Parallel Solution of Large Sparse Linear Systems by a Balance Scheme Preconditioner

Tz. Ostromsky[*] **A. Sameh**[†] **V. Sarin**[‡]

Computer Science Department, Purdue University,
West Lafayette, IN 47907 - 1398

Abstract

A parallel algorithm for preconditioning large and sparse linear systems is proposed. Both structural and numerical dropping are used to construct a preconditioner with proper structure. The Balance method is used to solve the linear system involving such preconditioner in each iteration.

The algorithm can be used together with any iterative method (GMRES is used for the experiments in this paper). As shown by numerical experiments, this approach is quite robust and has attractive parallel performance. It has been successful in solving quickly, and accurately, some ill-conditioned systems, which proved to be difficult for other preconditioned iterative methods.

Keywords: linear system, iterative method, preconditioner, sparse matrix, drop-tolerance, block partitioning, bandwidth, parallel computations.

1 Introduction

Iterative methods are most commonly used for solving large and sparse linear problems [9], [13], [16]. They are often faster and require less storage than direct solvers [4]. The main concern about their use, however, is their robustness as well as the accuracy achievable. Without proper preconditioning they often fail or stagnate and, even worse, could converge far from the actual solution (see the results for

[*] *e-mail:* tto@cs.purdue.edu
[†] *e-mail:* sameh@cs.purdue.edu
[‡] *e-mail:* sarin@cs.purdue.edu

GRE216B in Table 1 in the last section of this paper). That is why the preconditioning technique is at least as important as the iterative method, a lot of research has been conducted in this area [3, 6, 8, 14, 15, 16, 17]. More details regarding construction of our preconditioners is given in Section 2.

The Balance method (its original projection-based version is described in [10]) is used to perform the preconditioning. After block-row partitioning and factorization of the blocks, it eventually leads to the solution of a reduced system of smaller size.

Rather than factorizing the entire preconditioner, like in the well-known *Incomplete LU-factorization* (ILU) and similar preconditioning techniques, we factorize its blocks and the reduced system only, which are normally much smaller. In addition, there is a lot of natural parallelism in this task, which is highly desirable when using multiprocessors. To take full advantage of the features of the high-performance supercomputers, dense LAPACK [1] kernels are used in most of the floating-point computations. This results in a larger amount of arithmetics to be done, compared to classical sparse techniques ([4], [7], [17]), but results also in more efficiency on multiprocessors due to higher data locality. The approach generally gives good results for matrices, in which most of the nonzeros are packed in a band around the main diagonal (or matrices that can be reordered in such form). The bandwidth imposes certain restrictions on the number of blocks that can be used by the Balance scheme, which apparently limit the parallel performance of the method. In other words, the structure of the preconditioner is an important issue in our approach.

The Balance method, which is responsible for the preconditioning operations in our scheme, is described in more detail in Section 3. In Section 4 some results of numerical experiments on an SGI Origin-2000 are presented. The SPARSKIT version of GMRES ([13], [14]) is used as the underlying iterative algorithm. CG and BCGSTAB [16] have also been tested and give similar results, which are not presented in this paper. For comparison, GMRES has been run also with some ILU-type preconditioners, also available in SPARSKIT. The main conclusions of the experiments are summarized in Section 5.

2 Constructing the preconditioner

Generally speaking, a good preconditioner $\bar{A}$ of the given matrix A must satisfy the following conditions:

1. To be nonsingular ($rank(\bar{A}) = n$);
2. To be easy to invert / factorize;
3. The product $\bar{A}^{-1}A$ should be better conditioned than A.

It is very difficult to satisfy all of these requirements, especially for general sparse matrices. In our case, a mix of structural and numerical dropping strategies, combined with a proper reordering algorithm for bandwidth minimization, is used to obtain an appropriate preconditioner. The nonzero structure of the preconditioner has lower and upper staircase-like boundaries that contain the main diagonal (or, in general, a narrow band with lower and upper bandwidth k_1 and k_2 specified by the user). The structure is obtained in the following way:

---

1. For each $i = 1, \dots, n$
   (a) The i-th row is scanned to find the maximal absolute value of its element. Then the drop-tolerance for this row $\tau_i = \tau \max_j |a_{i,j}|$ is determined, where τ is the relative drop-tolerance, specified by the user.
   (b) Another scan is performed in order to extract all the elements, $\{a_{i,j} : |a_{i,j}| \geq \tau_i\}$. These form the i-th row of the *skeleton matrix* of A.

