UNIVERSITÄT SALZBURG

New Clique-Based Parallel Orderings for the Block-Jacobi EVD/SVD Algorithm

Gabriel Okša[a] Marián Vajteršic

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

Technical Report 2006-05 December 2006

Department of Computer Sciences

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

Technical Report Series

New Clique-Based Parallel Orderings for the Block–Jacobi EVD/SVD Algorithm

Gabriel Okša[*] and Marián Vajteršic[†]

Abstract. *We propose a new method for finding a parallel ordering needed in the parallel two-sided block-Jacobi EVD/SVD method. For a given matrix A, partitioned into block columns and block rows, such an ordering defines the subproblems that are solved in parallel in each parallel iteration step. Our approach is based on modeling the matrix block partition as a complete, edge-weighted graph, where the weight of edge (i, j) is defined as the sum of squares of Frobenius norms of the off-diagonal blocks A_{ij} and A_{ji}. The distinction between the physical and logical blocking factors enables to compose the SVD subproblems of varying size by using the contexts of processors under the Message Passing Interface paradigm. We show that finding the ordering that maximalizes the off-diagonal Frobenius norm of covered matrix blocks is equivalent to finding the partition of a complete graph into disjunct cliques of a given size where the total sum of all weights through all cliques is maximized. Since this task belongs to the class of NP-hard, we have designed and implemented a serial genetic algorithm for solving this problem approximately. We report first numerical results using* 12 *processors and well-conditioned matrices with a multiple minimal singular value of orders from* 1000 *to* 10000.

1 Introduction

The computation of a singular value decomposition (SVD) belongs to the most intensive computational tasks in the numerical linear algebra. Its application includes the signal and image processing, latent semantic indexing, analysis of biology-based arrays in the gene research, etc. Among many other approaches, one- and two-sided block Jacobi SVD algorithms for an arbitrary (complex or real) matrix are well-known since the last quarter of the 20th century. They are based on the decrease of the Frobenius norm of off-diagonal matrix blocks by means of one- or two-sided unitary (orthogonal) transforms which diagonalize the smaller 2-by-2 block subproblems.

The order in which the individual subproblems are solved is important for the convergence of the Jacobi algorithm. In principle, each off-diagonal block has to be nullified during the computational process. In this context, the ordering of subproblems is a prescribed, static

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

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

list according to which the off-diagonal matrix blocks are combined. During one *sweep* of the algorithm, each off-diagonal matrix block is nullified exactly once.

In the past, a large effort has been devoted to design many orderings and to investigate their convergence properties. However, for the parallel SVD computation on p processors, it is desirable to find such orderings that enable to solve p subproblems in parallel with an optimal data re-distribution between processors. This task was solved in papers [2, 3, 4] for distributed memory systems.

The main disadvantage of any serial or parallel block Jacobi algorithm with a static ordering is the fact that it does not take into account the actual status of a matrix under consideration. It may happen that some off-diagonal blocks with too small Frobenius norms are combined during one iteration step so that the decrease of the overall off-diagonal Frobenius norm is not optimal. On the other hand, one would like to decrease the off-diagonal norm as much as possible in each iteration step because this approach would lead to the (substantial) reduction of the number of iteration steps needed for the convergence.

Based on these considerations, we have designed, implemented and tested the parallel two-sided block-Jacobi algorithm with the so-called dynamic ordering of subproblems (see [5]). On a parallel computer with p processors, one parallel iteration step corresponds to p serial iteration steps. Then the task of decreasing the Frobenius norm of the off-diagonal blocks as much as possible can be formulated in terms of graph theory as the *maximum-weight perfect matching problem*. The nodes of the complete graph are numbered from 0 to$l-1$, where l is the blocking factor for the column-wise partition of matrix A, and the edge (i, j), $i < j$ has the weight equal to the sum of the Frobenius norms of matrix blocks A_{ij} and A_{ji}. In [5], the complexity of the whole parallel algorithm was analyzed in detail. Especially, it was shown that the complexity of the greedy approach for the solution of the maximum-weight perfect matching problem is of order $O(n^2/p+p^2 \log p)$ per one iteration step, whereas the complexity of 2-by-2 SVD problems including the matrix multiplications is of order $O(n^3/p^3 + n^3/p^2)$ per iteration step. Hence, the solution of the maximum-weight perfect matching problem does not represent any large overhead when compared with the number of arithmetic operations needed for the local SVD in each processor.

