MPI implementation of a PCG solver for nonconforming FEM problems: overlapping of communications and computations

Gergana Bencheva[*], Svetozar Margenov[†], Jiří Starý[‡]

February 27, 2006

Abstract

New theoretical and experimental results concerning a recently introduced parallel preconditioner for the solution of large nonconforming FE linear systems are presented. The studied algorithm is based on the modified incomplete Cholesky factorization MIC(0) applied to a locally constructed approximation of the original stiffness matrix. The overlapping of communications and computations is possible due to a suitable reordering of the computations applied in the MPI code. Theoretical estimates for the execution time of the modified algorithm are derived. The obtained improvement of the real performance is illustrated by numerical tests on a Beowulf-type Linux cluster, on a Sun symmetric multiprocessor and on an SGI Altix supercluster.

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

Key words: parallel algorithms, overlapping communications with computations, 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. This work is focused on the numerical solution of second order elliptic boundary value problems discretized by rotated bilinear nonconforming finite elements on quadrilaterals. The considered elements are originally proposed in [13] as a cheapest stable finite element (FE) approximation of the Stokes problem.

The parallel preconditioned conjugate gradient (PCG) solver for the related elliptic FE system was recently introduced in [2]. Our aim is to improve the performance of

[*] *Johann Radon Institute for Computational and Applied Mathematics, Austrian Academy of Sciences, Altenbergerstraße 69, A-4040 Linz, Austria (on leave from Institute for Parallel Processing, Bulgarian Academy of Sciences, since July 1, 2005)*

[†] *Institute for Parallel Processing, Bulgarian Academy of Sciences, Acad. G. Bontchev Str., Bl. 25A, 1113 Sofia, Bulgaria*

[‡] *Institute of Geonics, Academy of Sciences of the Czech Republic, Studentská 1768, 70800 Ostrava, Czech Republic*

its message passing interface (MPI) implementation. The preconditioner is based on the modified incomplete Cholesky factorization MIC(0) applied to a locally constructed approximation of the original stiffness matrix. The properties of the method have been analyzed already in [3], where some numerical results obtained on two Beowulf-type Linux clusters are compared with theoretically derived estimates of the execution times.

The rate of convergence and the computational complexity of one iteration are the mailstones in evaluation of sequential iterative solvers. The speed-up and efficiency coefficients play a key role in the analysis of parallel algorithms. The following models for the computation time $T_a = \mathcal{N} * t_a$ and the communication time $T_{com} = c_1 * t_s + c_2 * t_w$ will be used. Here, $\mathcal{N}$ is the maximal number of operations per processor, c_1 characterizes the number of stages at which communications are needed and c_2 is the total amount of the transferred words. Both c_1 and c_2 can be constants or functions of the number of processors and/or the problem size. The parameters t_a, t_s and t_w depend on the parallel computer. They represent the average unit time to perform one arithmetic operation on a single processor (t_a), the start-up time to initiate a communication (t_s) and the time necessary to send one word between two processors (t_w). The largest one is t_s and it can be hundreds and even thousands times larger than t_w. This is the reason why one and the same parallel solver can have quite different behaviour on different machines. Nowadays, when there exist various parallel architectures and the trend is to be able to execute programs anywhere in the world via GRID infrastructure, it is very important to design parallel solvers scalable on as many parallel architectures as possible.

The paper is organized as follows. The needed background of the bilinear nonconforming FE discretization, the essence of MIC(0) and the preconditioning strategy, are given in Section 2. The parallel implementation of the resulting PCG algorithm is described in Section 3, together with estimates of the execution time (as they were derived in [3]) and numerical experiments on three parallel architectures. In Section 4, a suitable reordering of the computations in the MPI code is proposed. This approach allows the usage of nonblocking send and receive operations, i.e. overlapping of communications and computations. Theoretical estimates for the execution time of the modified algorithm are derived in 4.3. The achieved improvement of the real performance is illustrated in 4.4 by numerical tests on a Beowulf-type Linux cluster, on a Sun symmetric multiprocessor and on an SGI Altix supercluster.

2 Preliminaries

The subject of investigations in this work is an iterative numerical method for efficient solution of one class of large systems of linear algebraic equations. The latter are obtained after discretization of a second order elliptic boundary value problem

$$\left|\begin{array}{rcll} -\nabla \cdot (a(y)\nabla u(y)) & = & f(y), & y = (y_1, y_2) \in \Omega \subset \mathbb{R}^2, \\ u & = & \mu(y), & y \in \Gamma_D \ (meas(\Gamma_D) \neq 0), \\ (a(y)\nabla u(y)) \cdot n & = & g(y), & y \in \Gamma_N. \end{array}\right. \tag{1}$$

Here $\nabla u(y)$ denotes the gradient of u and $\nabla \cdot q$ denotes the divergence of the vector q. Further, we assume that Ω is a polygonal domain in $\mathbb{R}^2$, $f(y)$, $\mu(y)$, $g(y) \in L^2(\Omega)$

are given functions and n is the outward unit vector normal to the boundary $\Gamma = \partial\Omega$, $\Gamma = \bar{\Gamma}_D \cup \bar{\Gamma}_N$. We suppose also that $a(y) = \{a_{ij}(y)\}_{i,j=1}^2$ is a symmetric matrix, where $a_{ij}(y)$ are piece-wise smooth functions on $\bar{\Omega}$ satisfying the uniform positive definiteness condition of the matrix $a(y)$.

The structure of the resulting system depends on the type of discretization. Finite difference (FDM) and finite element (FEM) methods are among the most popular and frequently used approaches. We use here quadrilateral nonconforming finite elements based on rotated bilinear shape functions. They are proposed by R. Rannacher and S. Turek [13] to solve the Stokes problem using nodal basis functions of lower degree. Another application of these elements is to obtain uniform approximation of the elasticity problem in the case of almost incompressible materials, as it is done by P. Hansbo and M. Larson [10]. The recent efforts in development of efficient solution methods for nonconforming FEM systems are inspired by their importance for various applications in scientific computations and engineering.

An essential feature of the nonconforming FEM is the regular sparsity structure of the stiffness matrix for nonregular meshes. This type of elements gives also specific opportunities for parallel implementation. Our goal is to obtain a scalable parallel preconditioner.

In this section we review briefly how the discrete system for this particular type of elements is obtained. After that we present the essence of the modified incomplete Cholesky (MIC(0)) factorization of sparse symmetric positive definite matrices which is in the root of the considered algorithm. At the end of the section we sketch the proposed in [2] approach to construct the parallel preconditioner.

2.1 Nonconforming FEM discretization

We consider the case of anisotropic coefficients and homogeneous boundary conditions. Since the stiffness matrix is not changed for the case of nonhomogeneous boundary conditions, this assumption does not lead to restrictions in the solution method.