2. The skeleton matrix is reordered in order to minimize its bandwidth. The same permutation is applied to the original system (matrix A and right-hand side b).

3. For each $i = 1, \dots, n$
   (a) The i-th row of A is scanned to find the leftmost and the rightmost column index (l_i and r_i) of a nonzero, larger (by absolute value) than τ_i. If nonzero lower and upper bandwidth k_1 and k_2 are specified, then $l_i = i - k_1 \ , \quad r_i = i + k_2$ are taken as initial values of l_i and r_i in order to preserve all the nonzeros within the band.
   (b) The i-th row is scanned again and all the nonzeros outside the interval $[l_i, r_i]$ are dropped (i.e. considered to be zeros).

This procedure is consistent with the classical numerical dropping strategy (see [7], [17]), as all the nonzeros larger (by absolute value) than τ_i are kept. The elements in between the boundaries, however, are never dropped irrespective of their numerical value. As the matrix will be processed as dense blocks later, no saving would be realized even if the elements were dropped. Saving these elements, however, results in a more robust preconditioning compared to the incomplete LU-factorization (see Table 1 in Section (Numerical results).

The "pseudo-banded" structure of the preconditioners is illustrated in Fig.1. This structure is exploited by the Balance method, described in the next section.

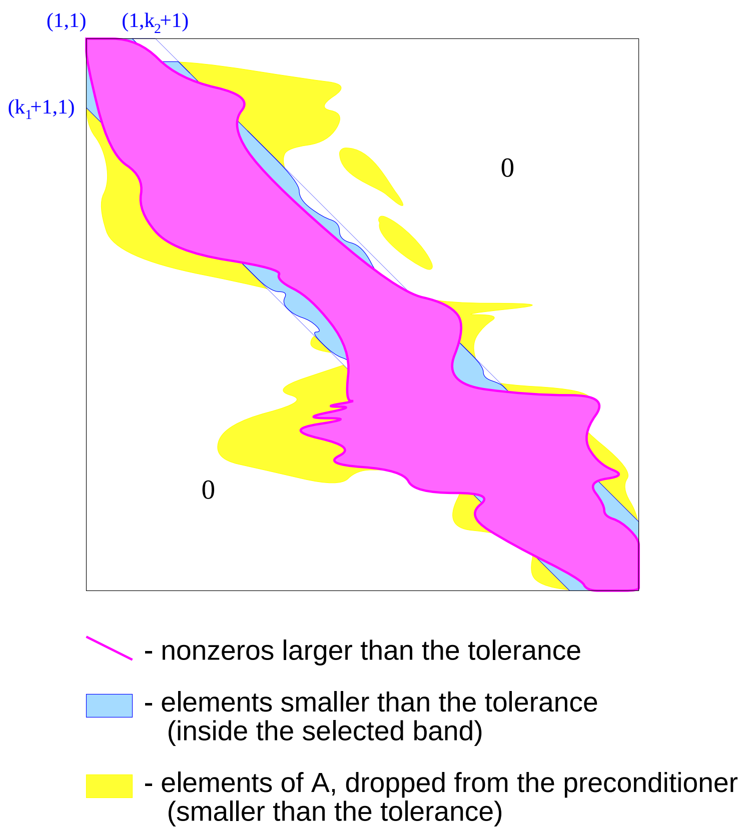
(1,1)
(1,k_2+1)
(k_1+1,1)
0
0
- nonzeros larger than the tolerance
- elements smaller than the tolerance (inside the selected band)
- elements of A, dropped from the preconditioner (smaller than the tolerance)

Figure 1: Sparsity structure of a “pseudo-banded” preconditioner

3 The Balance method

The Balance method, introduced in [10], is a powerful block-parallel algorithm for solving sparse linear systems. Its main idea is to reduce the original problem into several subproblems of smaller size. After the partitioning, it computes the QR-factorizations of non-overlapping block-rows (these are independent tasks) followed by the solution of a system that corresponds to the unknowns common for neighboring blocks (called the *reduced system*). For its block-partitioning, the Balance method uses the specific structure of the matrix (banded, block-banded or "pseudo-banded", as described in the previous section). For the sake of efficiency the band should be as narrow as possible.