In this paper, we develop the concept of dynamic ordering further. Our new strategy for accelerating the convergence of the parallel Jacobi SVD algorithm is based on the distinction between the *physical* and *logical* blocking factor. This distinction enables to work with the static data distribution on processors and to create, at the beginning of each iteration step, an actual set of so-called *contexts* using the Message Passing Interface (MPI) paradigm of parallel processing. Each context contains a given number of processors which co-operate at the solution of one SVD sub-problem. The contexts are produced as a result of an approximate solution of another problem in the graph theory—namely, the maximum-weight decomposition of an edge-weighted, complete graph into a given number of cliques (i.e., complete sub-graphs). Since this problem in NP-hard, we have designed and implemented the genetic algorithm for its approximate solution.

The paper is organized as follows. In Section 2 we recall the basic features of dynamic ordering in the parallel block-Jacobi method. New clique-based ordering is described in detail in Section 3 and the new genetic serial algorithm in Section 4. The whole SVD algorithm was implemented on a cluster of personal computers using the MPI communication library. We report first results from our numerical experiments in Section 5. Finally, Section 6 contains conclusions and an outlook of future work.

Throughout the paper, $\|A\|_{\mathrm{F}}$ 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 [5].

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:         $\begin{pmatrix} A_{ri} & A_{rj} \\ A_{si} & A_{sj} \end{pmatrix} = X_{rs,t}^H \cdot \begin{pmatrix} A_{ri} & A_{rj} \\ A_{si} & A_{sj} \end{pmatrix}$
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 [6]). However, our experiments have shown that using $\ell = 2p$ ensures the least total parallel execution time in most cases.

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 $(W + W^T)_{ij} = \|A_{ij}\|_F^2 + \|A_{ji}\|_F^2$. Details concerning the dynamic ordering can be found in [5].

Figure 1 depicts an example of matching on a complete graph constructed for $p = 3$ processors, i.e., using the block factor $\ell = 2p = 6$. In this case the index pairs $(1, 6)$, $(2, 3)$ and $(4, 5)$