The Galerkin variational formulation of problem (1) is: *for a given* $f \in L^2(\Omega)$ *find a function* $u \in H_D^1(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$, *satisfying*

$$\mathcal{A}(u, v) = (f, v) \qquad \forall v \in H_D^1(\Omega), \tag{2}$$

where

$$\mathcal{A}(u, v) = \int_\Omega a(y)\Big(\frac{\partial u}{\partial y_1}\frac{\partial v}{\partial y_1} + \varepsilon\frac{\partial u}{\partial y_2}\frac{\partial v}{\partial y_2}\Big)dy$$

and $\varepsilon > 0$ *is the anisotropy parameter.*

The discrete problem corresponding to (2) is constructed in the following way. The domain Ω is divided into convex quadrilaterals $e \in \omega_h$. The partitioning ω_h is aligned with the discontinuities of the coefficient $a(y)$ so that over each element $e \in \omega_h$ the function $a(y)$ is smooth. We suppose as well that the partitioning is quasi-uniform with a characteristic mesh-size h.

The rotated bilinear nonconforming finite elements on quadrilaterals (see [13]) are implemented to obtain the finite element space V_h relevant to ω_h. Two alternative

constructions of V_h (denoted by MP and MV) are considered depending on the choice of the reference element nodal basis functions $\{\hat{\phi}_i\}_{i=1}^4$.

The unit square $(-1,1)^2$ with sides Γ_j, $j=1,\ldots,4$, parallel to the coordinate axes and nodes $j=1,\ldots,4$ in the mid-points of the sides (see Figure 1) is used as a reference element E in the definition of the isoparametric rotated bilinear element.

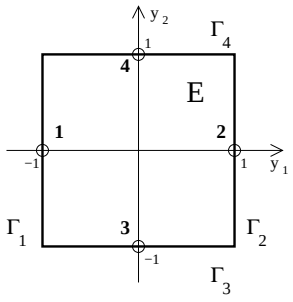


Figure 1: Reference nonconforming finite element E.

Mid-point and integral mid-value interpolation operators are used to determine the reference element nodal basis functions $\widehat{\phi}_i \in S_p$, $S_p = span\{1, y_1, y_2, y_1^2 - y_2^2\}$ for the variants MP and MV respectively. The standard interpolation conditions $\widehat{\phi}_i(j) = \delta_{i,j}$, $i,j = 1,\ldots,4$ are used for MP, where $\delta_{i,j}$ is the Kronecker symbol. The conditions in the case MV are

$$\frac{1}{|\Gamma_j|}\int_{\Gamma_j} \widehat{\phi}_i \, dy = \delta_{i,j},\ i,j = 1,\ldots,4\,,$$

where Γ_j is the side of E containing the node j. The expressions for $\hat{\phi}_i$ in these two cases are as follows.

Basis MP	Basis MV
$\widehat{\phi}_1(y_1,y_2) = \frac{1}{4}(1 - 2y_1 + (y_1^2 - y_2^2))$	$\widehat{\phi}_1(y_1,y_2) = \frac{1}{8}(2 - 4y_1 + 3(y_1^2 - y_2^2))$
$\widehat{\phi}_2(y_1,y_2) = \frac{1}{4}(1 + 2y_1 + (y_1^2 - y_2^2))$	$\widehat{\phi}_2(y_1,y_2) = \frac{1}{8}(2 + 4y_1 + 3(y_1^2 - y_2^2))$
$\widehat{\phi}_3(y_1,y_2) = \frac{1}{4}(1 - 2y_2 - (y_1^2 - y_2^2))$	$\widehat{\phi}_3(y_1,y_2) = \frac{1}{8}(2 - 4y_2 - 3(y_1^2 - y_2^2))$
$\widehat{\phi}_4(y_1,y_2) = \frac{1}{4}(1 + 2y_2 - (y_1^2 - y_2^2))$	$\widehat{\phi}_4(y_1,y_2) = \frac{1}{8}(2 + 4y_2 - 3(y_1^2 - y_2^2))$

For a given element $e \in \omega_h$, let $\psi_e : E \to e$ be the corresponding bilinear transformation. The element nodal basis functions determined by the relations

$$\{\phi_i\}_{i=1}^4 = \{\hat{\phi}_i \circ \psi_e^{-1}\}_{i=1}^4$$

define a nonconforming basis in the space V_h.

Then the finite element formulation is: *find a function $u_h \in V_h$, satisfying the equation*

$$\mathcal{A}_h(u_h, v_h) = (f, v_h) \qquad \forall v_h \in V_h\,, \tag{3}$$

where

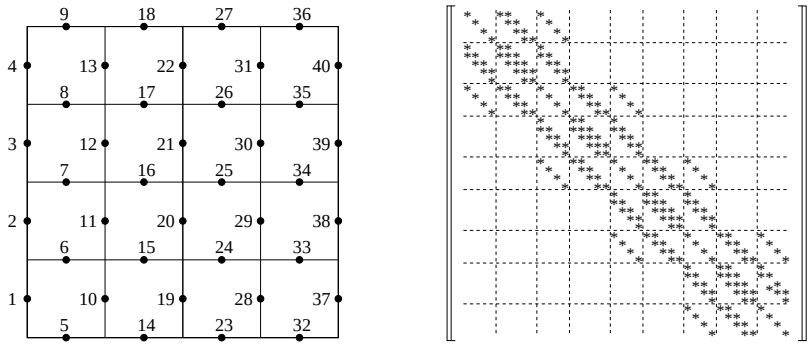
$$\mathcal{A}_h(u_h, v_h) = \sum_{e\in\omega_h} \int_e a(e)\Big(\frac{\partial u_h}{\partial y_1}\frac{\partial v_h}{\partial y_1} + \varepsilon \frac{\partial u_h}{\partial y_2}\frac{\partial v_h}{\partial y_2}\Big) dy\,.$$

Here $a(e)$ is defined as the integral mid-value $a(e) = \dfrac{1}{|e|}\displaystyle\int_e a(y)dy$ over the current element $e \in \omega_h$. Let us note that we allow strong coefficient jumps through the interface boundaries between the elements.

The standard finite element procedure (see, e.g., [15]) continues with computation of the element stiffness matrices $A_e = \{\alpha_{ij}^e\}_{i,j=1}^4$, $\alpha_{ij}^e = \displaystyle\int_e a(e)\Big(\frac{\partial\phi_i}{\partial y_1}\frac{\partial\phi_j}{\partial y_1} + \varepsilon\frac{\partial\phi_i}{\partial y_2}\frac{\partial\phi_j}{\partial y_2}\Big)dy$, $e \in \omega_h$. 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

$$A\mathbf{x} = \mathbf{b}\,. \tag{4}$$

The stiffness matrix $A = \{a_{ij}\}_{i,j=1}^N$ is sparse, symmetric and positive definite, with at most seven nonzero elements per row. The structure of A is one and the same for the cases MP and MV but depends on the triangulation and numbering of the unknowns. For a model problem, square mesh and column-wise nodes' ordering (Figure 2 (a)), it is presented in Figure 2 (b). The sparsity pattern is different from the usual block



(a) nodes' numbering; (b) stiffness matrix $A_{N\times N}$

Figure 2: Nodes' numbering and nonzero structure of the stiffness matrix for standard triangulation using nonconforming rotated bilinear finite elements.

banded case for conforming finite elements. The preconditioning strategy used in this work depends only on the structure of the matrix, but not on the type of the nodal basis functions.

