<u>TECHNICAL REPORT HG CP 94–00820–CICT–6.1</u>

Parallel iterative solution methods for elasticity problems

Krassimir Georgiev Ivan Lirkov Svetozar Margenov
BAS, Sofia, Bulgaria

Abstract

This report follows the research plan of the HIPERGEOS Project COP–94 00820. We analyze the parallel complexity of the preconditioning for the numerical solution of large scale elasticity systems where domain decomposition or displacement decomposition preconditioners are used.

1 Introduction

In this report we are concerned with the numerical solution of 3D elasticity systems. After discretization, such problems are reduced to find the solution of linear systems of the form $K\mathbf{x} = \mathbf{F}$ where K is sparse symmetric positive definite matrix. Such linear systems are often solved by the Preconditioned Conjugate Gradient method.

We mention the possibilities of determining effective parallel algorithms and implementation on modern high-speed computers mainly parallel computers with shared and distributed memory including here clusters of workstations. The communication cost in a network of workstations can be modeled as the distributed-memory parallel computers considered above. The only difference is that the network parameters *start-up time* and *time for transferring of single word* are larger.

One of the most popular and the most successful class of preconditioners is the class of incomplete LU (ILU) factorizations. One potential problem with the ILU preconditioners is that they have limited degree of parallelism. Some attempts to modify the method and to devise more parallel variants often result in a deterioration of the convergence rate.

In recent years there has been a considerable development in domain decomposition (DD) methods for numerical approximation of partial differential equations in the framework of finite differences, finite elements and spectral methods.

In [4], R. Chan and T. F. Chan proposed another class of preconditioners which is based on averaging coefficients of A to form a block-circulant approximation (see also [12, 13]). The block-circulant preconditioners are highly parallelizable but they are substantially sensitive with respect to a possible high variation of the coefficients of the elliptic operator.

The sensitivity of the block-circulant approximations with respect to a high variation of the problem coefficients was relaxed in the circulant block factorization (CBF) preconditioners ([16]).

We consider preconditioners for solving a 3D linear elasticity problem on a parallel computer or on a cluster of workstations working under MPI (Message Passing Interface).

2 Elasticity problem

Let $\mathcal{B}$ be an elastic body occupying bounded polyhedral domain $\Omega \in \mathbb{R}^3$, impose Dirichlet/Neumann boundary conditions on $\Gamma \equiv \partial\Omega$. We denote by $\mathbf{u} = [u_1, u_2, u_3]^T$ the displacement, by $\mathbf{F} = [f_1, f_2, f_3]^T$ the vector of the body forces, by $\underline{\sigma} = (\sigma_{ij})(\mathbf{u})$ the stress tensor and by $\underline{\epsilon}(\mathbf{u}) = (\epsilon_{ij}(\mathbf{u}))$ the strain tensor (the deformation associated with $\mathbf{u}$).

The variational formulation of the problem for obtaining the displacements of the above determined elastic body is:

Find: $\mathbf{u} \in (H_0^1(\Omega))^3$, such that

$$a_\Omega(\mathbf{u}, \mathbf{v}) \equiv \int_\Omega (\lambda \; div\mathbf{u} \; div\mathbf{v} \; + \; \mu \sum_{i,j=1}^{3} \epsilon_{ij}(\mathbf{u}) \; \epsilon_{ij}(\mathbf{v}) \; dx = F(\mathbf{v}), \qquad \forall \mathbf{v} \in (H_0^1(\Omega))^3,$$

where

$$F(\mathbf{u}) = \sum_{i=1}^{3} \int_\Omega f_i u_i dx,$$

and μ is a constant depends on the modulus of elasticity (Young's modulus) - E and on the contraction ratio (Poisson ratio) of the elastic material of the body $\mathcal{B}$ - ν.

The bilinear form $a_\Omega(\mathbf{u}, \mathbf{v})$ is a second order, symmetric and coercive bilinear form.

The computational domain is the parallelepiped

$$\Omega \; = \; [0, x_1^{max}] \times [0, x_2^{max}] \times [0, x_3^{max}].$$

Single pile in a homogeneous sandy clay soil layer is considered. An uniformly load distribution on the cross section of the pile is assumed.

Boundary conditions of Dirichlet and Neumann type on the different parts of the boundary of the domain are given as follows:

- Homogeneous Dirichlet boundary conditions on the *bottom side* of the parallelepiped $x_3 = 0$, i.e the displacements are equal to zero;

- Homogeneous Neumann boundary conditions on the *vertical sides* of the parallelepiped
$$x_1 = 0, \quad x_1 = x_1^{max}, \quad x_2 = 0, \quad x_2 = x_2^{max},$$
i.e the stresses are equal to zero;

- Nonhomogenuous Neumann boundary conditions on the *top side* of the parallelepiped $x_3 = x_3^{max}$, where the load from the upper construction is applied.