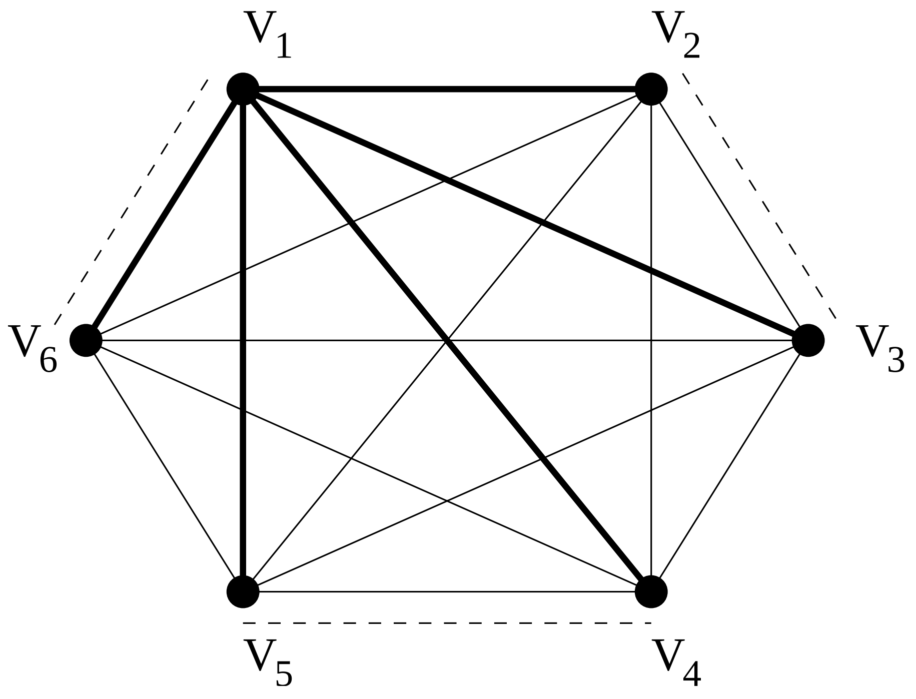


Figure 1: Maximum-weight perfect matching for $\ell = 6$ ($p = 3$).

were matched so that the first 2×2 SVD subproblem is given by matrix blocks 11, 66, 16, 61, the second one by 22, 33, 23, 32, and the last one by 44, 55, 45, 54. Based on the location of matrix blocks in processors during a previous iteration step, current blocks for each SVD subproblem must be grouped in some processor. This requires a point-to-point communication of various extent; in [6] an optimal communication algorithm was designed for minimalization the communication overhead.

After grouping matrix blocks in each processor, 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).

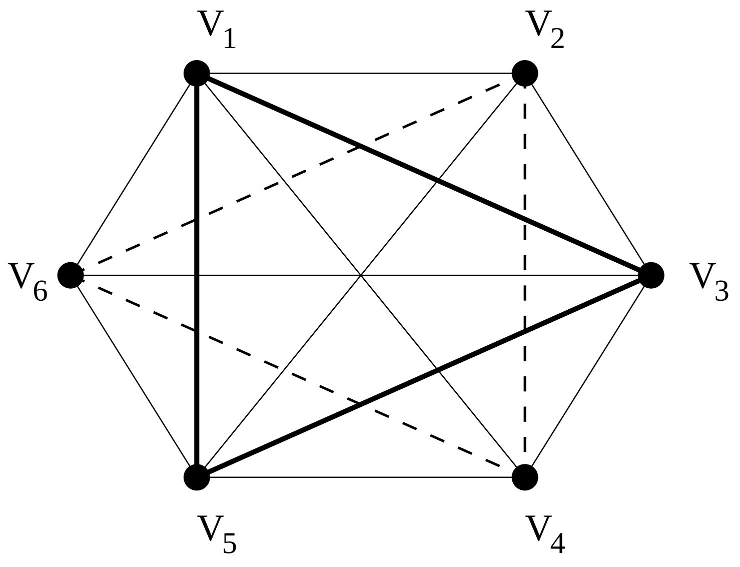
3 New clique-based block ordering

In contrast to the dynamic block ordering with a *fixed* blocking factor $\ell = 2p$, we now propose a more flexible scheme which allows to create larger SVD sub-problems to be solved in parallel.

Suppose a matrix A is divided into p block columns so that each processor owns exactly one block column. For a fixed number of processors this is a fixed *physical* blocking factor. In addition, the matrix can be considered divided into p block rows (at least virtually) so that we can work with blocks A_{ij} of matrix A. Numerical experiments in [5] have shown that, for a given matrix A, the number of parallel iteration steps needed for the convergence increases almost linearly on the blocking factor. Hence, to decrease the actual blocking factor we are dealing with in the computation (i.e., to increase the size of each SVD subproblem), one can build up a (variable) *logical* blocking factor ℓ over the physical one. In this case the matrix blocks are physically distributed over processors and the task is to construct larger logical blocks in each iteration step so as to maximize the decrease of the Frobenius norm of off-diagonal blocks.