The band of such a matrix, split into several block-rows, has block structure as shown in (1). To a certain extent one has freedom to vary the number and the size of the block-rows, which themselves strictly determine the entire block structure. The smaller the bandwidth, the larger the number of blocks this matrix allows to be partitioned on, and the smaller the size of the reduced system, provided partitioning on the same number of blocks is used.

The corresponding block-matrix expression of the original problem (solving the sparse system $Ax = b$) is given in (1). The overlapping parts (set of columns) of the neighboring block-rows should be separated by non-overlapping parts (blocks A_i), as shown below:

$$(1) \qquad \begin{pmatrix} A_1 & B_1 & & & & & & \\ & C_2 & A_2 & B_2 & & & & \\ & & & C_3 & A_3 & B_3 & & \\ & & & & & \cdots & \cdots & \\ & & & & & & C_p & A_p \end{pmatrix} \begin{pmatrix} x_1 \\ \xi_1 \\ x_2 \\ \vdots \\ \xi_{p-1} \\ x_p \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_p \end{pmatrix}$$

where x_i and ξ_i are the unknowns, corresponding to A_i and B_i (or C_{i+1}) respectively.

Splitting the above system by block-rows (and duplicating temporarily the unknowns in ξ_i, common for both B_i and C_{i+1}), we obtain:

$$(2) \qquad (A_1, B_1) \begin{pmatrix} x_1 \\ \xi_1 \end{pmatrix} = b_1$$

$$(3) \qquad (C_i, A_i, B_i) \begin{pmatrix} \tilde{\xi}_{i-1} \\ x_i \\ \xi_i \end{pmatrix} = b_i \quad i = 2 ... p-1$$

$$(4) \qquad (C_p, A_p) \begin{pmatrix} \tilde{\xi}_{p-1} \\ x_p \end{pmatrix} = b_p$$

Denote by E_i the non-trivial part of the i-th block-row.

$$E_1 = (A_1, B_1) \tag{5}$$
$$E_i = (C_i, A_i, B_i) \qquad i = 2...p-1 \tag{6}$$
$$E_p = (C_p, A_p) \tag{7}$$

The solution of each of the underdetermined subsystems (2) – (4) can be obtained via QR-factorization, as follows:

$$E_i z_i = b_i \quad (i = 1, \ldots, p) \tag{8}$$
$$E_i^T = (Q_i, \tilde{Q}_i) \begin{pmatrix} R_i \\ 0 \end{pmatrix} \tag{9}$$
$$\bar{z}_i = Q_i R_i^{-T} b_i \tag{10}$$
$$z_i = \bar{z}_i + \tilde{Q}_i y_i \tag{11}$$

where $\bar{z}_i$ is a particular solution, $\tilde{Q}_i$ is a basis for the null space $\mathcal{N}(E_i)$, z_i is the general solution, and y_i is an arbitrary vector.

Back into the systems (2)-(4) we obtain correspondingly:

$$\begin{pmatrix} x_1 \\ \xi_1 \end{pmatrix} = \begin{pmatrix} \bar{z}_{1,1} \\ \bar{z}_{1,2} \end{pmatrix} + \begin{pmatrix} \tilde{Q}_{1,1} \\ \tilde{Q}_{1,2} \end{pmatrix} y_1$$
$$\begin{pmatrix} \tilde{\xi}_{i-1} \\ x_i \\ \xi_i \end{pmatrix} = \begin{pmatrix} \bar{z}_{i,1} \\ \bar{z}_{i,2} \\ \bar{z}_{i,3} \end{pmatrix} + \begin{pmatrix} \tilde{Q}_{i,1} \\ \tilde{Q}_{i,2} \\ \tilde{Q}_{i,3} \end{pmatrix} y_i \quad i = 2...p-1$$
$$\begin{pmatrix} \tilde{\xi}_{p-1} \\ x_p \end{pmatrix} = \begin{pmatrix} \bar{z}_{p,1} \\ \bar{z}_{p,2} \end{pmatrix} + \begin{pmatrix} \tilde{Q}_{p,1} \\ \tilde{Q}_{p,2} \end{pmatrix} y_p$$

