Mathematica Balkanica

Mathematical Society of South-Eastern Europe
A quarterly published by
the Bulgarian Academy of Sciences – National Committee for Mathematics

Mathematica
Balkanica

New Series Vol. 17, 2003, Fasc. 3-4

Performance Analysis of a Parallel MIC(0) Preconditioning of Rotated Bilinear Nonconforming FEM Systems

Gergana Bencheva, Svetozar Margenov

Presented by Bl. Sendov

A new parallel preconditioner for solution of large FEM linear systems is theoretically and experimentally studied. The considered second order elliptic problem is discretized by rotated bilinear nonconforming finite elements on quadrilaterals. The proposed algorithm is based on the modified incomplete Cholesky factorization MIC(0) applied to a locally constructed approximation of the original stiffness matrix. The obtained preconditioner has a special well parallelizable block structure preserving the robustness of the pointwise incomplete factorization. Theoretical estimates of the parallel times are derived. They are compared with the measured performance data on two Beowulf type Linux clusters. The related speed-up and efficiency are analyzed. A special attention is focused on the real distribution of computation and communication times.

AMS Subj. Classification: 65F10, 65N30, 65Y05

Key Words: parallel algorithms, sparse matrix computations, preconditioning methods, nonconforming FEM

1. Introduction

The nonconforming finite elements and the parallel algorithms are two advanced computational mathematics topics which have provoked a lot of publications during the last decades. How to incorporate the recent achievements in both areas to obtain efficient numerical solution methods of the nowadays real life mathematical models? This is the leading question, motivating the present study. A lot of important engineering and environmental problems, e.g. petroleum recovery, ground-water contamination, seismic exploration etc., are modeled by boundary value problems in the presence of strong heterogeneities, anisotropy and large coefficient jumps. The mixed finite element method (FEM) is known to be the best generally applicable discretization approach for such problems. However, this technique leads to enlarged system of algebraic equations which is additionally more expensive to solve, because the problem is saddle-point. Arnold and Brezzi (see [1]) have shown that the lowest order Raviart-Thomas mixed FEM approximation is equivalent to the usual Crouzeix-Raviart nonconforming FEM approximation. Further such relationship has been established and studied for a large variety of mixed finite element spaces and the related nonconforming spaces.

This work is focused on the implementation of rotated bilinear quadrilateral elements, first proposed in [6] as a cheapest stable FEM approximation of the Stokes problem. Here, the goal is to analyze theoretically and experimentally the properties of the recently introduced (see [8]) parallel preconditioned conjugate gradient (PCG) solver for related elliptic FEM system. Our study can be viewed as a generalization of the results published by Gustafsson and Lindskog in [4, 5], where a parallel MIC(0) factorization is constructed for the conforming linear FEM system, corresponding to a triangulation on a skewed mesh. A two-level parallel algorithm of this class, but for Crouzeix-Raviart nonconforming FEM systems, is developed and theoretically studied in [9].

The outline of the paper is as follows. In Section we give the needed background of the bilinear nonconforming FEM discretization. We briefly introduce the algorithms under consideration, where mid-point (MP) and integral mid-value (MV) interpolation operators are used in the computation of the nodal basis functions. In Section , the element-by-element construction of the preconditioner is presented. How the resulting PCG algorithm is implemented in parallel is described in Section . Some advantages and disadvantages of the solver, based on the derived estimates of the parallel times, are observed at the end of this section. Last part of the paper is devoted to the analysis of the numerical results obtained on two Beowulf type Linux clusters. Their architectures give an idea how the drawbacks influence the speed-up, but also show the potential of the developed parallel solver for implementations on shared memory machines.

2. Nonconforming finite element discretization

We consider the isotropic two-dimensional elliptic problem associated with the bilinear form

$$a_h(u_h, v_h) = \sum_{e\in\omega_h} \int_e a(e) \sum_{i=1}^{2} \frac{\partial u_h}{\partial x_i}\frac{\partial v_h}{\partial x_i} dx\,. \tag{1}$$

Here ω_h is a decomposition of the computational domain into convex quadrilaterals denoted by e. The coefficient $a(e)$ is a constant on each $e \in \omega_h$ defined as an averaged value of the coefficients of the original boundary value problem.

The finite element space V_h corresponding to ω_h is obtained using the rotated bilinear nonconforming finite elements, as proposed in [6]. The reference element E is the unit square with sides Γ_j, $j = 1, \ldots, 4$, parallel to coordinate axes, and nodes $j = 1, \ldots, 4$ which are the mid-points of the sides. Mid-point and integral mid-value interpolation operators are implemented in construction of the nodal basis functions $\varphi_i \in S_p$, $S_p = \text{span}\{1, \mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_1^2 - \mathrm{x}_2^2\}$. This leads to two alternative constructions of V_h, referred as Algorithm MP and Algorithm MV respectively. The standard interpolation conditions $\varphi_i(j) = \delta_{ij}$, $i, j = 1, \ldots, 4$ are used in Algorithm MP, where δ_{ij} is the Kronecker symbol. The conditions for Algorithm MV are

$$\frac{1}{|\Gamma_j|}\int_{\Gamma_j} \varphi_i dx = \delta_{ij},\ i,j = 1,\ldots,4\ ,$$

where Γ_j is the side of E containing the node j. The expressions of φ_i for these two cases are as follows:

Algorithm MP	Algorithm MV
$\varphi_1(x_1,x_2)=\frac{1}{4}(1-2x_1+(x_1^2-x_2^2))$	$\varphi_1(x_1,x_2)=\frac{1}{8}(2-4x_1+3(x_1^2-x_2^2))$
$\varphi_2(x_1,x_2)=\frac{1}{4}(1+2x_1+(x_1^2-x_2^2))$	$\varphi_2(x_1,x_2)=\frac{1}{8}(2+4x_1+3(x_1^2-x_2^2))$
$\varphi_3(x_1,x_2)=\frac{1}{4}(1-2x_2-(x_1^2-x_2^2))$	$\varphi_3(x_1,x_2)=\frac{1}{8}(2-4x_2-3(x_1^2-x_2^2))$
$\varphi_4(x_1,x_2)=\frac{1}{4}(1+2x_2-(x_1^2-x_2^2))$	$\varphi_4(x_1,x_2)=\frac{1}{8}(2+4x_2-3(x_1^2-x_2^2))$.

The standard finite element procedure (see, e.g., [10]) continues with computation of the element stiffness matrices $A_e = \{\alpha_{ij}\}_{i,j}^4$, $e \in \omega_h$. It follows the isoparametric technique using the reference element E basis functions. The assembling of A_e, $e \in \omega_h$ leads to the linear system of equations

$$A\mathbf{x} = \mathbf{b}\,. \tag{2}$$

The stiffness matrix $A = \{a_{ij}\}_{i,j=1}^N$ is sparse, symmetric and positive definite, with at most seven nonzero elements per row. Its structure is one and the same for MP and MV but it is different from the usual block banded case for conforming finite elements. The preconditioning strategy depends only on the structure of the matrix, but not on the type of the nodal basis functions. So we use again MP and MV notations only in the tables with numerical results.

3. Preconditioning strategy

The preconditioned conjugate gradient (PCG) method is known to be the best solution method of (2) for the case of large scale problems. We use PCG with preconditioner based on Modified Incomplete Cholesky (MIC(0)) factorization of sparse matrices. The essence of MIC(0) is given below. Some more details may be found in [2, 3].

Let us rewrite the real symmetric matrix A as a sum of its diagonal (D), strictly lower $(-\widetilde{L})$ and upper $(-\widetilde{L})^t$ triangular parts

$$A = D - \widetilde{L} - \widetilde{L}^t . \tag{3}$$

We say that the following approximate factorization of A

$$\mathcal{C}_{\mathrm{MIC}(0)}(\mathcal{A}) = (\mathcal{X} - \widetilde{\mathcal{L}})\mathcal{X}^{-\infty}(\mathcal{X} - \widetilde{\mathcal{L}})^{\sqcup}, \quad \mathcal{X} = \lceil\rangle\dashv\}(\S_{\infty}, \ldots, \S_{\mathcal{N}}) \tag{4}$$

is a *stable* MIC(0) factorization if $X > 0$ and thus $\mathcal{C}_{\mathrm{MIC}(0)}(\mathcal{A})$ is positive definite. Concerning construction and stability of MIC(0) factorization, the following theorem holds.

Theorem 1. *Let $A = \{a_{ij}\}$ be a symmetric real $N \times N$ matrix and let $A = D - \widetilde{L} - \widetilde{L}^t$ be the splitting (3) of A. Let us assume that*

$$\begin{aligned} \widetilde{L} &\geq 0 \\ A\mathbf{e} &\geq 0 \\ A\mathbf{e} + \widetilde{L}^t\mathbf{e} &> 0, \quad \mathbf{e} = (1, \cdots, 1)^t \in \mathbf{R}^N , \end{aligned}$$

i.e. that A is a weakly diagonally dominant matrix with non-positive off-diagonal entries and that $A + \widetilde{L}^t = D - \widetilde{L}$ is strictly diagonally dominant. Then the relation

$$x_i = a_{ii} - \sum_{k=1}^{i-1} \frac{a_{ik}}{x_k} \sum_{j=k+1}^{N} a_{kj}$$

gives only positive values and the diagonal matrix $X = diag(x_1, \cdots, x_N)$ defines a stable $MIC(0)$ factorization of A.

To solve the system with preconditioner $\mathcal{C}_{\mathrm{MIC}(0)}(\mathcal{A})$ we have to solve one system with lower triangular matrix, one - with upper triangular and one - with diagonal matrix. Two of these steps are based on recursive computations and therefore the resulting PCG algorithm is inherently sequential. To overcome this disadvantage (in the sense of a parallel implementation), we introduce a locally constructed approximation B of the original stiffness matrix A. To illustrate the basic ideas of the algorithm, we will consider a problem in a square domain Ω. Let us assume that the mesh is rectangular, and the numbering of the unknowns follows the columns of the nodes. The structure of the matrix A corresponding to such a model problem is shown in Fig. 1(a). We see that there