Now, let $V^h(\Omega) \subset (H_0^1(\Omega))^3$ be the space of continuous piecewise linear functions, on the triangulation, which vanish on the boundary $\partial\Omega$. Then, the discrete variational problem is:

Find: $u_h \in V^h(\Omega)$, such that
$$a_\Omega(u_h, v_h) = F(v_h), \quad \forall\, v_h \in V^h(\Omega).$$

We will use the standard nodal basis functions Φ_i for the space V_h. If we express the solution
$$u_h \;=\; \sum x_i\, \Phi_i,$$
we will obtain the linear system of equations
$$K\mathbf{x} \;=\; \mathbf{F}, \tag{1}$$
where $\mathbf{x} = [x_i]^T$ is the vector of unknowns — x_i, $i = 1, 2, \ldots, n$ and n is the number of all nodes in Ω, $\mathbf{F}$ — the right-hand side is the vector of components $F_i = \mathbf{F}(\Phi_i)$, and K is the stiffness matrix with entries $K_{ij} = a_\Omega(\Phi_i, \Phi_j)$.

To solve the linear system of equations (1) the following preconditioned conjugate gradient procedure is used [1].

PCG — procedure:

- $M\mathbf{U}_0 \;=\; \mathbf{F}$ – *initial iterate*;
- $\mathbf{r}_0 \;=\; \mathbf{F} \;-\; K\mathbf{U}_0$ — *initial defect*;

for $\quad k = 0, 1, \ldots \quad$ *until convergence*

- $M\mathbf{z}_k \;=\; \mathbf{r}_k$;
- $\beta_k \;=\; \dfrac{(\mathbf{r}_k, \mathbf{z}_k)}{(\mathbf{r}_{k-1}, \mathbf{z}_{k-1})}, \qquad \beta_0 \;=\; 0$;
- $\mathbf{p}_k \;=\; \mathbf{z}_k \;+\; \beta_k\mathbf{p}_{k-1}$ — *next search direction*;
- $\alpha_k \;=\; \dfrac{(\mathbf{r}_k, \mathbf{z}_k)}{(K\mathbf{p}_k, \mathbf{p}_k)}$;
- $\mathbf{U}_{k+1} \;=\; \mathbf{U}_k \;+\; \alpha_k\mathbf{p}_k$ — *next iterate;;*
- $\mathbf{r}_{k+1} \;=\; \mathbf{r}_k \;-\; \alpha_k K\mathbf{p}_k$ — *next quasi-residual*;

3 MPI

MPI is standard message passing interface for applications and libraries running on concurrent computers with a logically distributed memory [21]. It is used successively on massively parallel computers and clusters of workstations. The main advantages of MPI are portability and ease–of–use, and that MPI is a key component in the development of concurrent computing environment in which applications and tools can be transparently ported between different computers. Some of the main futures of MPI are: (*i*) a set of routines that support point–to–point communication between pairs of processors in blocking and nonblocking versions; (*ii*) a communicator abstraction that provides support for the design of modular parallel software libraries; (*iii*) application topologies specifying the logical layout of the processes; (*iv*) a rich set of collective communication routines performing coordinated communications among a set of processes. Each process is assigned a unique rank in the range 0 to $n-1$, where n is the number of the processes. The rank is used to identify a process. In a point–to–point communication operation it is used to specify the source and the destination processes while in certain collective communication operations it is used to specify the root process.

An MPI program consists of autonomous processes, executing their own code, in an MIMD style. The codes executed by each process need not be identical. The processes communicate via calls to MPI communication primitives.

Sending and receiving of messages by processes is the basic MPI communication mechanism. The basic point–to–point communication operations are **send** and **receive**, in *blocking* (if return from the procedure indicates the user is allowed to re–use resources specified in the call) and *nonblocking* (if the procedure may return before the operation completes, and before the user is allowed to re–use resources, such as buffers, specified in the call) versions.

Here, we will present a short description of the operations used in the now existing version of the computer code realizing DD–BSR-BILU preconditioner for the 3D linear elasticity problem on a parallel computer or on a cluster of workstations. As one can improve performance on many systems by overlapping communications and computations, in this version of our computer code we have used **nonblocking communications** as well as some **collective communications**. The nonblocking communications use opaque **request** objects to identify communication operations and match the operation that initiates the communication with the operation that terminates it. These are system objects that are accessed via a handle.

Communication initiation

MPI_ISEND(buf, count, datatype, dest, tag, comm, request)

buf	initial address of send buffer (choice)
count	number of elements in send buffer (integer)
datatype	datatype of each send buffer element (handle)
dest	rank of destination (integer)
tag	message tag (integer)
comm	communicator (handle)
request	communication request (handle)

MPI_IRECV(buf, count, datatype, source, tag, comm, request)

buf	initial address of receive buffer (choice)
count	number of elements in receive buffer (integer)
datatype	datatype of each receive buffer element (handle)
source	rank of source (integer)
tag	message tag (integer)
comm	communicator (handle)
request	communication request (handle)

These calls allocate a communication request object and associate it with the request handle (the argument request). The request can be used later to query the status of the communication or wait for its completion.

Communication completion

The functions MPI_WAIT and MPI_TEST are used to complete a nonblocking communication. The completion of a send operation indicates that the sender is now free to update the locations in the send buffer while the completion of a receive operation indicates that the receive buffer contains the received message and the receiver is now free to access it. It is often more convenient to be able to wait for the completion of any, some, or all the operations in a list. A call to MPI_WAITALL can be used to wait for all pending operations in a list.