Imposing the natural requirement $\xi_i = \tilde{\xi}_i$, $i = 1, ...p-1$, we obtain the *reduced system* $My = g$, equivalent to the original system $Ax = b$. The matrix M of the reduced system has a block-bidiagonal form with rectangular blocks (12), obtained from the QR-factorization of E_i^T. $\tilde{Q}_i$ is actually a null space basis of the linear transform corresponding to E_i^T. A storage-saving alternative to the QR-factorization, the projection-based approach (based on [12]), has been proposed in [10].

$$M = \begin{pmatrix} \tilde{Q}_{1,2} & -\tilde{Q}_{2,1} & & & \\ & \tilde{Q}_{2,3} & -\tilde{Q}_{3,1} & & \\ & & \ddots & \ddots & \\ & & & \tilde{Q}_{p-1,3} & -\tilde{Q}_{p,1} \end{pmatrix} \tag{12}$$

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}, \quad g = \begin{pmatrix} \bar{z}_{2,1} - \bar{z}_{1,2} \\ \bar{z}_{3,1} - \bar{z}_{2,3} \\ \vdots \\ \bar{z}_{p,1} - \bar{z}_{p-1,3} \end{pmatrix}$$

To find the unique solution of (1), one has to find the solution of the reduced system $My = g$, and to combine it with the particular solutions of the underdetermined subsystems (2)-(4). The size of M is usually much smaller than the size of A.

Most modern supercomputers have extremely efficient low-level implementation of the dense basic linear algebra subroutines (BLAS), especially for matrix-vector (BLAS 2) and matrix-matrix (BLAS 3) operations. Hardware developments also favor dense matrix computations, as the speed of arithmetic operations is much faster than the memory operations, especially as cache and register size increase. As a result, the dense matrix factorization becomes more and more efficient and achieves computational speed that almost reach the peak performance of the machine. In many cases applying dense matrix techniques to sparse systems results in faster solution, provided the matrix is not too large or extremely sparse. Some successful hybrid techniques for sparse matrices have recently been proposed in [11], treating parts of the sparse matrix as dense blocks.

Our algorithm exploits not only the high-performance of the dense computational technique, but also its inherent parallelism. On the bottom level of our code calls to the parallel version of appropriate LAPACK [1] routines are used. Thus two nested levels of parallelism are created, making the algorithm perfectly suitable for cluster-based parallel machines.

4 Numerical results

All the experiments presented in this section are carried out on the NCSA[1] SGI-CRAY Origin 2000 cluster parallel machine, based on R10000 type processors.

The experiments illustrate the accuracy of the solution, the performance and the parallelism of the proposed algorithm, which includes our Balance method preconditioner (denoted shortly as BM) and an iterative solver. The SPARSKIT version of GMRES is used in the current implementation. GMRES is also used with three other preconditioners, available in SPARSKIT, and the results are presented for comparison. These preconditioners are:

- ILUT – Incomplete LU factorization with dual truncation strategy [15];
- ILUTP – ILUT with column pivoting;
- ILUD – ILU with standard threshold dropping and diagonal compensation.

[1]National Computational Science Alliance, University of Illinois at Urbana-Champaign.

4.1 Parameters of GMRES and the SPARSKIT preconditioners

For some user-specified input parameters of GMRES the following common values are used in all the experiments, presented in the paper. The maximal number of iterations for GMRES has been set to 400. Another stopping criterion is the residual-based convergence test, controlled by two floating-point parameters α and β. The iterations are terminated (and the solution considered successful) if the norm of the last residual r_s satisfies the following condition:

$$\|r_s\| \leq \beta \, \|b\| + \alpha \tag{13}$$

Choosing smaller values of β and α results in higher accuracy of the solution, larger ones – in fewer iterations and better time. In our experiments $\beta = 10^{-7}$ and $\alpha = 10^{-16}$ are used. The number of Krylov vectors is another important parameter for GMRES as well as for the other CG-like iterative methods (see for more details [9], [13], [16]). Its value has been set to 35.

