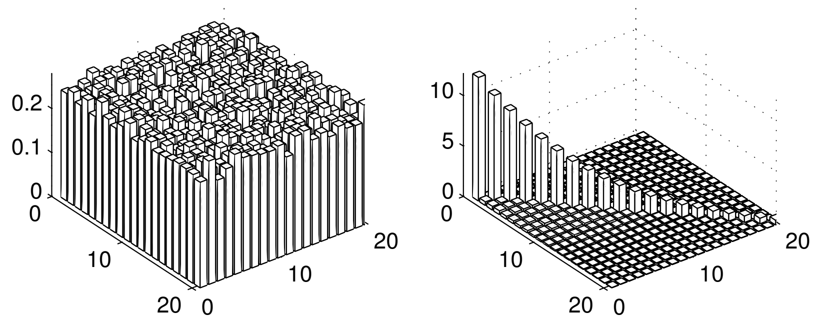

FIG. 3.2. *Relative block distribution (in per cent) of the Frobenius norm of an original matrix A (left) and after the QRF with CP + LQF (right). Random matrix A with* $n = 600$, $\ell = 20$, $\kappa = 10$, *and with singular values forming the geometric sequence in the interval* $[10^{-1}, 1]$, *where* $\sigma_1 = 1$ *and* $\sigma_{600} = 10^{-1}$.

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 after applying one (two) factorization(s) 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, Salzburg, Austria.

The first variant, denoted by [SVD], simply applies Algorithm 2 to an original matrix A without any preconditioning. The second method, denoted by [QRCP, SVD(R)], first computes the QRF with CP of A and then applies Algorithm 2 to the R-factor. The computation ends by the post-processing according to Eq. (3.3). 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 Algorithm 2 to the L-factor that comes out from the final factorization. To get the SVD of an original matrix A, the post-processing step according to Eq. (3.6) is required.

The cluster of PCs consists of 25 nodes arranged in a 5×5 two-dimensional torus.

Nodes are connected by the Scalable Coherent Interface (SCI) network; its bandwidth is 385 MB/s and latency $< 4\mu$s. Each node contains 2 GB RAM with two 2.1 GHz ATHLON 2800+ CPUs, while each CPU contains 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 preformed 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. (2.2) and (2.3)). The number of processors p was variable, $p = 4, 8, 24, 40$, and depended on the order n of a (square) test matrix A, which covered the range $n = 2000$, 4000, 6000 and 8000.

Real matrix elements in all cases were generated randomly using the Gaussian distribution $N(0, 1)$ with a prescribed condition number κ and a known distribution of singular values $1 = \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n = 1/\kappa$. With respect to κ, there were well-conditioned matrices with $\kappa = 10$ and ill-conditioned matrices with $\kappa = 10^8$. In all cases, the singular values 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 singular value with $\sigma_1 = 1$ and $\sigma_2 = \sigma_3 = \cdots = \sigma_n = \kappa^{-1}$. In the second case, the singular values were distributed in the form of a geometric sequence with $\sigma_1 = 1$ and $\sigma_n = \kappa^{-1}$ (i.e., all singular values were distinct, but clustered towards σ_n).

Numerical computations were performed using standard numerical libraries, either from local (LAPACK) or distributed (ScaLAPACK) software packages. In particular, the QRF with CP 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.

We begin with results for *well-conditioned matrices with a multiple minimal singular value*, which are summarized in Table 4.1. Its last two columns contain ratios

TABLE 4.1
Performance for $\ell = 2p$, prec $= 10^{-13}$, $\kappa = \mathbf{10}$, multiple minimal sing. value.

		[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

of n_{iter} and T_p for two methods 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

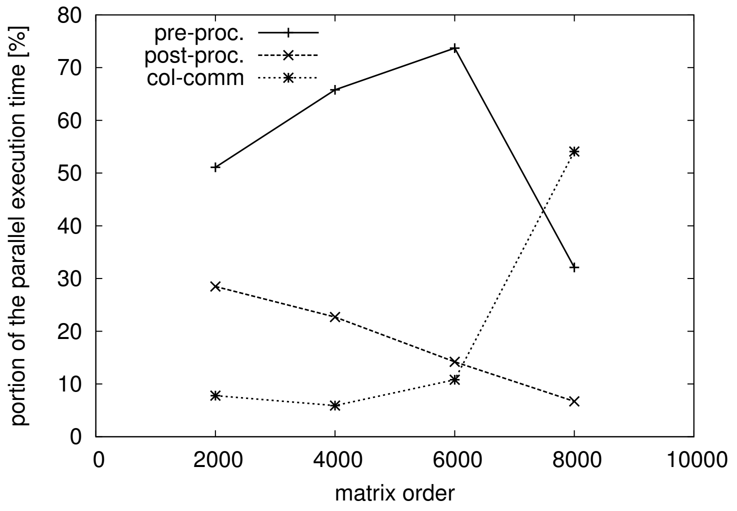

FIG. 4.1. *Portion of T_p needed in the pre-processing, post-processing and collective communication using the [QRCP, SVD(R)] for $\kappa = 10$ and the multiple minimal singular value.*

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 $n_{\rm iter}$ substantially.

However, savings in T_p are of one order of magnitude less than those in $n_{\rm iter}$. The reason of this behavior can be deduced from Figure 4.1. 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 $n_{\rm iter}$ is not sufficient 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 for other experiments.

Results for *ill-conditioned matrices with a multiple minimal singular value* are depicted in Table 4.2. When compared with well-conditioned matrices (see Table 4.1), one can conclude that for the [QRCP, SVD(R)] method the number of parallel iteration steps $n_{\rm iter}$ 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.

Finally, we present the results for *ill-conditioned matrices* in Table 4.3 where the

TABLE 4.2
Performance for $\ell = 2p$, prec $= 10^{-13}$, $\kappa = \mathbf{10^8}$, multiple minimal sing. value.

		[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

singular values form the *geometric* sequence in the interval $[10^{-8}, 1]$ with $\sigma_1 = 1$ and $\sigma_n = 10^{-8}$.

TABLE 4.3
Performance for $\ell = 2p$, prec $= 10^{-13}$, $\kappa = \mathbf{10^8}$, geometric sequence of sing. values.

		[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

Hence all singular values are distinct, but they are clustered towards σ_n. 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 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 *substantially* decrease n_{iter}. Consequently, T_p is decreased albeit the savings, as compared to the direct SVD of A, are only around $40 - 50$ per cent.

5. Conclusions.

When evaluating the preconditioner applied to the computation of the SVD of a general, dense matrix A by the parallel block-Jacobi algorithm, one has to consider at least two measures: the number of parallel iteration steps needed for the convergence at a given accuracy, and the total parallel execution time. The first measure is more bias-free than the second one, especially when one computes on a parallel system, which is not stand-alone, and uses some vendor's supplied numerical and communication libraries as 'black boxes'.

Our experiments show that the largest savings, both in n_{iter} and T_p, as compared to the simple Jacobi SVD, can be observed for well-conditioned matrices with a multiple minimal singular value. 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 singular values, 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.

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. It is an open and interesting question whether this state of affairs can be improved. We also plan to extend numerical experiments to the larger matrix dimensions of order $10^5 - 10^6$. However, with the current cluster of PCs, it will not be possible to gather such huge dense matrices on one processor alone anymore. Hence, all preparation, computation and evaluation will have to be performed in a fully distributed manner.

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), pp. 243–262.

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

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

[5] Z. Drmač and K. Veselić, *New fast and accurate Jacobi SVD algorithm*, 2004, in preparation.