MPI_WAITALL(count, array_of_requests, array_of_statuses)

count	lists length (integer)
array_of_requests	array of requests (array of handles)
array_of_statuses	array of status objects (array of status)

Collective communication

MPI includes reduce operations where the result is returned to all processes in the group. MPI requires that all processes involving in these operations receive identical results.

MPI_ALLREDUCE(sendbuf, recvbuf, count, datatype, op, comm)

sendbuf	starting address of send buffer (choice)
recvbuf	starting address of receive buffer (choice)
count	number of elements in send buffer (integer)
datatype	datatype of elements of send buffer (handle)
op	operation (handle)
comm	communicator (handle)

4 Domain decomposition preconditioning

The large penalties for interprocessor communication on networks of workstations represent an extreme case of the two-level memory structure for which domain decomposition is so suited. There are two main possibilities to use workstations for realization of the overlapped domain decomposition algorithms: to use a single workstation and exploit a virtual memory or to use a network of workstations. Why not use a single workstation? The answer is that the page thrashing reduces the effective computation rate. Knowledge of this effect and effects related to the cache memory is important; it explains so called supper-linear speed-up that is some times observed.

4.1 Statement of the problem

DD methods are based on a decomposition of the spatial domain of the problem into several subdomains. It is motivated by the need of solvers which are easily parallelized on coarse grain parallel computers. Sometimes DD algorithms can reduce the complexity of the solvers on sequential computers too.

Most DD methods can be classified as either an overlapping or a non-overlapping subdomain approach.

Now, we will stop at Schwartz algorithms based on many overlapping subdomains [6].

The triangulation of Ω is done in the following way. The domain Ω is first divided into non-overlapping substructures $\Omega_i, i = 1, 2, ..., N$ with diameters of the order of H. All the substructures Ω_i are further divided into elements (tetrahedral or parallelepipedal) with characteristic diameters of order of h.

We extend each sub-domain Ω_i to a larger region $\widehat{\Omega}_i$. We assume that the overlap is generous, assuming that the distance between the boundaries $\partial\Omega_i$ and $\partial\widehat{\Omega}_i$ is bounded from below from a fixed fraction of H_i, the diameter of Ω_i.

We also assume that $\partial\widehat{\Omega}_i$ does **not cut** through any element. We make the same construction for the substructures that are next to the boundary of the domain Ω except that we cut off the part of $\widehat{\Omega}_i$ that is outside of Ω.

Now, let $V^H(\Omega) \subset (H_0^1(\Omega))^3$ be the space of continuous piecewise linear functions, on the triangulation, which vanish on the boundary $\partial\Omega$.

Let $\widehat{K_i}$ be a local stiffness matrix according to the sub-domain $\widehat{\Omega}_i$. We define restriction maps R_i and extension maps R_i^T, as follows. Let the global stiffness matrix K is of order n and denote with $\widehat{n}_i$ the number of the interior with respect to $\widehat{\Omega}_i$, nodes. For each sub-domain $\widehat{\Omega}_i$ the matrix R_i is $(n \times \widehat{n}_i)$ restriction matrix with entries 1's and 0's , taking into account the indexes of the nodes belong to $\widehat{\Omega}_i$. This matrix restricts a vector $\mathbf{x}$ of length n to $R_i\mathbf{x}$ of length $\widehat{n}_i$. Thus, the local stiffness matrix $\widehat{K_i}$ corresponding to the sub-domain $\widehat{\Omega}_i$ is

$$\widehat{K_i} \; = \; R_i \, K \, R_i^T .$$

In the case of many subdomains, the additive Schwartz algorithm is a generalization of the well known two-sub-domain case with preconditioner

$$M_{add}^{-1} = \sum_{i=1}^{N} R_i^T \ K_i^{-1} \ R_i.$$

Indeed, if we start with an iterate (k), we compute $(k+1)$ by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \left(\sum_{i=1}^{N} R_i^T \ \widehat{K_i^{-1}} \ R_i \right) \left(\mathbf{F} - K \ \mathbf{x}^{(k)} \right)$$

As $R_i^T \ \widehat{K_i^{-1}} \ R_i \ \mathbf{y}$ can be computed in parallel for the different subdomains, this leads to a coarse grain parallelism. Taking into account the definition of the restriction matrix R_i (and the extension or interpolation matrix R_i^T) we may conclude that it is not necessary to store these matrices.

So defined preconditioner M_{add} is a straightforward generalization of the standard block-Jacobi preconditioner. The convergence rate of this preconditioner deteriorates when the number of subdomains increases.

4.2 The Solver

The algorithm for solving the system $M\mathbf{z}_k = \mathbf{r}_k$ with $M = M_{add}$ is as follows.