The ILU versions with dual truncation strategy, ILUT and ILUTP, depend on two parameters: the drop-tolerance τ and the maximal number of fill-ins per row f. Large f ($f \geq n$) results in no restriction on the number of fill-ins. Such a large value is used in the experiments throughout the paper.

The diagonal compensation in ILUD is controlled by the parameter α. The value $\alpha = 1$ is used, that means full diagonal compensation.

A large initial value of the drop-tolerance τ (2^{-1} in our experiments) is passed initially to all the preconditioners. In case of failure of the preconditioner (due to singularity) or in the iterative method, the current tolerance is decreased twice and the preconditioner is restarted with the new value. This repeats until either a successful solution is obtained or until the current tolerance becomes smaller than the lower limit (10^{-10} in our experiments). In the latter case, the solver is declared to be "not convergent", see Table 1. Results for the first successful τ (if any) are given in the tables.

4.2 Test matrices

Several nonsymmetric matrices from the well-known Harwell-Boeing collection [5] (type RUA) are used in the experiments. These include the Grenoble matrices (abbreviated as GRE*size*, where *size* is the size of the corresponding matrix).

These are unstructured sparse matrices. In order to convert them in some banded form (which will benefit any of the preconditioners used in the experiments), the reverse Cluthill-McKee reordering has been applied prior to the experiments. In our approach, further reduction of the bandwidth is achieved by the preconditioning technique, described in Section 2.

4.3 Constructing the test problems

It is convenient to have test problems with a known solution. Therefore the following procedure has been used in all experiments. The right-hand side of the problem $b = A\bar{x}$ is calculated by using the chosen matrix and an exact solution, which either has all components equal to one or the components of which are calculated by a random number generator. This procedure allows us (i) to calculate the norm of the exact error vector of the computed solution and (ii) to compare the norm of the exact error vector with the norm of the evaluated (by the code) error vector.

4.4 Accuracy of the solution and number of iterations

Matrix name, nonzeros, cond. number	Accuracy of the solution and the number of iterations for the largest successful tolerance $\tau = 2^{-k}$			
	ILUT	ILUTP	ILUD	BM
GRE216A nz=876 $\kappa \approx 10^2$	1.94E-5 (30 it) $\tau = 2^{-3}$	1.94E-5 (30 it) $\tau = 2^{-3}$	2.69E-5 (164 it) $\tau = 2^{-3}$	4.42E-14 (6 it) $\tau = 2^{-1}$
GRE216B nz=876 $\kappa \approx 10^{14}$	Fails	Fails	Fails	1.01E-3 (2 it) $\tau = 2^{-6}$
GRE343 nz=1435 $\kappa \approx 10^2$	1.71E-5 (21 it) $\tau = 2^{-4}$	1.71E-5 (21 it) $\tau = 2^{-4}$	9.04E-6 (26 it) $\tau = 2^{-4}$	5.50E-14 (6 it) $\tau = 2^{-1}$
GRE512 nz=2192 $\kappa \approx 10^2$	1.10E-5 (19 it) $\tau = 2^{-4}$	1.10E-5 (19 it) $\tau = 2^{-4}$	1.78E-5 (66 it) $\tau = 2^{-5}$	9.34E-14 (6 it) $\tau = 2^{-1}$
GRE1107 nz=5664 $\kappa \approx 10^8$	Not conv.	Not conv.	Not conv.	5.51E-9 (3 it) $\tau = 2^{-4}$

Table 1: The accuracy of the solution and the number of iterations for the largest successful value of the tolerance τ=2^{-k} , obtained in the solution of several test problems by the preconditioned GMRES with the Balance method (BM) and three ILU-type preconditioners from SPARSKIT.

In Table 1 the number of iterations and the accuracy achieved by the end of the iterative process are presented for the GMRES algorithm, preconditioned by the

Time (GMRES + Preconditioner) on 4 processors				
Matrix	Preconditioner to GMRES			
	ILUT	ILUTP	ILUD	BM
GRE216A	0.021	0.030	0.069	0.018
GRE216B	Fails	Fails	Fails	0.018
GRE343	0.034	0.066	0.032	0.036
GRE512	0.063	0.131	0.126	0.056
GRE1107	Not conv.	Not conv.	Not conv.	0.734

Table 2: Timing resilts for the solution of the test problems in Table 1 on 4 processors, with each of the matrices partitioned into 4 blocks. The corresponding drop-tolerance τ is the same, as in Table 1.