2.2 MIC(0) factorization

The algorithm presented in the remainder of the paper is based on the modified incomplete Cholesky (MIC(0)) factorization of sparse matrices. The essence of MIC(0) is given below. Some more details may be found in [5, 7].

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

$$A = D - \widetilde{L} - \widetilde{L}^t\,. \tag{5}$$

Then we consider the approximate factorization of A in the following form

$$\mathcal{C}_{\mathrm{MIC(0)}}(A) = (X - \widetilde{L})X^{-1}(X - \widetilde{L})^t\,, \tag{6}$$

where $X = diag(x_1, \cdots, x_N)$ is a diagonal matrix determined by the condition of equal rowsums $\mathcal{C}_{\mathrm{MIC}(0)}\underline{e} = A\underline{e}$, $\underline{e} = (1, \cdots, 1)^t \in \mathbb{R}^N$.

We say that the factorization (6) of A is a *stable* MIC(0) factorization if $X > 0$ and thus $\mathcal{C}_{\mathrm{MIC}(0)}(A)$ is positive definite. Concerning the introduced stability, the following theorem holds.

Theorem 1 *Let $A = \{a_{ij}\}$ be a symmetric real $N \times N$ matrix and $A = D - \widetilde{L} - \widetilde{L}^t$ be the splitting (5) of A. Let us also assume that*

$$\widetilde{L} \geq 0\,, \quad A\mathbf{e} \geq 0\,, \quad A\mathbf{e} + \widetilde{L}^t\mathbf{e} > 0\,, \quad \mathbf{e} = (1, \ldots, 1)^t \in \mathbb{R}^N\,,$$

i.e. A is a weakly diagonally dominant matrix with nonpositive 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.*

2.3 Preconditioning strategy

Here, we briefly present the approach proposed in [2]. To solve the system of linear algebraic equations we use the preconditioned conjugate gradient (PCG) method. Two steps are performed to construct the preconditioner.

At first, a local modification with a diagonal compensation is applied to the stiffness matrix A. Its geometrical sense is the following. Each node P of the mesh is connected with the nodes from two neighbouring quadrilateral elements as shown in Figure 3 (a). Two of the links are "cut" (see Figure 3 (b)) in the locally modified element stiffness

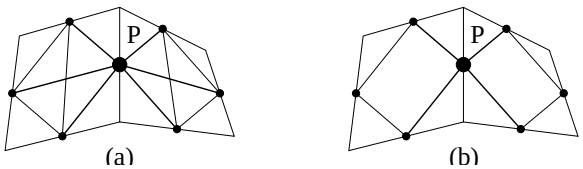


Figure 3: The connectivity pattern.

matrices. The resulting global matrix B allows for a stable MIC(0) factorization.

At the second step, MIC(0) factorization of B is performed, i.e. the preconditioner for A is $\mathcal{C} = \mathcal{C}_{\mathrm{MIC}(0)}(B)$. It has a specific well parallelizable block structure preserving the robustness of the pointwise incomplete factorization.

The steps needed to solve a system with a preconditioner $\mathcal{C}_{\mathrm{MIC}(0)}(A)$ are based on recursive computations and therefore the resulting PCG algorithm is inherently sequential. The locally constructed approximation B of the original stiffness matrix A provides a possibility of parallel implementation.

The structure of the matrices A and B is illustrated in Figure 4. They correspond to a problem in a square domain, discretized by a rectangular $n_1 \times n_2$ mesh with column-wise nodes' numbering (see also Figure 2). Then $N = n_1(2n_2 + 1) + n_2$ is the number of degrees of freedom.

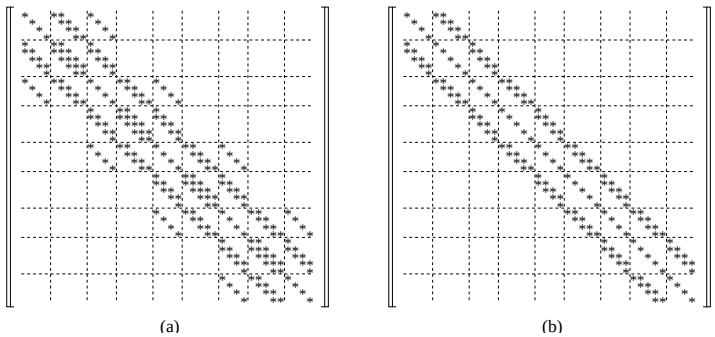


Figure 4: Structure of the matrix A (a) and the locally modified matrix B (b).

A condition number model analysis of $B^{-1}A$ for the isotropic (i.e. $\varepsilon = 1$) and the anisotropic (i.e. $\varepsilon \neq 1$) cases is presented in [2] and [4] respectively. The obtained results are given in the following theorem (see [4] for the proof).

Theorem 2 *Consider the anisotropic elliptic problem (3), discretized on a square $n \times n$ mesh of nonconforming rotated bilinear finite elements. Then:*

(i) the sparse approximation B of the stiffness matrix A satisfies the conditions for a stable MIC(0) *factorization;*

(ii) the matrices B and A are spectrally equivalent where the next relative condition number estimates hold uniformly with respect to the mesh parameter $h = \frac{1}{n}$ and any possible coefficients jumps:

$$\kappa((B^{MP})^{-1}A^{MP}) \leq 2 \quad for \quad \varepsilon \in [\frac{1}{2}, 1]\,,$$
$$\kappa((B^{MV})^{-1}A^{MV}) \leq 3 \quad for \quad \varepsilon \in [\frac{1}{3}, 1]\,.$$

It is shown in [2] that the computational complexity of one PCG iteration is

$$\mathcal{N}_{it}^{PCG/\mathrm{MIC}(0)}(A^{-1}\mathbf{b}) \approx 34\,N.$$

The rate of convergence depends on $\kappa(\mathcal{C}^{-1}A)$, where $\mathcal{C} = \mathcal{C}_{\mathrm{MIC}(0)}(B)$. For model problems $\kappa(\mathcal{C}^{-1}A)$ is analyzed by I. Gustafsson [6, 7] and R. Blaheta [5]. They have shown that the rate of convergence is of order $\mathcal{O}(N^{\frac{1}{4}})$, i.e. $\kappa(\mathcal{C}^{-1}A) = \mathcal{O}(N^{\frac{1}{2}})$.

In [4] it is experimentally shown that: a) the PCG convergence rate for various values of the anisotropy parameter ε and both types of MP and MV of the basis functions is of order $\mathcal{O}(N^{\frac{1}{4}})$; b) for the intervals, where $\kappa(B^{-1}A)$ is uniformly bounded (cf. Theorem 2 *(ii)*) the number of iterations slowly increases when ε decreases; c) when $\kappa(B^{-1}A) \to \infty$, i.e. $\varepsilon \to 0$, the number of iterations rapidly increases in both cases MP and MV.