This approach can be also formulated in the language of graph theory. Suppose that we work with a logical blocking factor ℓ which is connected to the physical blocking factor p via $\ell = p/r$ for some integer r. Consider a complete graph with p nodes where the edge (i, j), $i < j$, has the weight equal to the sum of the Frobenius norms of matrix blocks A_{ij} and A_{ji}. Notice that the blocks on the level of physical blocking factor p are considered for the construction of the complete graph. In one parallel iteration step, ℓ local SVDs are computed in parallel where each SVD subproblem is composed of $r(r-1)$ off-diagonal blocks of A (together with r corresponding diagonal blocks) that are given by the physical blocking factor p. To decrease the off-diagonal Frobenius norm of A maximally is equivalent to finding the partition of complete graph onto ℓ disjunct cliques of size r where the weight of cliques (i.e. the sum of weights through the edges belonging to cliques) is maximized. Notice that this task is a generalization of the maximum-weight perfect matching problem on a complete graph which we have already used above. However, a huge step in the computational complexity arises. While the maximum-weight perfect matching problem on a complete graph has an optimal polynomial algorithm for its solution, the maximum-weight perfect clique problem on a complete graph with the size of each clique larger than 2 is NP-hard.

To illustrate the creation of cliques from a complete graph, Figure 2 depicts the decomposition of a complete graph with $p = 6$ vertices (processors) into 2 cliques, each one of size $r = 3$. This decomposition defines two triples of indices, i.e., two SVD subproblems which are to be solved in parallel. The first SVD subproblem is given by a triple (1, 3, 5), so that processors 0, 2 and 4 are involved in its solution via one *context*. Note that processor i contains the whole i-th block column of matrix A. Therefore, the data within this first context is composed of matrix blocks 11, 31, 51 (delivered by processor 0), 13, 33, 53 (delivered by processor 2), and 15, 35, 55 (delivered by processor 4). Similarly, the second context is composed of processors 2, 4, 6 which deliver matrix blocks 22, 42, 62 (processor 1), 24, 44, 64 (processor 3) and 26, 46, 66 (processor 5). Hence, in general, the processors are divided in p/r disjunct contexts, where the number of contexts is equal to the number of cliques. Each context is responsible for solving one SVD subproblem. Note that this SVD subproblem is of block size $r \times r$ where r is the size of a clique. It is also clear that for $r = 2$ we get the maximum-weight perfect matching; however, now each processor contains only one block column so that even for $r = 2$ two processors have



CLIQUE 1: INDICES (1, 3, 5) (blocks and processors)

CLIQUE 2: INDICES (2, 4, 6) (blocks and processors)

Figure 2: Clique-based block ordering for $p = 6$, $r = 3$.

to collaborate in solving one SVD subproblem.

It turns out that an important parameter with respect to the convergence of the whole algorithm is the *portion* of off-diagonal blocks of matrix A covered by all contexts in one parallel iteration step. If all off-diagonal blocks were covered, the algorithm would converge in one (outer) iteration. Using the physical blocking factor p, there are together $p(p-1)$ off-diagonal blocks. In one iteration step, there are p/r cliques where each clique covers $r(r-1)$ off-diagonal blocks, so that there are together $p(r-1)$ off-diagonal blocks covered. Hence, the covered portion of off-diagonal blocks is given by

$$p_{\text{cov}} = \frac{r-1}{p-1},$$

which is the increasing function of r (when p is fixed). The minimum value of r is 2–each clique is of size 2 (matching) and there are $p/2$ contexts (cliques), but only $1/(p-1)$ off-diagonal blocks are nullified in each parallel iteration step. Conversely, the maximum value of r is $p/2$–we have only 2 contexts (cliques), but each clique has a maximal possible size–namely, $p/2$, and the covering of off-diagonal blocks in each parallel iteration step is also maximal: $p_{\text{cov}} = \frac{p-2}{2(p-1)}$. Thus one can expect more parallel iteration steps needed for the convergence in the former case than in the latter.

4 Genetic algorithm for the clique-based block ordering

To solve the NP-hard problem approximately, we have designed and implemented a serial genetic algorithm for finding the cliques, which is called in processor 0 at the beginning of each parallel iteration step in the two-sided parallel block-Jacobi method. This partition into cliques provides the ordering of off-diagonal matrix blocks, which is then broadcasted to all processors. Processors are then grouped into ℓ contexts, each context having r processors. One context is responsible for solving one SVD sub-problem of the block size $r \times r$.