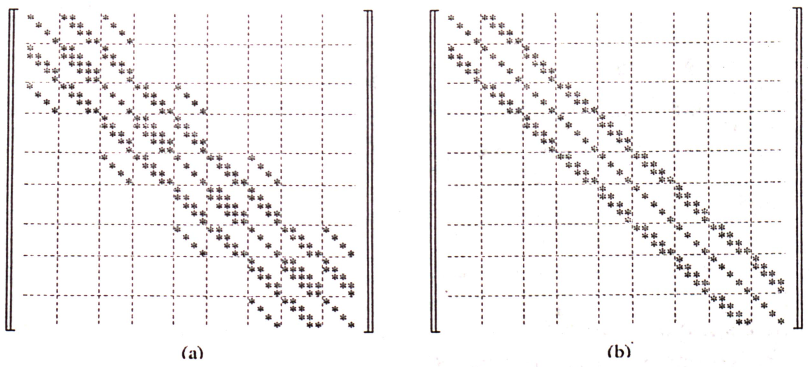

Figure 1: Structure of (a) matrix A and (b) introduced matrix B.

are not more than seven nonzero elements per row and only five of them are with steady indices in terms of co-diagonals. The rest are alternatively changing their position, forming either diagonal blocks on the main diagonal and five nonzero blocks per block row, or tridiagonal blocks on the main diagonal and three nonzero blocks per block row. The connectivity pattern corresponding to the stiffness matrix A_e for a given rotated bilinear nonconforming finite element is presented in Fig. 2(a). The diagonals of the dashed quadrilaterals are mapped

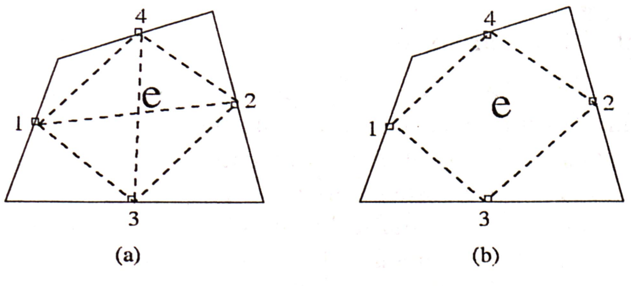

Figure 2: Connectivity pattern of (a) matrix A_e and (b) matrix B_e.

into the nonzero elements with "changing position". So if we "cut" these links in A_e, the corresponding elements in the global stiffness matrix will be vanished. Following this idea, we introduce the locally modified element stiffness matrix B_e with connectivity pattern shown in Fig. 2(b). The components of B_e are defined as follows:

$$A_e = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}, \quad B_e = \begin{bmatrix} \beta_{11} & 0 & \alpha_{13} & \alpha_{14} \\ 0 & \beta_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \beta_{33} & 0 \\ \alpha_{41} & \alpha_{42} & 0 & \beta_{44} \end{bmatrix},$$

where

$$\beta_{11} = \alpha_{11} + \alpha_{12}\,, \ \beta_{22} = \alpha_{22} + \alpha_{21}\,, \ \beta_{33} = \alpha_{33} + \alpha_{34}\,, \ \beta_{44} = \alpha_{44} + \alpha_{43}\,.$$

i.e. A_e and B_e have equal rowsums. Assembling the element-by-element defined matrices B_e we get a global matrix B with a regular block structure illustrated by Fig. 1(b). We solve the system (2) using PCG algorithm with preconditioner C for A defined by

$$\mathcal{C} = \mathcal{C}_{\mathrm{MIC}(0)}(\mathcal{B})\,.$$

A condition number model analysis of $B^{-1}A$ can be found in [8]. It is shown there that: a) the matrices A and B are spectrally equivalent; b) the conditions for a stable MIC(0) factorization hold for B; and c) the PCG convergence rates of the preconditioners $\mathcal{C}_{\mathrm{MIC}(0)}(\mathcal{B})$ and $\mathcal{C}_{\mathrm{MIC}(0)}(\mathcal{A})$ are very similar preserving the robustness of the pointwise incomplete factorization. The diagonal blocks of the matrix B allow for a parallel implementation of the resulting PCG algorithm. The related parallel algorithm is presented and analyzed in the next part of the paper.

4. Parallel implementation

How are the data and the computations distributed among the processors? What kind of communications are required? How good is the developed parallel solver? Such kind of questions are treated in this section.

Let us have NP processors denoted by $P_0, P_1, \dots, P_{NP-1}$. We also assume that the model square domain shown in Fig. 3 is decomposed into $n_1 \times n_2$ nonconforming quadrilateral elements. Each vertical line of nodes (column-wise numbering is used) corresponds to one of the diagonal blocks of the matrices A and B (see also Fig. 1). Note that the first line has n_2 nodes, the second - n_2+1 nodes, the third one has again n_2 nodes and so on. This leads to alternatively

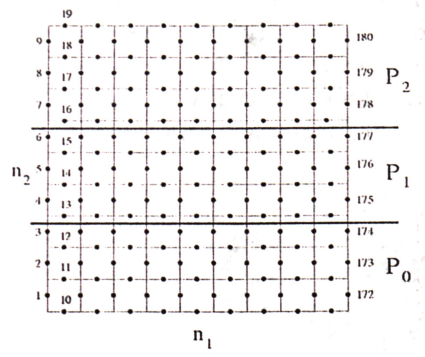

Figure 3: Data distribution - $NP = 3$, $N = n_1(2n_2 + 1) + n_2$, $n_1 = 9$, $n_2 = 9$.

changing size of the diagonal blocks of A (and B), namely odd blocks are of order n_2 while for even blocks it is $n_2 + 1$. The total number of blocks is $2n_1 + 1$ and for the size N of the discrete problem we have $N = n_1(2n_2 + 1) + n_2$.