3 Parallel implementation

We present in this section the parallel algorithm proposed and studied in [2, 3]. First, we focus our attention on the questions ,,How are the data and computations distributed among the processors? What kind of communications are required?" Next, we summarize the obtained theoretical estimates for the parallel execution times. The exposition at this stage is based on the theoretical results in [3]. At the end of the section, we illustrate the real performance of the solver with new numerical tests on a Beowulf-type Linux cluster, on a Sun symmetric multiprocessor and on an SGI Altix supercluster. The first numerical tests presented in [3] are performed on two Beowulf-type Linux clusters with up to 9 processors. Here, the experimental part is enriched by results from numerical tests performed on two new parallel architectures. The observations and conclusions from [3] are kept in mind to get a more detailed picture of the real performance of the considered parallel solver.

3.1 Data distribution, computations and communications

The model square domain shown in Figure 5 is decomposed into $n_1 \times n_2$ nonconforming

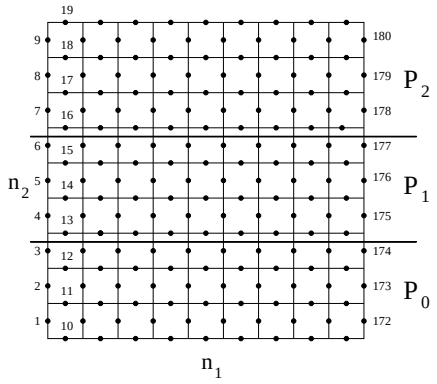


Figure 5: Data distribution: $N_p = 3$, $N = n_1(2n_2 + 1) + n_2$, $n_1 = n_2 = 9$.

quadrilateral elements. For the parallel implementation of the considered algorithm we suppose that N_p processors denoted by P_0, P_1, ... , P_{N_p-1} are available. The domain is then partitioned into N_p horizontal strips with approximately equal number of elements (see Figure 5). Each two strips with a common boundary are associated with processors with successive indices. 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, i.e. each processor contains a strip from each of the block equations, not one strip from the whole system. All the vectors are distributed in the same manner. Some additional storage is allocated for communications. We have to point out that the blocks on the diagonal of B are diagonal and therefore they allow a parallel implementation of the considered PCG algorithm. There is still recursion but it is for the blocks, and each block equation is handled in parallel.

How are computations partitioned and what kind of communications are required? Detailed answer of this question is given in [3]. Each PCG iteration includes the solution of one system with the preconditioner $\mathcal{C}$, one matrix-vector multiplication with the original matrix A, two inner products, and three linked vector triads of the form $\mathbf{v} := \alpha\mathbf{v} + \mathbf{u}$. Here we point out that: a) computations are equally distributed among the processors; b) PCG communications needed for the inner products are global, and those for the matrix vector multiplication and for the solution of the system with the preconditioner are local. For the linked vector triads no communications are required.

Figure 6 illustrates the communications related to P_i for the matrix-vector multipli-

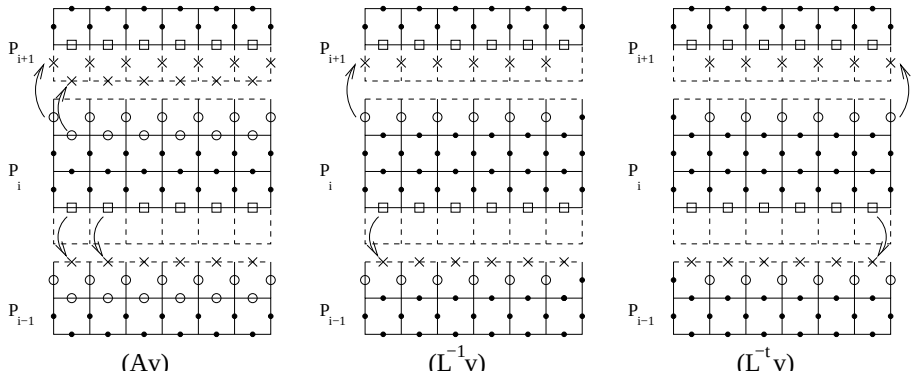


Figure 6: Communication scheme for matrix-vector multiplication (Av) and for solution of systems with lower triangular ($L^{-1}v$) and upper triangular ($L^{-t}v$) matrices.

cation $A\mathbf{v}$ and for the solution of the systems with the lower ($L = X - \widetilde{L}$) and the upper (L^T) triangular factors of the preconditioner $\mathcal{C} = \mathcal{C}_{\mathrm{MIC}(0)}(B)$ (see (5),(6)).

Each processor P_i has to send to P_{i+1} the components of a vector corresponding to the nodes denoted by a circle and to P_{i-1} – those denoted by a square. The sign $\times$ stands for the place where the transferred data is stored. Although it is not shown in the figure, P_i has to receive also a similar set of data from P_{i-1} and P_{i+1}. To implement the matrix vector multiplication, the required components are determined on the previous iteration step and they are sent at two communication stages for each pair of processors. For the solution of systems with L and L^T this is not possible and one number is transferred at each of the needed $2n_1$ communication stages. The next subsection deals with the question "How good is the described parallel solver?"

3.2 Parallel complexity

The parallel time T_{N_p} for the solution of the system (4) with N_p processors depends on the computational complexity, the type and the amount of communications and the characteristics of the specific parallel machine. The theoretical estimate for T_{N_p} is derived in [3] under the following assumptions: a) the computations and the communications are not overlapped, and therefore, T_{N_p} is a sum of the computation and communication times; b) the execution of M arithmetic operations (a. o.) on each processor takes time $T_a = M.t_a$, where t_a is the average unit time to perform one arithmetic operation

on each processor (no vectorization); c) 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_{N_p} per iteration and to use this time in the speed-up and efficiency analysis.

It is shown in [3] that if $N_p > 2$ and $n_1 \gg N_p$, then the time for one PCG iteration is

$$T^{it}_{N_p} = T^{it}_a + T^{it}_{com} \approx 34\frac{n_1(2n_2+1)+n_2}{N_p} \cdot t_a + 8n_1 \cdot t_s + 14n_1 \cdot t_w \,. \tag{7}$$

What can we predict for the real performance time? The qualifying part of the needed communications are local and the performance of the algorithm weakly depends on the number of processors and the parallel architecture. The parameters t_s, t_w and t_a are machine dependent. The coefficients in front of t_s and t_w in the estimate of the communication time depend on the problem size. This is the reason why the performance depends substantially on the values of t_s, t_w and t_a. 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, the speed-up $S_{N_p} = \dfrac{T_1}{T_{N_p}}$ for a shared memory computer will be close to its theoretical upper bound $S_{N_p} \leq N_p$. The reason is that the substantially faster access to the memory is used in practice for shared memory machines instead of communications between the processors. From the estimate (7) it follows that big ratios $\dfrac{t_s}{t_w}$ and $\dfrac{t_s}{t_a}$ could bear a performance penalty of more than 50% for large scale discrete problems.