Now we shortly describe the main constituents of our genetic algorithm. The first thing to be solved is the representation of a decomposition of complete graph with p vertices into $\ell = p/r$ disjunct cliques where each clique is of size r. This is done in the form of a two-dimensional (2D) binary string (matrix) of dimensions ℓ rows $\times$ p columns–such table is called a *genome*. Each row of a genome encodes one clique in the form of a 1D binary string of length p, whereby the value 1 at position i means that the vertex i is contained in a given clique. Note that each row of a genome must contain exactly r 1s and each column of a genome must contain exactly one 1. Such a genome is called *valid*, because it encodes (as a whole) one valid decomposition of a complete graph into disjunct cliques.

Next we must somehow evaluate the quality of a genome. In the framework of the theory of genetic algorithms it means to compute the *score* of a genome. In our case the quality of a genome is interpreted as the total weight of all cliques encoded in that genome. Recall that the complete graph is edge-weighted, whereby the weight of edge (i, j) is given by $\|A_{ij}\|_{\mathrm{F}}^2 + \|A_{ji}\|_{\mathrm{F}}^2$. Based on the binary encoding of each clique in a genome, we can compute the weight of each clique individually and then the score of a genome as the sum of weights of all cliques present in that genome. Recall that we want to find the maximum-weight decomposition of a complete graph into disjunct cliques. This gives us the criterion for judging the *quality* of a genome: the greater its score, the higher its quality.

A genetic algorithm works with a *population* of genomes and the population size is its one parameter. At the beginning of a solution process this population is created randomly, each genome is evaluated by its score, and then the *evolution* phase begins. Two current genomes (parents) meet in the *crossover* and produce two new genomes (a son and a daughter) with some probability p_{cr}. In our case, to maintain the quality of best genomes reached during the computation, a son is always a copy of the better parent, i.e., of the parent with higher score. A daughter is created from both parents by using the cycle through the individual genes (cliques) of both parents and copying the better gene (i.e., the clique with higher score) from a given pair of genes into a daughter. Of course, this copy may lead to a genome which is not valid since the cliques, chosen from two parents, may not be disjunct. Therefore each daughter is checked for validity and if not valid it is repaired. Finally, its score is evaluated.

To keep some level of random changes in genomes, we use the *mutation* of a genome. Any gene in a given genome can be changed into its opposite value randomly with a given probability p_{mt}. Of course, this change creates a genome which is not valid, so that it must be repaired. Note that the mutation changes randomly the score of a genome and it can also decrease it. Hence the mutation, in some sense, acts against the selection of best genomes in the crossover function and prevents the genetic algorithm from reaching the steady-state too soon. (The steady-state can be understood as a population of genomes with a small dispersion of their scores, so that one cannot expect a substantial increase of a score in future generations.)

Usually a genetic algorithm runs for a given number of generations of genomes. When finished, the best genome (i.e., the one with the highest score) is picked up. In our case, the best genome represents the decomposition of a complete graph into disjunct cliques with the highest achieved total weight. Of course, this solution is only an approximation of a maximum-weight decomposition of a complete graph into cliques. What we hope for is that our genetic algorithm will produce the solution which is not far from the optimum.

The serial genetic algorithm for the maximum-weight clique partition has been implemented in `C++` using the free available library `GALIB (v.2.4.6)` from MIT, USA. A typical run on processor 0 at the beginning of each parallel iteration step was defined by `population size` $= 20$, `number of generations` $= 10^4$, $p_{\mathrm{cr}} = 0.8$ and $p_{\mathrm{mt}} = 0.01$. The best genome (which represents the current partition of a complete graph into disjoint cliques) was broadcast to all remaining processors. Based on this information, ℓ actual contexts were created and, subsequently, ℓ SVD subproblems were solved in parallel.