- $\widehat{\mathbf{r}}_k^i$ – vector of a length $\widehat{n}_i$; entries are equal to those of $\mathbf{r}_k$ for the components corresponding to the meshpoints in $\widehat{\Omega}_i$ and 0 elsewhere;
- $\widehat{\mathbf{z}}_k^i \ : \ \widehat{K}_i \widehat{\mathbf{z}}_k^i = \widehat{\mathbf{r}}_k^i;$
- $\mathbf{z}_k^i$ – vector of a length n; an extension of $\widehat{\mathbf{z}}_k^i$ with 0 entries outside $\widehat{\Omega}_i$;
- $\mathbf{z}_k = \sum_{i=1}^{N} \mathbf{z}_k^i$

The coefficient matrices of the systems of linear algebraic equations

$$\widehat{K}_i \widehat{\mathbf{z}}_k^i = \widehat{\mathbf{r}}_k^i \tag{2}$$

are symmetric positive definite block matrices and the preconditioned conjugate gradient method with block–size reduction block–ILU preconditioner proposed by Chan and Vassilevski [8] (see also: [5] and [7]) is used to solve the systems in each sub-domain.

4.3 Short description of the code

The parallel code follows the main ideas in the sequential code for the same problem. As was mentioned above, we solve the system of linear equations (1) with the Preconditioned Conjugate Gradient Method with preconditioner constructed via overlapping domain decomposition technique. The domain decomposition preconditioner is used as an outer preconditioner while the Block Size Reduction–MILU preconditioner is used as an inner preconditioner for solving the systems (2) by PCG method in each sub-domain in parallel. The outer iteration procedure only is parallelized. An outline of the code follows.

```
include 'mpif.h'
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, nodes, ierr)
call MPI_GET_PROCESSOR_NAME(name, lname, ierr)
```

$\Downarrow$

$\boxed{\text{Input data}}$

$\Downarrow$

Generation of the stiffness matrix and the right–hand–side, factorizing diagonal blocks and computing entries needed for the Sherman–Morrison–Woodbury formula for each sub-domain in parallel. No communications at this stage.

$\Downarrow$

Solving the system (1) via PCG method with DD preconditioner

Computing the initial iterate in parallel
PCG method with BSR-MILU preconditioner.
Each processor consists of its own part of the vector of the initial iterate.
No communications at this stage.

****

Computing the initial defect in parallel.
This stage consist of matrix–vector product and addition of vectors and therefore each processor has to send/receive a part of the vectors corresponds to the overlapping regions to/from his neighbors. After that the processors work in parallel and each processor consists of its own part of the vector of the initial defect.

MPI communication primitives used:

MPI_ISEND, MPI_IRECV, MPI_WAITALL

****

Computing the initial quasi-defect in parallel
PCG method with BSR-MILU preconditioner.
Each processor consists of its own part of the vector of the initial quasi-defect.
No communications at this stage.

****

Iteration cycle:

Dot products, vectors additions and matrix–vector product.

MPI communication primitives used:

MPI_ISEND, MPI_IRECV, MPI_WAITALL, MPI_SUM, MPI_ALLREDUCE.

The computing of the next search direction, which leads to solving systems of equations, is done in parallel in each sub-domain without any communications (PCG method with BSR-MILU preconditioner).

$\Downarrow$

$\boxed{\text{Output data}}$

$\Downarrow$

```
call MPI_FINALIZE(ierr)
```

5 Displacement decomposition circulant block factorization preconditioning

Let us remind that a circulant matrix C has the form $(C_{k,j}) = \left(c_{(j-k) \bmod m}\right)$, where m is the size of C. Let us also denote for any given coefficients $(c_0, c_1, \dots, c_{m-1})$ by $C = (c_0, c_1, \dots, c_{m-1})$ the circulant matrix

$$\begin{bmatrix} c_0 & c_1 & c_2 & \dots & c_{m-1} \\ c_{m-1} & c_0 & c_1 & \dots & c_{m-2} \\ \vdots & \vdots & \vdots & & \vdots \\ c_1 & c_2 & \dots & c_{m-1} & c_0 \end{bmatrix}.$$

Any circulant matrix can be factorized as

$$C = F \Lambda F^*,$$

where Λ is a diagonal matrix containing the eigenvalues of C, and F is the Fourier matrix

$$F = \frac{1}{\sqrt{m}} \left\{ e^{2\pi \frac{jk}{m} \mathbf{i}} \right\}_{0 \le j,k \le m-1}. \tag{3}$$

Here $\mathbf{i}$ stands for the imaginary unit.

The main goal of this study is analysis of the parallel complexity of the PCG method with considered displacement decomposition circulant block-factorization preconditioner. We estimate only the parallel execution time for one PCG iteration. We analyze the parallel complexity, speed-up and parallel efficiency for the studied preconditioner on various standard architectures.

5.1 Displacement decomposition

There are a lot of works dealing with preconditioning iterative solution methods for the FEM elasticity systems. In an earlier paper, Axelsson and Gustafson [2] have implemented modified point-ILU factorization for such problem. As the coupled system does not lead to an M-matrix, they construct their preconditioners based on the point-ILU factorization of the displacement decoupled block-diagonal part of the original matrix. This approach is known as *displacement decomposition*. The displacement decomposition remains until now one of the most robust approaches (see also, e.g., [3]). A new block-ILU factorization preconditioner based on block-size reduction is studied in [5, 10]. An advantage of this approach is that the approximate factorization exists for symmetric and positive definite block-tridiagonal matrices that are not necessarily M-matrices.

To define the displacement decomposition preconditioner M_{DD} of the matrix K we consider the auxiliary Laplace equation $-u_{x_1x_1}-u_{x_2x_2}-u_{x_3x_3}=f$ with boundary conditions corresponding to the coupled elasticity problem. Let us assume primary, that this Laplace equation is discretized by the same finite elements as the original problem and let K_0 be the corresponding stiffness matrix.