We partition the domain into NP horizontal strips with approximately equal number of elements. Let $n_2 = q.NP + r$, $q, r \in \mathbb{N}$, $r < NP$. Then the first r strips have $q + 1$ lines of elements, the rest consist of q lines, and each line has n_1 elements. Each two strips with a common boundary are associated with processors with successive indices. The nodes on the boundaries of these strips belong by assumption to the processor with a larger number (see Fig. 3). This leads to a division of each of the block rows of the matrices A and B into strips with almost equal number of equations. All the vectors are distributed in the same manner. Some additional storage is allocated for communications. We have to point out that each processor contains a strip from each of the block equations, not one strip from the whole system.

How are computations partitioned and what kind of communications are required? Let us first recall that each PCG iteration includes the solution of one system with the preconditioner C, one matrix-vector multiplication with the original matrix A, two inner products, and three linked vector triads of the form $\mathbf{v} := \alpha\mathbf{v} + \mathbf{u}$.

Each processor calculates its part of the vector $\mathbf{v}$ and no communications are required for the triads. After computing the inner products corresponding to their parts of vectors, the processors have to perform one global reduction operation for one number to sum up the final result.

To obtain the components of $A\mathbf{v}$ for which a given processor, say the processor P_i, is responsible, it first has to communicate with processors P_{i-1} and P_{i+1}. P_i has to receive some components of the vector $\mathbf{v}$ and also to send to P_{i-1} and P_{i+1} the related data. After the transfer is completed, P_i is able to perform a matrix-vector multiplication with its part of A.

Fig. 4 illustrates the communications related to P_i not only for $A\mathbf{v}$ but

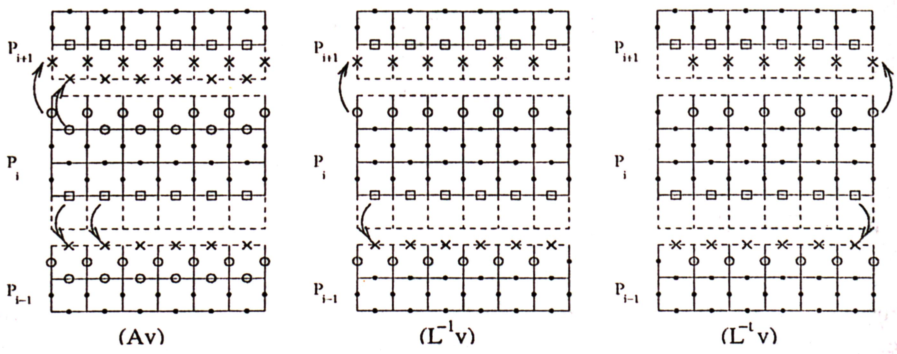

Figure 4: Communication scheme for matrix-vector multiplication (Av) and for solution of systems with lower triangular ($L^{-1}v$) and upper triangular ($L^{-t}v$) matrices.

also for the solution of systems with preconditioner C. Each of its components (Av), ($L^{-1}v$) and ($L^{-t}v$) represents the strip of the domain associated with the processor P_i and the near-by parts of the data in P_{i-1} and P_{i+1}. Dashed lines mean that data from these parts of the domain are not located in the processor, but additional storage is provided to ensure the communications. With $\circ$ are denoted the components which P_i sends to P_{i+1}, and with $\square$ - those to P_{i-1}. The sign $\times$ indicates the place where the transferred components are stored. Although it is not mapped in the figure, the processor P_i also have to receive the corresponding components from processors P_{i-1} and P_{i+1}.

Let us go back to the matrix-vector multiplication. Processor P_i has to send $2n_1 + 1$ elements of $\mathbf{v}$ (all $\circ$-s) to P_{i+1} and to receive from it n_1 (all $\square$-es of P_{i+1}) numbers. All these data are computed at the previous iteration step. They could be transfered in two portions - one to be sent from P_i, and one to be received by it. Next, P_i has to communicate with P_{i-1} in a similar way. Now it sends one portion of length n_1 and receives a vector of length $2n_1 + 1$. Note that if we have 2 processors only, one of the neighbours P_{i+1} and P_{i-1} will not exist. Hence a half of the above data are to be transferred.

To handle the system with the preconditioner (see (3), (4))

$$\mathcal{C}_{\mathrm{MIC}(0)}(\mathcal{B})\mathbf{w} \equiv (\mathbf{X}-\widetilde{\mathbf{L}})\mathbf{X}^{-1}(\mathbf{X}-\widetilde{\mathbf{L}})^{\mathbf{t}}\mathbf{w} = \mathbf{v}$$