Balance method as well as by the ILU-type preconditioners from SPARSKIT. Each algorithm is applied to the set of Grenoble matrices from the Harwell-Boeing sparse matrix collection. Some of these matrices are rather ill-conditioned (the condition number of each matrix together with its name and number of nonzeros is given in the leftmost column of Table 1). The results show clearly the robustness of our preconditioning algorithm. In general, BM achieves much higher accuracy in just a few iterations.

Note that the word "Fails" in Table 1 (for the matrix GRE216B) means that the iterations converge, but the solution obtained is not the correct one (the norm of the actual error is very large). Knowing the exact solution allows us to detect this error, but in practice it will not be detected. For GRE1107 none of the ILU-based solvers is convergent, even with τ as small as 10^{-16}, while BM succeeds in just 3 iteration.

The CPU time for these test problems are given in Table 2. Although our preconditioners are denser (and thus, more expensive) than these obtained by ILU, the savings resulting from the reduced number of iterations and parallelism utilization are crucial for achieving the overall favorable performance.

4.5 Time, performance, and relative cost of the main stages

The Grenoble matrices are not large enough for time and performance analysis, and especially to analyze the relative cost of the main stagtes of our algorithm. In addition, the Grenoble matrices do not allow partitioning into more than 4 - 6 blocks, while the number of blocks is crucial in using efficiently more processors in parallel. That is why a larger matrix from the Harwell-Boeing set of unsymmetric square matrices, LNSP3937, has been chosen for the experiments in this subsection. It has 3937 rows and columns, 25407 nonzero elements, and allows partitioning into more than 16 blocks.

There are several user-controlled parameters, that eventually affect the speed and the performance of BM algorithm. The most important are the tolerance τ and the number of blocks. These parameters sometimes have opposite effect on the time for the different stages of the algorithm. Tables 3 and 4 contain more detailed timing results (by stages), which helps to understand better the behaviour and the relations between the main stages, as well as the overall performance of the algorithm. The stages are listed below with their characteristics and parallelization abilities:

1. Construction of preconditioner, including its block-partitioning (inherently sequential);
2. QR-factorization of the blocks (coarse-grain parallel);
3. LU-factorization of the reduced system (fine-grain parallel);
4. Iterative stage (partially parallel).

Matrix LNSP3937 [n=3937, nz=25407, $\kappa \approx 10^6$] (partitioned on 8 block-rows)				
Stage	Time	**Time (*Speeding factor*)**		
	1 proc.	2 proc.	4 proc.	8 proc.
Construction of preconditioner	0.22	0.22 (*1.0*)	0.22 (*1.0*)	0.22 (*1.0*)
QR-factorization of the blocks	7.74	3.92 (*1.9*)	2.19 (*3.5*)	1.46 (*5.3*)
Reduced system factorization	0.58	0.49 (*1.2*)	0.39 (*1.5*)	0.38 (*1.5*)
Iterations	0.40	0.29 (*1.3*)	0.21 (*1.9*)	0.18 (*2.2*)
TOTAL	8.94	4.92 (*1.8*)	3.01 (*3.0*)	2.23 (*4.0*)

Table 3: Times and speeding factors of the main stages of BM algorithm for an 8 blocks partitioning of the test problem

The results for the 4 stages of the BM algorithm , using partitioning into 8 and 16 block-rows are given in Tables 3 and 4 respectively. The block QR-factorization stage, performed entirely by the Balance method, accelerates quite well, especially