The following Korn's inequality is the theoretical background of the displacement decomposition methods:

$$\kappa(M_{DD}^{-1}K) = \mathcal{O}\left(\frac{1}{1-2\nu_{\max}}\right), \tag{4}$$

where $\nu_{\max} = \max_{\Omega} \nu$, $\nu \in (0, 1/2)$ is the Poisson ratio and

$$M_{DD} = \begin{pmatrix} K_0 & 0 & 0 \\ 0 & K_0 & 0 \\ 0 & 0 & K_0 \end{pmatrix}. \tag{5}$$

The next step in the construction used here is to substitute K_0 by A_0 in (5), where A_0 stands for the Laplace stiffness matrix corresponding to linear finite elements or seven point finite difference stencil discretization. This is motivated by the more simple structure of A_0 as well as by the spectral equivalence

$$\kappa(A_0^{-1}K_0) = \mathcal{O}(1). \tag{6}$$

5.2 Circulant block factorization

The CBF preconditioning technique incorporates the circulant approximations into the framework of the *LU* block factorization. The computational efficiency and parallelization of the resulting algorithm are as high as of the block circulant one.

If we consider the 2D elliptic problem:

$$-(a(x_1,x_2)u_{x_1})_{x_1} - (b(x_1,x_2)u_{x_2})_{x_2} = f(x_1,x_2) \tag{7}$$

on the unit square $[0,1]\times[0,1]$ with Dirichlet boundary condition, where the domain is discretized by uniform grid with n_1 and n_2 grid points in the coordinate directions and a standard five-point FDM (FEM) approximation is used, then the stiffness matrix K admits a block-tridiagonal structure where the diagonal blocks are tridiagonal matrices and the off-diagonal blocks are diagonal matrices.

For this problem, the general form of the *circulant block factorization preconditioner* is defined by

$$M_{CBF} = tridiag(-C_{i,i-1}, C_{i,i}, -C_{i,i+1}) \qquad i = 1, 2, \ldots, n_1, \tag{8}$$

where $C_{i,j} = Circulant(K_{i,j})$ is a circulant approximation of the corresponding block $K_{i,j}$.

Let us consider now the similar 3D elliptic problem:

$$-(a(x_1,x_2,x_3)u_{x_1})_{x_1} - (b(x_1,x_2,x_3)u_{x_2})_{x_2} - (c(x_1,x_2,x_3)u_{x_3})_{x_3} = f(x_1,x_2,x_3) \tag{9}$$

on the unit cube $[0,1] \times [0,1] \times [0,1]$ with Dirichlet boundary condition. If the domain is discretized by uniform grid with n_1, n_2 and n_3 grid points in the coordinate directions, and if a standard (for such a problem) seven-point FDM (FEM) approximation is used, then the stiffness matrix K admits again a block-tridiagonal structure. The matrix K can be written in the block-form

$$K = tridiag(-K_{i,i-1}, K_{i,i}, -K_{i,i+1}) \qquad i = 1, 2, \ldots, n_1,$$

where $K_{i,i}$ are block-tridiagonal matrices which correspond to one x_1-plane and off-diagonal blocks are diagonal matrices.

Now, we apply the general CBF preconditioning approach and construct the preconditioner M_{CBF} in the form:

$$M_{CBF} = tridiag(-C_{i,i-1}, C_{i,i}, -C_{i,i+1}) \qquad i = 1, 2, \ldots n_1, \tag{10}$$

where $C_{i,j} = Block - Circulant(K_{i,j})$ is a block-circulant approximation of the corresponding block $K_{i,j}$. The relative condition number of the CBF preconditioner for the model (Laplace) 3D problem is studied in [14] for $n_1 = n_2 = n_3 = n$, where the following estimate is derived:

$$\kappa(M_0^{-1} A_0) \leq 4n. \tag{11}$$

We use here the CBF preconditioner (10) for the matrix A_0. In this way we get the DD CBF preconditioner for the stiffness matrix K of the problem (1) defined by

$$M_{DD\ CBF} = \begin{pmatrix} M_0 & 0 & 0 \\ 0 & M_0 & 0 \\ 0 & 0 & M_0 \end{pmatrix}.$$

Now, we summarize (4), (6) and (11), and get the following estimate for the condition number of the DD CBF preconditioner

$$\kappa(M_{DD\ CBF}^{-1} K) = \mathcal{O}\left(\frac{n_{\max}}{1 - 2\nu_{\max}}\right). \tag{12}$$

Unfortunately, the constant in (12) increases when the coefficient of the differential problem are varying.

5.3 Solving a system with a circulant block-factorization preconditioner

In the previous section we described an algorithm to construct a preconditioner M for a given block-tridiagonal matrix.

The main advantage of the circulant preconditioners is that they possess much more parallelism compared to the ILU preconditioners and this is what we will show in this section. We describe now how to implement an action of the inverse of the preconditioner on a given vector.