5 Implementation and experimental results

The cluster of PCs consisted of 36 nodes arranged in a 6×6 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 $p = 12$, and the matrix A was of order form $n = 1000$ to $n = 10000$ 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, we present here the results for well-conditioned matrices with $\kappa = 10$ and with a multiple minimal SV with $\sigma_1 = 1$ and $\sigma_2 = \sigma_3 = \cdots = \sigma_n = \kappa^{-1}$. It is well known that the SVD of matrices with multiple or clustered SVs 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. Collective communication between processors was performed using the communication libraries BLACS (Basic Linear Algebra Communication Subroutines) and MPI.

Results of our numerical experiments are summarized in Table 1. The first column denotes the matrix order. The the results for two extreme cases are reported using 2 and 6 cliques, respectively, in each parallel iteration step. For each number of cliques the results are presented in three columns. The first column contains the number of parallel iteration steps needed for convergence, the second one contains the total parallel execution time T_p in seconds, and the third one the percentage of T_p spent in the serial genetic algorithm executed only in processor

Table 1: Performance for $p = 12$, $prec = 10^{-13}$, $\kappa = 10$, multiple minimal SV. T_p is in seconds, $R_{GA} = (T_{GA}/T_p) * 100$.

	2 cliques			6 cliques		
n	n_{iter}	T_p	R_{GA}	n_{iter}	T_p	R_{GA}
1000	37	98.8	4.3	409	866.1	7.5
2000	39	463.5	1.0	416	1544.8	4.2
3000	38	1393.3	0.3	406	3922.6	1.6
4000	36	3084.8	0.1	402	8054.4	0.8
5000	37	6144.0	0.1	403	15101.6	0.4
6000	37	9994.2	< 0.1	427	25951.0	0.2
7000	37	15926.5	< 0.1	412	39925.8	0.2
8000	35	23757.1	< 0.1	423	61874.0	0.1
9000	37	34040.0	< 0.1	417	85072.0	< 0.1
10000	38	56532.5	< 0.1	431	125658.6	< 0.1

0.

With respect to the number of parallel iteration steps, there is more than one order difference between 2 cliques and 6 cliques. In the previous section we have predicted the now observable trend (also for 3 and 4 cliques, not shown here) that the number of parallel iteration steps needed for the convergence will increase with increasing number of cliques due to the decreasing number of off-diagonal blocks covered in each parallel iteration step. However, there is more than tenfold increase of this number between 2 and 6. Clearly, the choice of six cliques is acceptable neither from the point of view of parallel iteration steps needed for the convergence, nor from the point of view of T_p. At the moment, we cannot answer the question of why is the deterioration of performance so large when using progressively larger number of cliques. We suspect that the quality of our genetic algorithm itself substantially deteriorates with an increasing number of cliques, but we need more experiments and analyses to document it.

The portion of T_p spent in the serial genetic algorithm decreases with an increasing matrix dimension and reaches less than 0.1 percent for 2 and 6 cliques by $n = 6000$ and $n = 9000$, respectively. This number is less than 8 per cent for matrix orders ≈ 1000. Therefore we can conclude that the genetic algorithm itself does not represent any significant computational burden and is successfully incorporated (from the point of view of timing) into the whole parallel software package.

6 Conclusions

We have presented a new strategy for accelerating the convergence of the two-sided block Jacobi SVD algorithm based on the decomposition of a complete, edge-weighted graph into a prescribed number of disjunct cliques where the total weight over all cliques is maximal. Since this problem is NP-hard, we have designed and implemented the genetic algorithm as a heuristics for solving the above problem approximately. First numerical results show quite a satisfactory performance of our new algorithm when using 2 cliques. However, the performance deteriorates quite rapidly for a larger number of cliques. More experiments are needed together with a better understanding of a genetic algorithm. We suspect that the genetic algorithm might be very sensitive to a variation of some of its parameters (e.g., population size, number of generations, probability of crossover) so that it will be not easy to improve its performance substantially.

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, S. Robert and M. Vajteršic, Experiments with parallel one-sided and two-sided algorithms for SVD, in: P. Zinterhof, M. Vajteršic and A. Uhl, eds., *Parallel Computation,* LNCS 1557 (1999) 48–57.

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

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

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

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

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

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

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

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

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

[12] 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.

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

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

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