one has to perform the following three steps: 1) find $\mathbf{y}$ from $L\mathbf{y} = \mathbf{v}$, where $L = X - \widetilde{L}$; 2) compute $\mathbf{y} := \mathbf{X}\mathbf{y}$; and 3) find $\mathbf{w}$ from $L^t\mathbf{w} = \mathbf{y}$. The matrix X is diagonal, distributed among the processors in the above described manner. The important advantage of the matrix B is that all of its diagonal blocks are diagonal. In this case, $\widetilde{L}$ has zero blocks $\widetilde{L}_{ii}$, $i = 1, \ldots, 2n_1 + 1$. The system $L\mathbf{y} = \mathbf{v}$ is solved using a standard forward recurrence. The block L_{11} is diagonal, divided into NP strips - one per each processor. So $L_{11}\mathbf{y_1} = \mathbf{v_1}$ is solved in parallel without any communications and with equally balanced amount of arithmetic operations. Then we have to determine $\mathbf{y_2}$ from $L_{21}\mathbf{y_1} + \mathbf{L_{22}y_2} = \mathbf{v_2}$. The component $\mathbf{y_1}$ is computed at the previous step but each processor has a strip of $\mathbf{y_1}$. The block L_{21} is two-diagonal and hence the processor P_i has to send one component to P_{i+1} and to receive another one from P_{i-1} (see Fig. 4($L^{-1}v$)). Then $L_{22}\mathbf{y_2} = \mathbf{v_2} - \mathbf{L_{21}y_1}$ is handled concurrently without any other communications. After that $L_{32}\mathbf{y_2} + \mathbf{L_{33}y_3} = \mathbf{v_3}$ has to be solved. Again, one component of $\mathbf{y_2}$ is to be transferred but now to processor P_{i-1}. The procedure continues till the last block of $\mathbf{y}$ is computed. The second step $\mathbf{y} := \mathbf{X}\mathbf{y}$ is communications-free and fully parallel. The last third step is handled in similar way as step 1) using a standard backward recurrence. The communications are illustrated in Fig. 4($L^{-t}v$). When we have only two processors, the transferred data at steps 1) and 3) are again two times less. It is important to note that here we are not able to combine components to be sent several at a time. In this sense there is still recursion, but it is for blocks, and each block is handled in parallel.

Now we are ready to derive estimates for the parallel times. Real time T_{NP} for solution of the system (2) with NP processors depends on computational complexity, type and amount of communications and characteristics of the specific parallel machine. We assume, like in [7], that the computations and communications are not overlapped, and therefore, T_{NP} is the sum of the computation and communication times. We also suppose that: a) the execution of M arithmetic operations (a. o.) on one processor takes time $T_a = M.t_a$, where t_a is the average unit time to perform one arithmetic operation on one processor (no vectorization); and b) the communication time to transfer M data elements from one processor to another is approximated by $T_{com} = l(t_s + M.t_w)$, where t_s is the start-up time, t_w is the incremental time necessary for each of the M words to be sent, and l is the graph distance between the processors. Since the parallel properties do not depend on the number of iterations, it is enough to evaluate the time T_{NP} per iteration and to use this time in the speed-up and efficiency analysis. The computational complexity per processor is $\mathcal{N}^{)\sqcup,\mathcal{NP}}_{\mathrm{PCG}} \approx \ni\triangle\mathcal{N}/\mathcal{NP}$ because: one solution of system with $C_{N\times N}$ requires $\approx 11N/NP$ a. o.; one matrix vector multiplication with $A_{N\times N}$ is done by $\approx 13N/NP$ a. o.; two inner products are performed for $4N/NP$ a. o.; and three linked vector triads $\mathbf{v} := \alpha\mathbf{v} + \mathbf{u}$ take $6N/NP$ a. o. Hence, the computation time is

$$T_a^{it} = 34\frac{N}{NP}.t_a\,. \tag{5}$$

The communication time is a sum of corresponding times for inner products, matrix-vector multiplication, and solution of a system with the preconditioner. The global reduction operation is presented as sum of one broadcast of one word and one gather again for one word: $T_{com}(in.pr.) = T(\text{bcast}, 1) + \text{T}(\text{gather}, 1)$. These times depend on the architecture (ring, 2D-mesh, H-cube) and the number of processors. Here we assume that they do not contribute to the leading terms of the total parallel times when $n_1 \gg NP$. In our case, the communications for $A\mathbf{v}$ and $C^{-\infty}\mathbf{v}$ are between processors with successive indices. They will be local if these processors are physically neighbours, i.e. $l = 1$. Hence, the communication time for $A\mathbf{v}$ is estimated by $T_{com}(A\mathbf{v}) = 2t_s + (3n_1 + 1)t_w$, when $NP = 2$ and $T_{com}(A\mathbf{v}) = 4t_s + 2(3n_1 + 1)t_w$, when $NP > 2$. For solution of a system with the preconditioner we have $T_{com}(C^{-\infty}\mathbf{v}) = \triangle\backslash_\infty(\sqcup_f + \sqcup_{\sqsupseteq})$ for the case $NP = 2$ and $T_{com}(C^{-\infty}\mathbf{v}) = \forall\backslash_\infty(\sqcup_f + \sqcup_{\sqsupseteq})$ for the case $NP > 2$ $(T_{com}(C^{-\infty}\mathbf{v}) = \mathcal{T}_{\rfloor\wr\Updownarrow}(\mathcal{L}^{-\infty}\mathbf{v}) + \mathcal{T}_{\rfloor\wr\Updownarrow}(\mathcal{L}^{-\sqcup}\mathbf{v}))$.