Matrix LNSP3937 [n=3937, nz=25407, $\kappa \approx 10^6$] (partitioned on 16 block-rows)					
Stage	Time	**Time (*Speeding factor*)**			
	1 pr.	2 proc.	4 proc.	8 proc.	16 proc.
Construction of preconditioner	0.22	0.22 (*1.0*)	0.22 (*1.0*)	0.22 (*1.0*)	0.22 (*1.0*)
QR-fact. of the blocks	4.35	2.18 (*2.0*)	1.16 (*3.8*)	0.61 (*7.1*)	0.44 (*9.9*)
Reduced system factorization	1.44	1.16 (*1.3*)	0.96 (*1.5*)	0.87 (*1.7*)	0.86 (*1.7*)
Iterations (4)	0.72	0.38 (*1.9*)	0.32 (*2.3*)	0.27 (*2.7*)	0.28 (*2.7*)
TOTAL	6.73	3.94 (*1.7*)	2.65 (*2.8*)	1.97 (*3.4*)	1.80 (*3.7*)

Table 4: Times and speeding factors of the main stages of BM algorithm for a 16-blocks partitioning of the test problem

for the partitioning on larger number of blocks. As this is the most expensive part of the work, the total time is also smaller for 16 blocks. With increasing the number of blocks, however, the size of the reduced system also increases, which leads to spending larger part of the time for its factorization. As this stage is not so efficient, this finally results in slight decrease of the total speeding factor.

The iterative stage is less efficient, especially on larger number of processors. This is not a surprise, as part of the work in the rallel matrix-vector products are followed by inner products and other inherently sequential operations.

The total times (on 1 and 8 processors), the size of the reduced system and the number of iterations in dependence with the tolerance value are given in Table 5. It contains results of the solution with 3 different values of τ (10^{-2}, 10^{-4}, 10^{-6}) for partitioning into 8 blocks (the effect of the number of blocks is considered later). The larger the tolerance, the more elements have been dropped, which implies smaller size of the reduced system, and correspondingly, smaller time for its factorization. On the other hand, larger τ means less exact preconditioner and more iterations, so more time will be spent on the iterative stage. When a good balance is achieved between these two stages, a local minimum in the total time can be expected, as observed in Table 5.

The performance achieved (evaluated by using perfex software tool), is given in the last column of Table 6. It is rather high for a sparse system solver, which is due to the efficient dense block techniques, used as building blocks in the implementation

Matrix LNSP3937 [n=3937, nz=25407, $\kappa \approx 10^6$] (partitioned on 8 block-rows)				
Drop-tolerance	Total time		Size of M	Iterations
	1 proc.	8 proc.		
$\tau = 10^{-2}$	10.86	2.67 (*4.1*)	960	30 it.
$\tau = 10^{-4}$	8.94	2.23 (*4.0*)	1056	4 it.
$\tau = 10^{-6}$	9.67	2.49 (*3.8*)	1071	3 it.

Table 5: Times (on 1 and 8 processors), the size of the reduced system and the number of iterations for solving a system with the Harwell-Boeing matrix LNSP3937, partitioned into 8 blocks with 3 different values of τ.

Matrix LNSP3937, partitioned on 8 block-rows		
Number of proc.	Total time and speeding factor	MFLOPS (total)
1	10.86	98
2	7.21 (*1.5*)	145
4	3.92 (*2.7*)	269
8	2.67 (*4.1*)	403

Table 6: Times, speeding factors and performance of the algorithm BM. The performance is evaluated by using the perfex tool on the SGI Origin-2000.

of the Balance method.

4.6 Comparison with some direct solvers

In Table 4.6 two direct solvers from LAPACK are compared to the preconditioned GMRES with the previously discussed preconditioners. The dense LAPACK solver, in spite of its best speeding factor, is not competitive for so sparse matrix like SHERMAN3. The banded LAPACK solver is much cheaper, but its speeding factor tends to stagnate. The ILU-type preconditioners to GMRES are cheap, but not parallel. On 1 processor BM is as cheap as them. Its moderate speeding factor makes it the fastest among the tested algorithms on multiple processors.

Matrix SHERMAN3 [n=5005, nz=20033, $\kappa \approx 10^4$]				
Method	Time-	**Time** (***Speeding factor***)		
	1 proc.	4 proc.	8 proc.	16 proc.
dense LAPACK	292.7	87.11 (*3.3*)	53.08 (*5.5*)	36.10 (*8.1*)
band LAPACK	11.87	7.15 (*1.6*)	6.95 (*1.7*)	6.95 (*1.7*)
BM	5.34	1.98 (*2.7*)	1.49 (*3.6*)	1.41 (*3.8*)
ILUT	5.41	5.33 (*1.0*)	5.36 (*1.0*)	5.39 (*1.0*)
ILUTP	7.66	7.60 (*1.0*)	7.43 (*1.1*)	7.71 (*1.0*)
ILUD	4.63	4.56 (*1.0*)	4.61 (*1.0*)	4.69 (*1.0*)

Table 7: Two direct algorithms, the dense and the band LAPACK solver, compared to the preconditioned GMRES with the previously discussed preconditioners. Partitioning into 8 blocks is used in BM.