Using the standard LU factorization procedure, we can first split $M_0 = D - L - U$ into its block-diagonal and (negative) strictly block-triangular parts respectively. Then the *exact* block-factorization of M_0 can be written in the form

$$M_0 = (X - L)(I - X^{-1}U),$$

where $X = diag(X_1, X_2, \ldots, X_n)$ and the blocks X_i are determined by the recursion

$$X_1 = C_{1,1}, \text{ and } X_i = C_{i,i} - C_{i,i-1}X_{i-1}^{-1}C_{i-1,i}, \qquad i = 2, \ldots, n. \tag{13}$$

It is easy seen that X_i are also block-circulant matrices.

In order to compute $M_0^{-1}\mathbf{u}$ we rewrite the block-circulant blocks of preconditioner as

$$C_{i,j} = (F \otimes F)\Lambda_{i,j}(F^* \otimes F^*).$$

Here $\otimes$ denotes the Kronecker product. Then for X_i in (13) we have

$$X_i = (F \otimes F)D_i^{-1}(F^* \otimes F^*)$$

and the latter yields

$$\begin{aligned} D_1^{-1} &= \Lambda_{1,1}, \\ D_i^{-1} &= \Lambda_{i,i} - \Lambda_{i,i-1}D_{i-1}\Lambda_{i-1,i}. \end{aligned}$$

Let $\Lambda = tridiag(\Lambda_{i,i-1}, \Lambda_{i,i}, \Lambda_{i,i+1})$. Then the following relation holds

$$M_0\mathbf{w} = \mathbf{u} \qquad \Longleftrightarrow \qquad (I \otimes F \otimes F)\Lambda(I \otimes F^* \otimes F^*)\mathbf{w} = \mathbf{u}.$$

The above system can be written as

$$\begin{pmatrix} \mathcal{F} & & & & \\ & \mathcal{F} & & & \\ & & \mathcal{F} & & \\ & & & \ddots & \\ & & & & \mathcal{F} \end{pmatrix} \begin{pmatrix} \Lambda_{11} & \Lambda_{12} & & & \\ \Lambda_{21} & \Lambda_{22} & \Lambda_{23} & & \\ & \Lambda_{32} & \Lambda_{33} & & \\ & & & \ddots & \\ & & & & \Lambda_{nn} \end{pmatrix} \begin{pmatrix} \mathcal{F}^* & & & & \\ & \mathcal{F}^* & & & \\ & & \mathcal{F}^* & & \\ & & & \ddots & \\ & & & & \mathcal{F}^* \end{pmatrix} \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \\ \vdots \\ \mathbf{w}_n \end{pmatrix} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_n \end{pmatrix}$$

where $\mathcal{F} = F \otimes F$.

We can distinguish three stages in computing $\mathbf{w} = M_0^{-1}\mathbf{u}$:

$$\begin{array}{ll} (1) & \hat{\mathbf{u}} = (I \otimes F^* \otimes F^*)\mathbf{u} \\ (2) & \Lambda\hat{\mathbf{w}} = \hat{\mathbf{u}} \\ (3) & \mathbf{w} = (I \otimes F \otimes F)\hat{\mathbf{w}}. \end{array} \tag{14}$$

Due to the special form of F, as written in (3), we can use a fast Fourier transform to perform the first and third stages, namely we use two-dimensional block-FFT (see [17]). The second stage consist of solving of two recurrence equations

$$\left| \begin{array}{l} \hat{\mathbf{v}}_1 = D_1\hat{\mathbf{u}}_1 \\ \hat{\mathbf{v}}_i = D_i(\hat{\mathbf{u}}_i - \Lambda_{i,i-1}\hat{\mathbf{v}}_{i-1}) \\ \qquad i = 2, 3, \ldots n \end{array} \right. \qquad \left| \begin{array}{l} \hat{\mathbf{w}}_n = \hat{\mathbf{v}}_n \\ \hat{\mathbf{w}}_i = \hat{\mathbf{v}}_i - D_i\Lambda_{i,i+1}\hat{\mathbf{w}}_{i+1} \\ \quad i = n-1, n-2, \ldots 1 \end{array} \right. \tag{15}$$

Block-FFT is well parallelizable (see, e.g., in [17]) and (15) consist n independent systems which can be calculated in parallel. We recall that the blocks D_i and $\Lambda_{i,j}$ in the recurrences (15) are diagonal matrices.

5.4 Model of computation and communication times

We assume that the computations and communications are not overlapped and therefore, the execution time of the parallel implementation is the sum of the computation time and the communication time.

We shall use in our analysis standard models for the arithmetic and communication times [19]. First, assuming no arithmetic vectorization, the execution of M arithmetic operations on one processor takes time

$$T_a = M * t_a, \tag{16}$$

where t_a is the average unit time to perform one arithmetic operation on one processor. Second, the local communication time of transfer of M data from one processor to its neighbor is approximated by

$$T_{local} = t_s + M * t_c, \tag{17}$$

where t_s is the start-up time and t_c is the incremental time necessary for each of all M words to be sent.

Finally, for the various architectures, we denote the following two communication times, respectively by:

- Broadcasting a number from one processor to all others: $b(p)$, where there are p processors in the computer system;
- Gathering p data packets, each packet with M/p words, in one processor from all others: $g(M, p)$.