Hence, the total communication time per iteration (when $n_1 \gg NP$) is approximated by

$$T_{com}^{it} \approx \begin{cases} 4n_1.t_s + 7n_1.t_w\,, & \text{for } NP = 2\,, \\ 8n_1.t_s + 14n_1.t_w\,, & \text{for } NP > 2\,. \end{cases} \tag{6}$$

Therefore, if $NP > 2$, the time for one PCG iteration is

$$T_{NP}^{it} = T_a^{it} + T_{com}^{it} \approx 34\frac{n_1(2n_2 + 1) + n_2}{NP}.t_a + 8n_1.t_s + 14n_1.t_w\,. \tag{7}$$

What can we predict for the real performance time? The behaviour of the communication time should not depend on the number of processors. The only exception is the case $NP = 2$, when it should be two times less. The smaller computer system parameters t_s and t_w decrease the communication time and lead in general to a better speed-up and efficiency. In particular, if we have a shared memory computer, no communications are really performed. This means that the speed-up $S_{NP} = T_1/T_{NP}$ for this case will be close to its theoretical upper bound $S_{NP} \leq NP$. This is a basic advantage of the proposed algorithm. But the drawback is the coefficient n_1 in front of the start-up time t_s. Big ratios t_s/t_w and t_s/t_a could bear a performance penalty of more than 50% for large scale discrete problems.

5. Numerical tests and comments

The computer code implementing the presented parallel algorithm is developed using C language and MPI standard. The achieved performance on two Beowulf type Linux clusters referred as Parmac and Thea is analyzed. The Parmac cluster is located at Central Laboratory for Parallel Processing, Bulgarian Academy of Sciences. It consists of four dual processor Power Macintosh computers. Each node has 512 MB RAM and two processors Power PC G4 at 450 MHz clock frequency. The Thea cluster is located at Institute of Geonics, Academy of Sciences of Czech Republic, and consists of eight nodes. Each of them has 768 MB RAM and a single AMD Athlon processor at 1.4 GHz frequency. All the numerical results presented in this section are obtained using the C compiler from GNU Compiler Collection (GCC) and by LAM MPI version of the MPI standard. We have to note here that LAM MPI on Parmac has been compiled with a support for System V shared memory. It allows for a significant improvement in the local communication between the pair of processors at each of the nodes if shared memory is used instead of TCP/IP. Two ways of execution of an MPI program are possible on Parmac because of the dual processor nodes. First one is without a requirement "each process to be executed on separate node when possible" and the second one is with this requirement. They are ensured by the option N of "mpirun". For example, if we use "mpirun N -np 4 exec.out" the program exec.out will be executed on 4 processors allocated on 4 nodes - 1 processor per each node. This case will be denoted below by Parmac(N). If we use "mpirun -np 4 exec.out", i.e. "mpirun" without the option N (referred as Parmac), all working processors will be at two of the nodes. In such a way, shared memory will be used for communication between the processors from one node. To illustrate the properties of the parallel algorithm and the related code we consider the model Poisson equation in a unit square with homogeneous Dirichlet boundary conditions assumed at the bottom side. The partitioning of the domain is uniform where $n_1 = n_2 = n$. The size of the discrete problem is $N = n_1(2n_2+1)+n_2 = 2n(n+1)$. A relative stopping criterion $(C^{-1}r^{n_{it}}, r^{n_{it}})/(C^{-1}r^0, r^0) < \varepsilon$ is used in the PCG algorithm, where r^i stands for the residual at the i-th iteration step, $(\cdot,\cdot)$ is the standard Euclidean inner product, and $\varepsilon = 10^{-6}$. Two variants of the nodal basis functions were introduced in Section depending on the type of interpolation operator - mid-point or integral mid-value. Speed-up and efficiency coefficients for related parallel algorithms MP and MV are collected in Table 1 and Table

Table 1: Parallel PCG/MIC(0), Algorithm MP

		Parmac			Parmac(N)			Thea		
n, it	NP	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}
	1	16.74			16.82			9.26		
256	2	10.73	1.56	0.78	14.83	1.13	0.57	9.48	0.98	0.49
71	4	11.71	1.43	0.36	17.16	0.98	0.25	11.83	0.78	0.20
	8	10.92	1.53	0.19	20.07	0.84	0.11	11.08	0.84	0.11
	1	99.42			99.81			54.02		
512	2	63.69	1.56	0.78	68.71	1.45	0.73	41.11	1.31	0.66
104	4	50.22	1.98	0.50	65.44	1.53	0.38	41.14	1.31	0.33
	8	37.74	2.63	0.33	64.24	1.55	0.19	35.34	1.53	0.19
	1	577.00			567.11			288.27		
1024	2	373.19	1.55	0.78	340.12	1.67	0.84	198.70	1.45	0.73
148	4	233.55	2.47	0.62	274.27	2.07	0.52	153.35	1.88	0.47
	8	171.28	3.37	0.42	217.27	2.61	0.33	124.45	2.32	0.29

2 respectively. Both tables have a similar structure. There are two numbers in each box of the first column - the number n of the elements on each direction and the related number of iterations for that size of the discrete problem. The number of processors NP is given in the second column. The rest of the columns are grouped by three - one group per each way of the code's execution. Each of these groups has three fields - in the first one the measured cpu-time in seconds is given, and the rest two of them present the speed-up $S_{NP} = T_1/T_{NP}$ and the efficiency $E_{NP} = S_{NP}/NP$. The cpu-times T_{NP} are the best obtained from three measurements.