5 Conclusions

The following main conclusions can be drawn from the results of our numerical experiments, described in the previous section:

- BM is an efficient preconditioning algorithm, that achieves higher accuracy in less iterations, compared to the ILU-based preconditioners.

- BM is inherently parallel. On multiple processors it is much faster than the ILU preconditioners from SPARSKIT.

- The high performance achieved (not typical for a sparse solver) is due to the efficient dense block modules from LAPACK, used in the implementation of the Balance scheme.

- As a hybrid solver, BM successfully combines the speed of the iterative sparse solvers with the reliability and the speeding factor of the direct solvers.

More experiments with larger matrices of various pattern are needed to confirm the advantages of the proposed method and to find its ultimate applications.

Acknowledgements

This research was partially supported by NSF grants CCR-9619763 and ECS-9527123 (Grand Challenge), as well as by grant MU-MM-02/96 from the Bulgarian Ministery of Education, Science and Technology.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, *LAPACK: Users' guide*. SIAM, Philadelphia, 1992.

[2] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994.

[3] E. F. D'Azevedo, F. A. Forsyth, and W. P. Tang, *Towards a cost effective ILU preconditioner with high-level fill*, BIT, Vol.31, 1992, pp.442–463.

[4] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford-London, 1986.

[5] I. S. Duff, R. G. Grimes and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, Vol. 15, 1989, pp. 1–14.

[6] I. S. Duff, G. A. Meurant, *The effect of reordering on preconditioned conjugate gradients*, BIT, Vol.29, 1989, pp. 635–657.

[7] K. A. Gallivan, P. C. Hansen, Tz. Ostromsky and Z. Zlatev, *A locally optimal reordering algorithm and its application to a parallel sparse linear system solver*, Computing, Vol.54, No.1, 1995, pp. 39–67.

[8] K. A. Gallivan, A. H. Sameh and Z. Zlatev, *A parallel hybrid sparse linear system solver*, Comput. Systems Engineering, No.1, 1990, pp. 183–195.

[9] K. A. Gallivan, A. H. Sameh and Z. Zlatev, *Solving general sparse linear systems using conjugate gradient-type methods.* In Proceedings of the 1990 International Conference on Supercomputing, June 11–15 1990, Amsterdam, The Netherlands. ACM Press, 1990.

[10] G. H. Golub, A. H. Sameh and V. Sarin, *A parallel balanced method for sparse linear systems*, Preconditioned Iterative Solution Methods for Large Scale Problems in Scientific Computations (PRISM '97), May 1997.

[11] P. C. Hansen, Tz. Ostromsky, A. Sameh, Z. Zlatev, *Solving sparse linear least-squares problems on some supercomputers by using a sequence of large dense blocks*, BIT, Vol.37, No.3, 1997, pp.535–558.

[12] C. Kamath and A. Sameh *A projection method for solving nonsymmetric linear systems on multiprocessors*, Parallel Computing, No.9, 1989, pp.291–312.

[13] Y. Saad, *Krylov subspace methods on supercomputers*, SIAM J. Scient. Stat. Comp., No.10, 1989, pp. 1200–1232.

[14] Y. Saad, *A flexible iner-outer preconditioned GMRES algorithm*, SIAM J. Scient. Stat. Comp., Vol.14, No.2, 1993, pp. 461–469.

[15] Y. Saad, *ILUT: a dual threshold incomplete ILU preconditioner*, Numer. Linear Algebra Appl., Vol.1, No.4, 1994, pp. 387–402.

[16] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Co., Boston, 1996.

[17] Z. Zlatev, *Computational Methods for General Sparse Matrices*, Kluwer Academic Publishers, Dordrecht-Toronto-London, 1991.