3.3 Numerical tests I

The computer code implementing the presented parallel algorithm is developed using C language and MPI standard. Numerical tests are performed on three parallel architectures of different type with respect to the number of processors and the access to the memory. The names of the machines (as they will be referred in the exposition), their basic characteristics (with a web-page for more details) and location are as follows:

Thea – cluster of 8 computing nodes, having a single AMD Athlon processor at 1.4 GHz and 1.5 GB of RAM each, connected via 2 standard FastEthernet networks (http://www.ugn.cas.cz/math/index.php?p=other/computer/thea.php), *Institute of Geonics, Academy of Sciences of the Czech Republic, Ostrava, Czech Republic.*

Simba – separate domain of a Sun Fire 15k server based on a Sun Fireplane interconnect architecture, including 36 UltraSPARC III+ CPUs at 900 MHz and 36 GB of RAM (http://www.uppmax.uu.se/ComputerSystems/Ngorongoro/NgUserGuide.html), *Department of Information Technology, Uppsala University, Sweden.*

Lilli – Altix 3700 supercluster from the family SGI Altix 3000 with two nodes, each with 64 Intel Itanium-2 processors at 900 MHz and 64 GB of RAM connected via Giga-

bit Ethernet (http://www.zid.uni-linz.ac.at/HARDWARE/vorhanden/altix.html), *Johannes Kepler University, Linz, Austria.*

Simba and Lilli belong to the group of non-uniform shared memory architectures, while Thea represents the distributed memory parallel computers.

We consider as a test example 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 $(\mathcal{C}^{-1}r^{n_{it}}, r^{n_{it}})/(\mathcal{C}^{-1}r^0, r^0) < \varepsilon$ 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 2.1 depending on the type of interpolation operator – mid-point or integral mid-value. A comparative analysis of the achieved performance for both MP and MV algorithms and the real distribution of computation and communication times is made in [3]. It is experimentally shown there that: 1) the cpu-times for MV are larger than those for MP because of the larger number of iterations; the speed-up and efficiency coefficients are practically 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; 2) the computation time decreases almost twice when the number of processors is increased by two; 3) the communication time does not depend on the number of processors; 4) the communications for the system with the preconditioner is the most time consuming part.

The speed-up and efficiency coefficients obtained on Thea, Simba and Lilli for the parallel algorithm corresponding to MV are collected in Table 1 (we do not report the results related to MP, because of 1) above). Table 1 has the following format. The number of processors N_p is given in the first column. There are two numbers in each box of the second column – the number n of the elements on each coordinate direction and the related number of iterations for that size of the discrete problem. The rest of the columns are in three groups with similar structure – one per each parallel machine. Each of these groups has three fields. The measured cpu-time in seconds is given in the first field. The rest two of them present the speed-up $S_{N_p} = \dfrac{T_1}{T_{N_p}}$ and the efficiency $E_{N_p} = \dfrac{S_{N_p}}{N_p}$. The cpu-times T_{N_p} are the best obtained from ten measurements.

We observe that: 1) for a given number of processors the speed-up and respectively efficiency coefficients grow up for larger size of the problem; 2) the speed-up and the efficiency are the highest on Simba (with an exception n=2048, when they are achieved on Lilli), while those on Thea are far away from the theoretical upper bounds $S_{N_p} \leq N_p$, $E_{N_p} \leq 1$; 3) the total execution time on Lilli for $N_p > 1$ is the smallest one.

The reason for this behaviour is hidden in the system parameters t_s, t_w, t_a and the related architecture. Comparing T_1 for all the machines we observe that the computation times on Simba are the largest. The computation times on Lilli are slightly higher than those on Thea and almost twice lower than those on Simba. At the same time communications on Thea are much slower than those on Simba and Lilli since for two processors the total time on Thea is bigger than on the other two machines.

Table 1: No overlap of communications with computations – Algorithm MV

		Thea			Simba			Lilli		
N_p	$\frac{n}{iter}$	cpu	S_{N_p}	E_{N_p}	cpu	S_{N_p}	E_{N_p}	cpu	S_{N_p}	E_{N_p}
1		10.43			18.65			11.07		
2	256	10.89	0.96	0.48	8.97	2.08	1.04	5.87	1.89	0.95
4	81	13.10	0.80	0.20	3.73	5.00	1.25	3.35	3.30	0.83
8		12.70	0.82	0.10	2.83	6.59	0.82	2.30	4.81	0.60
16					3.37	5.53	0.35	1.43	7.74	0.48
32								1.55	7.14	0.22
1		61.70			123.59			63.54		
2	512	47.96	1.29	0.65	61.78	2.00	1.00	33.49	1.90	0.95
4	119	47.27	1.31	0.33	33.84	3.65	0.91	18.17	3.50	0.87
8		41.31	1.49	0.19	17.35	7.12	0.89	11.00	5.78	0.72
16					11.12	11.11	0.69	7.12	8.92	0.56
32								5.65	11.25	0.35
1		323.42			729.48			351.61		
2	1024	239.60	1.35	0.68	366.47	1.99	1.00	186.93	1.88	0.94
4	167	175.52	1.84	0.46	188.57	3.87	0.97	98.87	3.56	0.89
8		140.60	2.30	0.29	97.74	7.46	0.93	53.51	6.57	0.82
16					58.47	12.48	0.79	31.94	11.01	0.69
32								22.79	15.43	0.48
1					4274.98			2465.15		
2	2048				2153.53	1.99	0.99	1014.94	2.43	1.21
4	240				1119.91	3.82	0.95	535.33	4.60	1.15
8					570.16	7.50	0.94	281.53	8.76	1.09
16					311.40	13.73	0.86	160.88	15.32	0.96
32								90.79	27.15	0.85

Our goal is to obtain a better scalable preconditioner. What can we do to improve the performance on machines like Thea? The next section is devoted to this question.

4 How to improve the performance?

First of all we have to find the reason for such a behaviour. From the previous section and from the comparative analysis in [3] we conclude that the communications for the system with preconditioner is the most time consuming part. Here, we present another experiment which confirms this fact. After that we suggest a proper reordering of the communications and computations, which allows usage of non-blocking send and receive operations of MPI and overlapping of communications and computations. Next we derive theoretical estimates for the total execution time of the modified algorithm. At the end of this section we present numerical results on the same machines Thea, Simba and Lilli, and compare the performance with the old version of the code.