In this report we discuss three distributed memory architecture models: ring, square grid and hypercube, and shared memory model. We model the shared memory architecture by assuming that all the processors in the system can be considered as neighbors. For these architectures the above communication times are respectively:

	$b(p)$	$g(M,p)$
ring	$(t_s + t_c)\frac{p}{2}$	$p\, t_s + M\, t_c$
grid	$(t_s + t_c)\sqrt{p}$	$2\sqrt{p}\, t_s + M(1 + \frac{1}{\sqrt{p}})\, t_c$
hypercube	$(t_s + t_c)\log p$	$\log p\, t_s + M(1 - \frac{1}{p})\, t_c$
shared memory	$\lceil \log p \rceil (t_s + t_c)$	$(p-1)(t_s + \frac{M}{p} t_c)$

We consider broadcasting from the processor in the center of the square grid to obtain an optimal time. This assumption is not essential for the algorithm itself.

The iterative methods are highly usable on parallel computers because the main ingredients of these methods can be efficiently implemented in parallel. The basic computational steps of iterative methods are:

1. vector updates;
2. matrix-vector products;
3. inner product;
4. preconditioning step, i.e., the solution of systems $M\mathbf{w} = \mathbf{u}$.

The vector updates are of the linked triad type $\mathbf{x} = \mathbf{x} + \alpha\mathbf{s}$. In general they do not require communications and parallelize trivially: each processor updates its own segment of the vector and there is no need of communication.

Matrix-vector products: since each processor holds only its segment of the vector in its memory there is a need to exchange vector components between some processors. However, in problems that arise from finite difference and finite element methods for partial differential equations, there is a natural way to map vector entries to processors so that only communication between nearest neighbor processors occur. Let the domain Ω of definition of the partial differential equation is partitioned in subdomains equal to the number of processors. Each grid point in Ω corresponds to one element of the solution vector $\mathbf{x}$. Then let vector elements corresponding to grid points in one subdomain are stored on one and the same processor.

To achieve a low cost of the processor synchronization and communication, we partition the computational domain into p parallelepipeds so that each parallelepiped contains n_1/p x_1-planes. We map all grid points in one parallelepiped onto a processor so that the values at the grid points in neighbor parallelepiped are computed on the neighbor processor. In this way, the matrix-vector product requires a communication of only values from one grid plane with neighbor processors. Thus only local data exchange is required, namely for the nodes on the boundaries of neighboring subdomains (see Fig. 1).

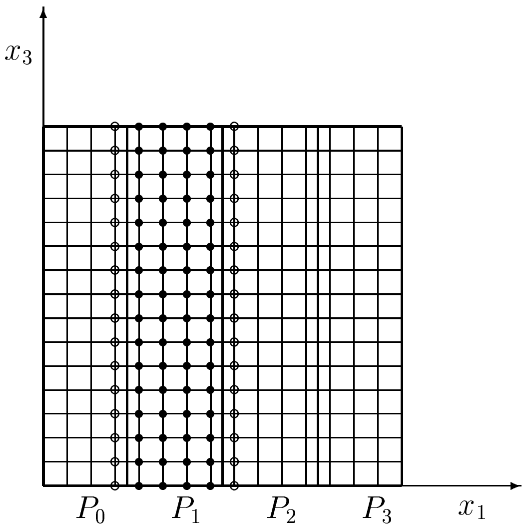

Figure 1: Vertical mapping of vector entries onto processors.

The inner products, however, require global communications. Each processor computes the local inner product for the segments of the two vectors mapped on it and then the local inner products are sent to all other processors to add up the total. In this way the processors compute the global inner product.

The preconditioning step is the most important part but in general also the most difficult part to implement. The main goal is to construct preconditioners which are both efficient on sequential machines and on parallel computers. The simplest preconditioning method is the diagonal scaling method but it is often insufficient to reduce the number of iterations. The standard incomplete factorization methods can reduce the number of iterations but the recursive depth of the preconditioner grows with n. Therefore such methods parallelize well only for a moderate number of processors. For certain problems such as elliptic differential equation, there exist various multilevel and multigrid methods. If properly implemented they are efficient on both sequential and parallel computers. Another class of preconditioners are the approximate factorization type of methods for which parallelization is more straightforward and they form an important class of methods, in particular for problem classes where multilevel methods are difficult to apply.

5.5 Parallel implementation of circulant block-factorization preconditioners

As was shown in (14) the CBF preconditioning can be split in three stages. If we use the vertical mapping from Fig. 1 for first and third stage there is no need of communication because we perform block-FFT on blocks which are stored on one processor. For second stage we have to reorder the vector entries as shown on Fig. 2. Then each of the recurrences

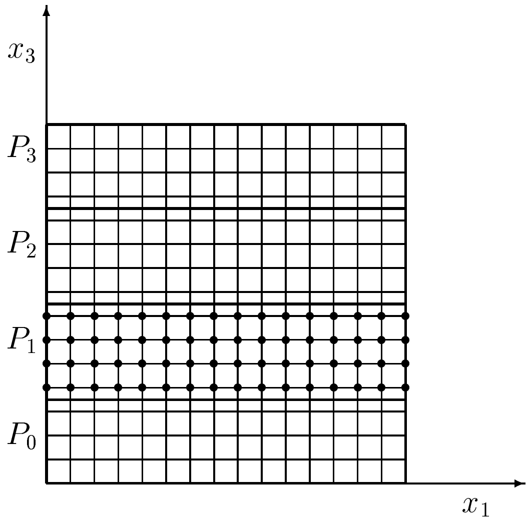

Figure 2: Horizontal mapping of vector entries onto processors.

in (15) can be split between the processors so that each processors will work only on the vector entries which are stored on it. In such way again there is no need of communication between processors. But we have to reorder the entries of the vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{w}}$ to coordinate first, second and third stages. This reordering corresponds to a matrix transposition on a parallel computer. Thus, if we use the first mapping for vector entries, we have to do a matrix transposition two times for each iteration step.