Our observations on these tables are the following:

- For a given "machine": cpu-times for MV are larger than those for MP because of the larger number of iterations; the speed-up and efficiency coefficients are approximately the same for MP and MV, which confirms that the properties of the parallel algorithm do not depend on the type of the basis functions.

- For a given number of processors: cpu-times for $NP = 1$ on Thea are approximately two times smaller than on Parmac for both ways of execution; the speed-up and respectively efficiency coefficients grow up for larger size of the problem, with only exception - the case $NP = 2$ for Parmac.

Table 2: Parallel PCG/MIC(0), Algorithm MV

		Parmac			Parmac(N)			Thea		
n, it	NP	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}
	1	19.09			19.16			10.55		
256	2	12.24	1.56	0.78	16.93	1.13	0.57	10.79	0.98	0.49
81	4	13.32	1.43	0.36	19.49	0.98	0.25	13.43	0.79	0.20
	8	12.93	1.48	0.19	22.85	0.84	0.11	13.04	0.81	0.10
	1	113.45			114.09			63.17		
512	2	73.00	1.55	0.78	78.49	1.45	0.73	49.91	1.35	0.68
119	4	57.66	1.97	0.49	77.14	1.48	0.37	47.04	1.34	0.34
	8	42.51	2.67	0.33	82.25	1.39	0.17	41.87	1.51	0.19
	1	635.63			639.18			326.63		
1024	2	409.39	1.55	0.78	383.58	1.67	0.84	223.35	1.46	0.73
167	4	264.88	2.40	0.60	309.19	2.07	0.52	173.26	1.89	0.47
	8	175.63	3.62	0.45	245.17	2.61	0.33	140.73	2.32	0.29

- For a given size of the problem: the general conclusion is, that on Parmac when a shared memory is used, the speed-ups are the best (except the case $NP = 2, n = 1024$), and on the Thea they are the worst; for all the machines, the speed-up and the efficiency are far away from the upper bounds $S_{NP} \le NP$, $E_{NP} \le 1$.

Some of the reasons for this behavior are hidden in the structures of the clusters.

- The processors of Thea are faster and therefore the times are smaller than those on Parmac.

- It was mentioned above that on Parmac, when a shared memory is possible to be used, the time for communication will be smaller, and respectively, the total cpu-times will be smaller than for the case when the option N is used in "mpirun".

- For relatively small sized problems, the total time on Parmac is influenced only by the communication time. But, when the size is larger, the two processes on one node compete for the memory access, and even the communications take smaller times than when the code is executed on two nodes, total time is larger. This is the reason for approximately the same speed-ups on Parmac for $NP = 2$, and for the better speed-up on Parmac(N) than on Parmac for $NP = 2$ and $n = 1024$.

We can not answer the question "Why are the speed-up and the efficiency relatively far away from their theoretical upper bounds?" if we look at the total cpu-times only. We can not see from these tables what is the particular behavior of the communication and computation times. We also can not say if the theoretical estimates from previous Section well represent the basic properties of the parallel algorithm. The distribution of the measured time for computations and communications for Algorithm MV for $n = 512$ and $n = 1024$ are presented in Table 3 and Table 4 respectively. First three columns give information about the machine, number of processors and related efficiency coefficients.

Table 3: Time distribution, Algorithm MV, $n = 512$, $N_{it} = 119$

machine	NP	E_{NP}	T_{NP}	T_a	T_c	T_c^{u*v}	T_c^{M*v}	T_c^{prec}
	1		109.85	109.85				
Parmac	2	0.77	71.63	70.35	1.28	0.02	0.08	1.18
	4	0.48	57.44	35.60	21.84	0.43	0.04	21.37
	8	0.32	43.25	18.17	25.08	1.26	0.06	23.76
	1		113.12	113.12				
Parmac(N)	2	0.73	77.48	55.82	21.66	0.04	0.29	21.33
	4	0.37	75.88	27.72	48.16	1.07	0.26	46.83
	8	0.19	73.77	16.78	56.99	1.32	0.31	55.36
	1		60.29	60.29				
Thea	2	0.66	45.89	30.55	15.34	0.14	0.23	14.97
	4	0.32	46.74	16.10	30.64	0.23	0.21	30.20
	8	0.18	42.76	8.74	34.02	0.30	0.20	33.52