4.1 No communications case

Let us see what will happen if no communications are performed in the preconditioner? This in fact means that we cut the links between the strips in the neighbouring processors for matrix $\mathcal{C}$, or which is the same – instead of transferring the corresponding components we vanish them. The resulting algorithm can be interpreted as a combination of block Jacoby and MIC(0) factorization of the related blocks. The number of iterations and the measured cpu-time for such an experiment are shown in Table 2. The tests are performed

Table 2: No communications in the preconditioner – Algorithm MV

		Thea		Simba		Lilli	
n	N_p	iter	cpu	iter	cpu	iter	cpu
256	1	81	10.31	81	18.72	81	11.09
	2	243	16.92	244	24.70	244	16.64
	4	295	10.84	288	9.87	289	9.78
	8	387	8.14	388	13.93	390	6.66
512	1	119	63.25	119	124.18	119	63.49
	2	418	114.59	439	220.78	426	119.86
	4	533	74.60	529	130.68	534	74.01
	8	661	48.74	661	88.11	661	49.63
1024	1	167	339.16	167	732.97	167	351.49
	2	680	793.64	666	1420.79	673	742.54
	4	730	387.68	733	785.24	728	779.37
	8	930	253.72	919	637.29	932	270.97

on the same machines Thea, Simba and Lilli with the test example from Section 3.3.

The number of iterations for a given value of n depends on the number of processors – the larger N_p causes an increase of the number of iterations. Although the iterations for 8 processors for each n are almost 5 times more than those for 1 processor, the cpu-time on all the machines is smaller. Furthermore, for $n = 256$ and $N_p = 4$, 8 the cpu-times on Thea are smaller than those presented in Table 1 for the same values of n and N_p. These results confirm ones more that the communications in preconditioner are the reason why the speed-up on Thea (i.e. on a distributed memory machine) is relatively far away from its upper bound. They do not have so strong influence on the performance on the other two machines with non-uniform shared memory architecture.

4.2 Overlapping of computations and communications (OCC)

How to reduce the influence of the communication time of the original MIC(0) algorithm for distributed memory machines with large ratios $\frac{t_s}{t_a}$ and $\frac{t_s}{t_w}$? Let us focus our attention on the system with preconditioner. We try to modify only this part of the solver, since the negative impact of the matrix-vector multiplication and of the inner products on the performance is not very strong.

To apply the preconditioner $\mathcal{C}_{\mathrm{MIC}(0)}(B)\mathbf{w} \equiv (X-\widetilde{L})X^{-1}(X-\widetilde{L})^t\mathbf{w} = \mathbf{v}$ (see (5), (6)), 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 $\mathbf{z} := X\mathbf{y}$; and 3) find $\mathbf{w}$ from $L^t\mathbf{w} = \mathbf{z}$. Data transfers between neighbouring processors (see below and [3] for more details) are needed at steps 1) and 3), while no communication is required at step 2). Let us remind the important advantage of the

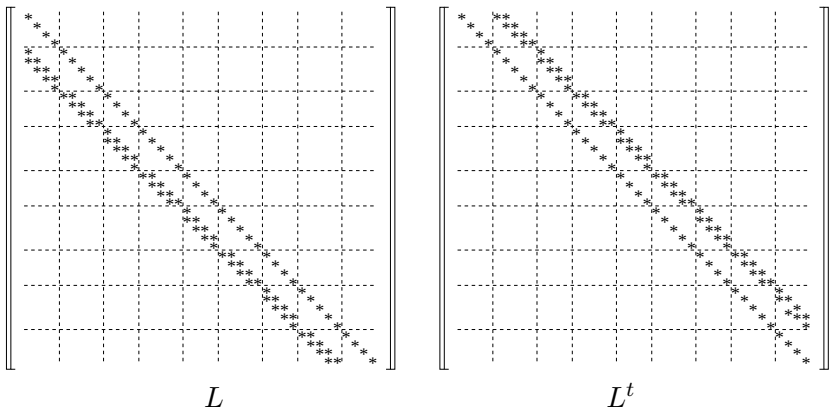


Figure 7: Nonzero structure of the triangular factors L and L^t of the preconditioner $\mathcal{C}_{\mathrm{MIC}(0)}(B)$ for A.

matrix B, that all of its diagonal blocks are diagonal and hence the same holds for L (see Figure 7). The nonzero structure of two successive block rows ($2j$ and $2j+1$, $j > 0$)

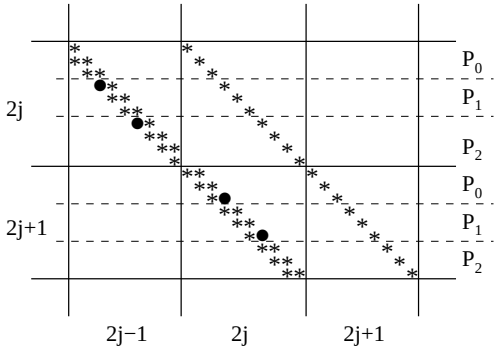


Figure 8: The scheme of communications and computations for the solution of $L\mathbf{y} = \mathbf{v}$. Note that $N_p = 3$, $n_1 = n_2 = 9$.

of L and their distribution among the processors is presented in Figure 8. The system $L\mathbf{y} = \mathbf{v}$ is solved by a standard forward recurrence. The block L_{11} is diagonal, divided into N_p strips – one per each processor. So $L_{11}\mathbf{y}_1 = \mathbf{v}_1$ is solved in parallel without any data transfers. Then we have to determine $\mathbf{y}_2$ from $L_{21}\mathbf{y}_1 + L_{22}\mathbf{y}_2 = \mathbf{v}_2$ (i.e. block row $2j$ for $j = 1$, Figure 8). 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 of $\mathbf{y}_1$ to P_{i+1} and to receive another one from P_{i-1}. These data are multiplied by entries denoted by $\bullet$ in Figure 8 (row $2j$), i.e. they participate only

in the computation of the first element of $\widetilde{\mathbf{v}}_2 = \mathbf{v}_2 - L_{21}\mathbf{y}_1$ in each P_i. So each P_i can calculate the first entry of $\widetilde{\mathbf{v}}_2$ at the end allowing in such a way the communication to be overlapped with the computations of the rest components. It is implemented in the MPI code using non-blocking send and receive operations. Then $L_{22}\mathbf{y}_2 = \mathbf{v}_2 - L_{21}\mathbf{y}_1$ is handled concurrently without any other communications. After that, $L_{32}\mathbf{y}_2 + L_{33}\mathbf{y}_3 = \mathbf{v}_3$ (i.e. block equation $2j+1$ for $j=1$) has to be solved. Again, one component of $\mathbf{y}_2$ is transferred, but now to processor P_{i-1}. It participates in the last element of $\widetilde{\mathbf{v}}_3$ in each P_i and it is determined after the communication, overlapped with computation of the remaining entries, is completed. The procedure continues till the last block of $\mathbf{y}$ is computed. The general rule is that we reorder computations only in the even blocks $2j$, namely we postpone the computations with the first row till all the computations overlapped with a communication are completed. For the odd blocks $2j+1$ no reordering is required – the last row is handled after the rest computations and the communication are performed. The third step is handled in a similar way as the first step, applying the backward recurrence.

