Parallel Numerics '05, 15-24 M. Vajteršic, R. Trobec, P. Zinterhof, A. Uhl (Eds.)
Chapter 2: Matrix Algebra ISBN 961-6303-67-8

Preconditioned Parallel Block–Jacobi SVD Algorithm

Gabriel Okša[1]*, Marián Vajteršic [2]

[1]Institute of Mathematics, Dept. of Informatics
Slovak Academy of Sciences, Bratislava, Slovakia

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

We show 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 in the parallel block-Jacobi SVD algorithm, whereby the details depend on the condition number and on the shape of spectrum, including the multiplicity of singular values. Best results were achieved for well-conditioned matrices with a multiple minimal singular value, where the number of parallel iteration steps has been reduced by two orders of magnitude. 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. 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 [3], 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, serial or parallel – belongs to the slowest known algorithms for computing the

*Corresponding author. E-mail: Gabriel.Oksa@savba.sk

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 10^3 – 10^4. In general, however, this is not enough to make the method competitive with faster (albeit possibly less accurate) algorithms based on the bi-diagonalization.

One way, how to further speed up convergence of the parallel two-sided block-Jacobi SVD algorithm 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. For the serial Jacobi method, the idea of the pre-processing of matrix A (prior to its SVD) by the QR factorization with column pivoting (QRFCP), optionally followed by the LQ factorization (LQF) of R-factor, was tested by Drmač and Veselić in [4]. 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 singular value, where the reduction can be as large as two orders of magnitude. However, due to the 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.

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 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 parallel iteration steps needed for convergence.

Finally, Section 5 summarizes achieved results and proposes lines for further research.

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

2 Parallel algorithm with dynamic ordering

We only briefly mention basic constituents of the parallel two-sided block-Jacobi SVD algorithm (PTBJA) with dynamic ordering; details can be found in [2].

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.

At the beginning of each parallel iteration step, it is necessary to map one 2×2 block SVD subproblem to each of p processors. This can be achieved by some type of ordering. The so-called *dynamic* ordering 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 termination criterion of the entire process is

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

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

A local 2×2 block subproblem is solved only if

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

where δ is a given subproblem accuracy.

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

$$AP = Q_1 R, \tag{3}$$

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. Let us denote the SVD of R by

$$R = U_1 \Sigma V_1^H, \tag{4}$$

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. (4) in Eq. (3), one obtains

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

so that

$$U = Q_1 U_1 \quad \text{and} \quad V = PV_1. \tag{5}$$

As can be seen from Eq. (5), 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.,

$$R = LQ_2, \tag{6}$$

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. [4, 5]).

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

$$L = U_2 \Sigma V_2^H, \tag{7}$$

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

$$U = Q_1 U_2 \quad \text{and} \quad V = P(Q_2^H V_2). \tag{8}$$

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

To illustrate the effect of both steps of pre-processing, Figure 1 depicts the relative block distribution of the square of Frobenius norm of a random

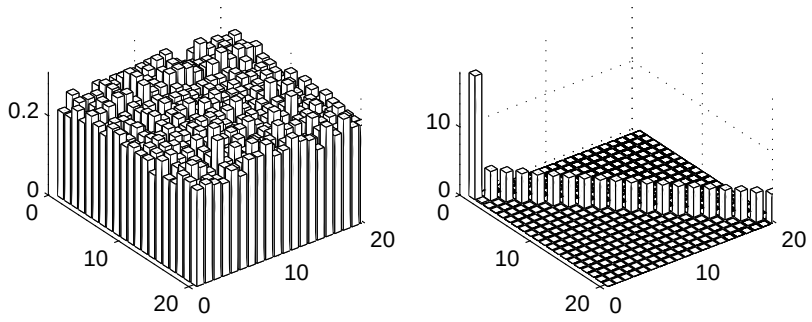

Figure 1: Relative block distribution (in per cent) of the square of Frobenius norm of an original matrix A (left) and after the QRF with CP + LQF (right). Random matrix A with $n = 600$, $\ell = 20$, $\kappa = 10$, and with a multiple minimal singular value.

dense matrix A before and after both pre-processing steps. The QRFCP and the LQF are clearly able to concentrate more that 90 percent of the Frobenius norm into diagonal blocks. Moreover, they 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 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, Salzburg, Austria. The first variant, denoted by [SVD], simply applies the PTBJA 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 the PTBJA to the R-factor. The computation ends by the post-processing according to Eq. (5). 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. (8) is required.

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.

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. (1) and (2)). 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, which covered the range $n = 2000$, 4000, 6000 and 8000.

Matrix elements in all cases were generated randomly, with a prescribed condition number κ and a known distribution of singular values $1 = \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n = 1/\kappa$. 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 spectrum 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 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 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.

4.1 Multiple minimal singular value

We begin with results for *well-conditioned matrices with a multiple minimal singular value*, which are summarized in Table 1. Its last two columns contain

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

ratios 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 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.

In contrast to n_{iter}, savings in T_p are of one order of magnitude less. The reason of this behavior lies in a high relative time complexity of the QRFCP. For all matrix orders, the QRFCP 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 decreasing 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 for other experiments.

Results for *ill-conditioned matrices with a multiple minimal singular value* are depicted in Table 2. When compared with well-conditioned matrices (see

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

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 singular values 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

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

seen, neither the [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

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

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 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.

5 Conclusions

The QRF with column pivoting, optionally followed by the LQF of R-factor, concentrates the Frobenius norm towards a diagonal (however, to a different

level for different spectra), and reveals the shape of the spectrum of A. 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 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. This is plainly seen in the case of well-conditioned matrices with geometrically distributed singular values, 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 matrix dimensions of order 10^5–10^6.

Acknowledgments

We thank the anonymous referees for their valuable comments and suggestions. This research was supported by the VEGA Grant no. 2/4136/24 from the Scientific Grant Agency of the Ministry of Education and Slovak Academy of Sciences, Slovakia.

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

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

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