The measured times are presented in the rest of the columns. Here T_{NP} stands for the total cpu-time for solution of the problem on NP processors, T_a and T_c are the times for computations and communications respectively. Following the theoretical estimates, the communications are split by type - for inner products T_c^{u*v}, for matrix-vector multiplication T_c^{M*v}, and for solution of the system with the preconditioner T_c^{prec}. We have used blocking send and receive operations of MPI, i.e. the computations and communications are not overlapped. Hence the following relations hold:

$$T_{NP} = T_a + T_c, \qquad T_c = T_c^{u*v} + T_c^{M*v} + T_c^{prec}.$$

Table 4: Time distribution, Algorithm MV,$n = 1024$, $N_{it} = 167$

machine	NP	E_{NP}	T_{NP}	T_a	T_c	T_c^{u*v}	T_c^{M*v}	T_c^{prec}
	1		637.30	637.30				
Parmac	2	0.77	412.62	407.71	4.91	0.10	0.23	4.58
	4	0.60	264.21	201.20	63.01	0.37	0.16	62.48
	8	0.47	170.18	100.64	69.54	5.07	0.22	64.25
	1		638.94	638.94				
Parmac(N)	2	0.84	381.55	319.48	62.07	0.14	0.73	61.20
	4	0.52	308.14	157.79	150.35	4.63	0.67	145.05
	8	0.32	247.03	96.96	150.07	5.19	0.83	144.05
	1		324.81	324.81				
Thea	2	0.73	223.95	171.27	52.68	0.93	3.02	48.73
	4	0.47	172.92	84.54	88.38	0.84	0.81	86.73
	8	0.29	140.87	44.51	96.36	0.81	0.58	94.97

We see that the computation time for all the machines decrease almost two times when the number of processors is increased by two. The exceptions for Parmac are explained by the memory access competition for large problems. So all the drawbacks come from the communications as it was expected. Our prediction that the times for communication for inner products and for matrix-vector multiplications will be negligible with respect to the communications for solving the preconditioned system are also confirmed. Because of faster processors, the time for computations on Thea are two times smaller than those on Parmac, but the ratio for communications is not the same. That is why the efficiency on Thea is smaller in general. There is enormous difference between times T_c for $NP = 2$ on Parmac and Parmac(N). Shared memory is used for communication on Parmac versus TCP/IP for Parmac(N). This shows the potential of our parallel algorithm and the related code for shared memory implementations. The expectation that the time for communications does not depend on the number of processors is also confirmed - it is approximately the same for $NP = 4$ and $NP = 8$ for all the machines. The case $NP = 2$ is analyzed separately and its time T_c is predicted to be two times smaller than that for $NP = 4$ which holds in practice for the distributed memory case.

What can we do to improve the performance? Approaches in two directions are possible depending on the architecture. For shared memory machines we can use Open MP, or combination of MPI and Open MP for clusters of shared memory computers. For distributed memory we can locally rearrange the computations in solution of the system with preconditioner and to use non-blocking send and receive operations to allow overlapping of the computations and communications.

6. Conclusions

A scalable parallel MIC(0) preconditioner has been presented in the paper. We have analyzed theoretically some of its advantages and disadvantages. The experimental results have shown promising features for both shared and distributed memory architectures as well as for clusters of shared memory machines. Some approaches for improvement of the real performance using Open MP and overlapping of computations and communications have been also derived. Our plans for future work on the topic include modifications to improve the performance and generalization of the considered algorithm to the 3-D case.

Acknowledgements

This work has been supported in part by MES Grants MM-801/98, MU-I-901/99 and by the Center of Excellence BIS-21 Grant ICA1-2000-70016. The authors would like to thank to the group in Institute of Geonics in Ostrava for the possibility to use their cluster Thea. The participation of G. Bencheva at the Second Workshop on Parallel Matrix Algorithms and Applications with a talk on this article was due to the sponsorship of the Swiss National Science Foundation.

References

[1] D.N. Arnold and F. Brezzi, Mixed and nonconforming finite element methods: implementation, postprocessing and error estimates, *RAIRO, Model. Math. Anal. Numer.* **19** (1985), 7–32.

[2] R. Blaheta, Displacement decomposition – incomplete factorization preconditioning techniques for linear elasticity problems, *Numer. Linear Algebra Appl.* **1** (1994), 107–126.

[3] I. Gustafsson, Modified incomplete Cholesky (MIC) factorization, In: D.J. Evans, ed., *Preconditioning Methods; Theory and Applications*, Gordon and Breach, 1984, 265–293.

[4] I. Gustafsson and G. Lindskog, On parallel solution of linear elasticity problems. Part I: Theory, *Numer. Linear Algebra Appl.* **5** (1998), 123–139.

[5] I. Gustafsson and G. Lindskog, On parallel solution of linear elasticity problems. Part II: Methods and some computer experiments, *Numer. Linear Algebra Appl.* **9** (2002), 205–221.

[6] R. Rannacher and S. Turek, Simple nonconforming quadrilateral Stokes Element, *Numer. Methods for PDEs* 8, No 2 (1992), 97–112.

[7] Y. Saad and M. Schultz, Data communication in parallel architectures, *Parallel Computing* **11** (1989), 131–150.

[8] G. Bencheva and S. Margenov, Parallel incomplete factorization preconditioning of rotated linear FEM systems. *Computers & Mathematics with Applications*. Submitted.

[9] R. Lazarov and S. Margenov, On a two-level parallel MIC(0) preconditioning of Crouzeix-Raviart non-conforming FEM systems, in: I. Dimov, I. Lirkov, S. Margenov and Z. Zlatev, eds., *Numerical Methods and Applications, LNCS*, **2542** (Springer-Verlag, Berlin, Heidelberg, 2003), 191–200.

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

Central Laboratory for Parallel Processing
Bulgarian Academy of Sciences
Acad. G. Bonchev str. Bl. 25-A
1113 Sofia, BULGARIA
e-mail: gery@fmi.uni-sofia.bg

Received 29.04.2003