4.3 Parallel complexity for the case of OCC

We now evaluate the execution time for the proposed modification of the algorithm again per iteration. We will not suppose any more that the total execution time is a sum of the computation time T_a and the communication time T_{com} as it was done in Section 3.2, but we still have the assumptions b) and c) from its beginning. We remind that for an $n_1 \times n_2$ mesh of nonconforming finite elements, the degrees of freedom are $N = n_1(2n_2+1)+n_2$.

We present the total execution time as a sum of the times required to perform the separate steps of a single PCG iteration using N_p processors:

$$T_{it}(N,N_p) = T_{it}^{precon}(N,N_p) + T_{it}^{mv}(N,N_p) + T_{it}^{inpr}(N,N_p) + T_{it}^{lvt}(N,N_p)\,. \qquad (8)$$

The terms in this sum stand for the required time in one solution of a system with $\mathcal{C}_{N\times N}$ ($T_{it}^{precon}(N,N_p)$); one matrix vector multiplication with $A_{N\times N}$ ($T_{it}^{mv}(N,N_p)$); two inner products ($T_{it}^{inpr}(N,N_p)$) and three linked vector triads $\mathbf{v} := \alpha\mathbf{v} + \mathbf{u}$ ($T_{it}^{lvt}(N,N_p)$).

We have $T_{it}^{lvt}(N,N_p) = \dfrac{6N}{N_p}\cdot t_a$ since there are no communications in the linked vector triads. For the inner products we have to perform $\dfrac{4N}{N_p}$ a. o. For each of them a global communication consisting of one all_to_one reduce for a single number and one one_to_all broadcast for a single number is required. Both parts take one and the same time denoted by $b(N_p,1)$ (see for some more details [11]). Therefore $T_{it}^{inpr}(N,N_p) = \dfrac{4N}{N_p}\cdot t_a + 4b(N_p,1)$. The value of $b(N_p,1)$ depends on the architecture and for the ring, 2D mesh and hypercube (upper indices r, m, h respectively) it is:

$$\begin{aligned} b^r(N_p,1) &= (t_s+t_w)\left\lceil \frac{N_p}{2}\right\rceil, \\ b^m(N_p,1) &= 2(t_s+t_w)\left\lceil \frac{\sqrt{N_p}}{2}\right\rceil, \\ b^h(N_p,1) &= (t_s+t_w)\log N_p\,. \end{aligned}$$

One matrix vector multiplication with $A_{N\times N}$ is done by $\approx \frac{13N}{N_p}$ a. o. The communications for $A\mathbf{v}$ and $\mathcal{C}^{-1}\mathbf{v}$ are between processors with successive indices. They will be local if these processors are physically neighbours, i.e. $l = 1$. The communication time for $A\mathbf{v}$ is estimated by $T_{com}(A\mathbf{v}) = 2t_s + (3n_1 + 1)t_w$, when $N_p = 2$ and $T_{com}(A\mathbf{v}) = 4t_s + 2(3n_1 + 1)t_w$, when $N_p > 2$ (see [3] for more details). Since there is no overlap between communications and computations we obtain that

$$T_{it}^{mv}(N, N_p) \approx \frac{13N}{N_p} \cdot t_a + \begin{cases} 2t_s + (3n_1 + 1) \cdot t_w \,, & \text{for } N_p = 2 \,, \\ 4t_s + 2(3n_1 + 1) \cdot t_w \,, & \text{for } N_p > 2 \,. \end{cases} \tag{9}$$

One solution with the preconditioner $\mathcal{C}$ consists of two types of steps. In the first one, communications are overlapped with part of the computations and we denote the required time with $T_{it}^{overl}(N, N_p)$. In the second type only arithmetic operations are performed for a total time $T_{it}^{adop}(N, N_p)$. We represent the execution time as their sum $T_{it}^{precon}(N, N_p) = T_{it}^{overl}(N, N_p) + T_{it}^{adop}(N, N_p)$. Steps 1) and 3) from the beginning of Section 4.2 contribute to both parts $T_{it}^{overl}(N, N_p)$ and $T_{it}^{adop}(N, N_p)$, while the second step participates with $\frac{N}{N_p} \cdot t_a$ only in $T_{it}^{adop}(N, N_p)$. Each of L and L^t from the representation $\mathcal{C} = LX^{-1}L^t$ with $L = X - \widetilde{L}$ (see (5), (6)) has $2n_1 + 1$ blocks of order n_2 (or n_2+1). Hence we have to solve $2n_1+1$ diagonal systems with the blocks L_{ii}, which do not require any communications. It is done by $(2n_1+1)\frac{n_2}{N_p}$ divisions added in $T_{it}^{adop}(N, N_p)$. To compute the right-hand side of $L_{ii}\mathbf{y}_i = \mathbf{v}_i - L_{i,i-1}\mathbf{y}_{i-1}$ for $2n_1$ of these systems we have to overlap one_to_one communication for one number with $4(\frac{n_2+1}{N_p} - 1)$ a. o. This step takes time $max(2(t_s + t_w), 4(\frac{n_2+1}{N_p} - 1) \cdot t_a)$ for a single block of L and contributes to $T_{it}^{overl}(N, N_p)$. There are additional 4 a. o. for each block which are not overlapped with a communication and they contribute in $T_{it}^{adop}(N, N_p)$. Taking into account that to solve the system with each of L and L^t we have $2n_1$ blocks with communications, we obtain

$$T_{it}^{overl}(N, N_p) = 4\, n_1 \text{ max } (2(t_s + t_w), 4(\frac{n_2 + 1}{N_p} - 1) \cdot t_a).$$

For $T_{it}^{adop}(N, N_p)$ we have

$$\begin{aligned} T_{it}^{adop}(N, N_p) &= \frac{N}{N_p} \cdot t_a + 16n_1 \cdot t_a + 2(2n_1 + 1)\frac{n_2}{N_p} \cdot t_a & (10) \\ &\approx (3\frac{N}{N_p} + 16n_1) \cdot t_a \,. & (11) \end{aligned}$$

From here we obtain

$$T_{it}^{precon}(N, N_p) \approx (3\frac{N}{N_p} + 16n_1) \cdot t_a + 4\, n_1\, max(2(t_s + t_w), 4(\frac{n_2 + 1}{N_p} - 1) \cdot t_a) \,.$$

The total execution time for a PCG iteration of the modified algorithm for $N_p > 2$ is

$$
\begin{aligned}
T_{it}(N, N_p) &= (3\frac{N}{N_p} + 16n_1) \cdot t_a + 4\, n_1\, max(2(t_s + t_w), 4(\frac{n_2+1}{N_p} - 1) \cdot t_a) && (12)\\
&+ \frac{4N}{N_p} \cdot t_a + 4b(N_p, 1) + \frac{6N}{N_p} \cdot t_a && (13)\\
&+ \frac{13N}{N_p} \cdot t_a + 4t_s + 2(3n_1 + 1) \cdot t_w && (14)\\
&\approx (26\frac{N}{N_p} + 16n_1) \cdot t_a + 4b(N_p, 1) + 4t_s + 2(3n_1 + 1) \cdot t_w && (15)\\
&+ 4\, n_1\, max(2(t_s + t_w), 4(\frac{n_2+1}{N_p} - 1) \cdot t_a)\,. && (16)
\end{aligned}
$$

When $N_p = 2$ we have

$$
\begin{aligned}
T_{it}(N, N_p) &= (3\frac{N}{N_p} + 16n_1) \cdot t_a + 4\, n_1\, max((t_s + t_w), 4(\frac{n_2+1}{N_p} - 1) \cdot t_a) && (17)\\
&+ \frac{4N}{N_p} \cdot t_a + 4b(N_p, 1) + \frac{6N}{N_p} \cdot t_a && (18)\\
&+ \frac{13N}{N_p} \cdot t_a + 2t_s + (3n_1 + 1) \cdot t_w && (19)\\
&\approx (26\frac{N}{N_p} + 16n_1) \cdot t_a + 4b(N_p, 1) + 2t_s + (3n_1 + 1) \cdot t_w && (20)\\
&+ 4\, n_1\, max((t_s + t_w), 4(\frac{n_2+1}{N_p} - 1) \cdot t_a)\,. && (21)
\end{aligned}
$$

4.4 Numerical tests II

We present in Table 3 results for the case with overlapping of communications with computations. The test problem and the machines are the same as in Section 3.3. Table 3 has the same format as Table 1.

Again, for a given number of processors, the speed-up and the related efficiency grow up for larger size of the problem. On Thea cluster there is a significant improvement of S_{N_p} and E_{N_p} compared to the case without overlapping. The values of S_{N_p} and E_{N_p} on Lilli have also increased but not as much as on Thea and they are still smaller in general than those on Simba.

In contrast, the efficiency coefficients with and without overlapping on Simba are close to each other. They are larger in general than the highest ones obtained on Thea and Lilli (except the case n=2048 for Lilli). But on the other hand, the total execution time on Simba is the largest one. The superlinear speed-ups on Simba and Lilli are due to the type and the size of the cache memory. The performance is also influenced by the nonuniform memory access.

As expected, the results confirm that the performance of our code strongly depends on the system's parameters. The influence of communications is stronger on distributed memory machines, but it is not the only factor that affects the performance. The other

Table 3: Overlap of communications and computations – Algorithm MV

		Thea			Simba			Lilli		
N_p	$\frac{n}{iter}$	cpu	S_{N_p}	E_{N_p}	cpu	S_{N_p}	E_{N_p}	cpu	S_{N_p}	E_{N_p}
1		10.47			17.79			11.05		
2	256	9.23	1.13	0.57	8.39	2.12	1.06	5.80	1.91	0.96
4	81	8.42	1.24	0.31	3.64	4.89	1.22	3.17	3.49	0.87
8		7.73	1.35	0.17	2.82	6.31	0.79	1.95	5.67	0.71
16					3.30	5.39	0.34	1.32	8.37	0.52
32								1.12	9.87	0.31
1		61.88			118.19			63.50		
2	512	38.77	1.60	0.80	58.35	2.03	1.01	33.39	1.90	0.95
4	119	28.65	2.16	0.54	31.86	3.71	0.93	17.73	3.58	0.90
8		23.61	2.62	0.33	16.83	7.02	0.88	10.03	6.33	0.79
16					10.89	10.85	0.68	6.15	10.33	0.65
32								4.16	15.26	0.48
1		323.77			701.05			351.08		
2	1024	216.89	1.49	0.75	349.42	2.01	1.00	182.62	1.92	0.96
4	167	121.26	2.67	0.67	184.58	3.80	0.95	94.07	3.73	0.93
8		80.06	4.04	0.51	96.55	7.26	0.91	51.22	6.85	0.86
16					57.60	12.17	0.76	28.92	12.14	0.76
32								19.20	18.29	0.57
1					4157.97			2444.07		
2	2048				2051.36	2.03	1.02	1039.82	2.35	1.18
4	240				1041.95	3.99	1.00	558.73	4.37	1.09
8					547.90	7.59	0.95	273.51	8.94	1.12
16					299.22	13.90	0.87	147.22	16.60	1.04
32								85.46	28.60	0.89

machine characteristics like speed for computations and memory access, type and size of cache memory, can lead to speed-ups close to the theoretical upper bounds with total execution times much bigger than on machines with smaller speed-ups.

5 Concluding remarks

The recently introduced scalable parallel MIC(0) preconditioner is further developed in this paper. The performance for distributed memory machines is improved by overlapping of communications and computations. Theoretical estimates of the parallel execution times for overlapping of each local communication with computations are derived. The execution times, together with the achieved speed-ups and efficiency coefficients on three parallel architectures, are compared.

Our future plans include the generalization of the considered algorithm into 3D case and its implementation for linear elasticity problems.

Acknowledgements

This research has been supported in part by the Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) under the project p2004009 ,,Parallel computing in Geosciences“ and by the bilateral IG AS – IPP BAS interacademy exchange grant ,,Reliable Modelling and Large Scale Computing in Geosciences“. Thanks to these projects and the fact that G. Bencheva started a PostDoc in RICAM, Linz, we were able to study the differences in the performance of our parallel preconditioner on three different computer architectures.

The scientific visit of G. Bencheva to Uppsala was possible due to the sponsorship of the Royal Swedish Academy of Engineering Sciences, IVA. S. Margenov is also supported by the Bulgarian NSF grant I1402/04.

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] G. Bencheva and S. Margenov, Parallel incomplete factorization preconditioning of rotated linear FEM systems, *J. Comp. Appl. Mech.,* **4(2)** (2003) 105–117.

[3] G. Bencheva, S. Margenov, Performance analysis of a parallel MIC(0) preconditioning of rotated bilinear nonconforming FEM systems, *Mathematica Balkanica,* **17** (2003) 319–335.

[4] G. Bencheva, Parallel algorithms for separation of variables and sparse matrices factorization, PhD Theses (in Bulgarian), http://parallel.bas.bg/∼gery/theses.html (2004).

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

[6] I. Gustafsson, Stability and rate of convergence of modified incomplete Cholesky factorization methods, Report 79.02R. Dept. of Comp. Sci., Chalmers University of Technology, Goteborg, Sweden, (1979).

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

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

[9] 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.

[10] P. Hansbo and M.G. Larson, A simple nonconforming bilinear element for the elasticity problem, *Trends in Computational Structural Mechanics*, W.A. Wall, K.U. Bletzinger, and K. Schweizerkopf (Eds.), CIMNE, (2001) 317–327.

[11] Kumar, V., Grama, A., Gupta, A., Karypis, G., *Introduction to parallel computing: design and analysis of algorithms,* (Benjamin-Cummings Addison-Wesley Publishing Company,Inc., 1994).

[12] 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.

[13] R. Rannacher and S. Turek, Simple nonconforming quadrilateral Stokes Element, *Numer. Methods for PDEs* **8**(2) (1992) 97–112.

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

[15] Y. Saad, *Iterative methods for sparse linear systems*, (PWS Publishing Company, Boston, 1996).