5.6 Analysis of the parallel complexity

We can now estimate the total execution time T_{PCG} for one PCG iteration for the considered circulant block-factorization preconditioner on a studied parallel system with p processors. Each iteration consists of one matrix vector multiplication with the matrix K, one multiplication with the inverse of the preconditioner $M_{DD\ CBF}$ (solving a system of equations with matrix M), two inner products and three linked triads (a vector updated by a vector multiplied by a scalar). Consequently

$$T_{PCG}(p) = T_{mult} + T_{prec} + 2T_{inn\_prod} + 3T_{triads}.$$

For simplicity we assume that the mesh sizes are equal and they are equal to an exact power of two, i.e., $n_1 = n_2 = n_3 = n = 2^l$. Then

$$T_{mult} = 243\frac{n^3}{p}t_a + 4(t_s + 3n^2 t_c),$$

$$T_{inn\_prod} = 6\frac{n^3}{p}t_a + g(p,p) + b(p),$$

$$T_{triads} = 6\frac{n^3}{p}t_a,$$

$$T_{prec} = 6\frac{n^2}{p}T_{FFT}(n) + 36\frac{n^3}{p}t_a + 2g(3\frac{n^3}{p}, p),$$

and where $T_{FFT}(n)$ is the time for execution of FFT on a given n-vector on one processor. If we use 2-radix algorithm, then $T_{FFT}(n) = 5n\log n t_a$.

Combining these results we obtain the following estimates of the execution times for the studied architectures

$$T_{PCG}(p) \;=\; 3\,(103 + 10\log n)\,\frac{n^3}{p}t_a + 4\left(t_s + 3n^2 t_c\right) + 2g(3\frac{n^3}{p}, p) + 2g(p,p) + 2b(p).$$

For example, for shared memory model we obtain

$$\begin{aligned} T_{PCG}(p) \;&=\; 2\,(p + \lceil \log p \rceil + 1)\,t_s + 2\left[3(p-1)\frac{n^3}{p^2} + 6n^2 + \lceil \log p \rceil\right] t_c \\ &+\; 3\,(103 + 10\log n)\,\frac{n^3}{p}t_a \end{aligned}$$

Then, the leading terms of the parallel time complexity functions are:

$$T_{PCG}(p) \approx 2pt_s + 6(1 - \frac{1}{p})\frac{n^3}{p}t_c + 3(103 + 10\log n)\frac{n^3}{p}t_a. \tag{18}$$

Our next goal is to analyze the relative speed-up S_p and the relative efficiency E_p, where

$$S_p = \frac{T(1)}{T(p)} \leq p \quad \text{and} \quad E_p = \frac{S_p}{p} \leq 1.$$

We apply now (18) and obtain:

$$S_p \approx \frac{3(103 + 10\log n)}{2\frac{p^2}{n^3}\frac{t_s}{t_a} + 6(1 - \frac{1}{p})\frac{t_c}{t_a} + 3(103 + 10\log n)} p. \tag{19}$$

Obviously, for our preconditioner,

$$\lim_{n\to\infty} S_p = p \quad \text{and} \quad \lim_{n\to\infty} E_p = 1,$$

i.e., the algorithms are asymptotically optimal. More precisely, if

$$\log n \gg \frac{p^2}{n^3}\frac{t_s}{t_a} + \frac{t_c}{t_a},$$

then E_p is near to 1. Unfortunately, the start-up time t_s is usually much larger than t_a, and for relatively small n the first term of the denominators in (19) is significant, in this case the efficiency is much smaller than 1.

6 Conclusion and outlook

The overlapping domain decomposition technique is an effective tool to determine high-performance parallel algorithms for implementation on parallel computers or on clusters of workstations. The decomposition of the computational domain in several subdomains allows to solve large-scale problems because of possibility to store different part of the matrices into the memory of the different computers. Moreover, the computational work may be done in parallel for the big part of the algorithm.

The computation and the inversion of the circulant block-factorization preconditioners are highly parallelizable on a wide variety of architectures. For the used timing model we conclude in general that for relatively large problems or for relatively small number of processors the influence of the architecture on the execution time is negligible. The speed-up and the parallel efficiency increase with the size of the discrete problem.

References

[1] O. Axelsson. *Iterative solution methods.* Cambridge University Press, Cambridge, 1994.

[2] O. Axelsson and I. Gustafsson. Iterative methods for the solution of the Navier equations of elasticity. *Comp.Meth.Appl.Mech.Eng.*, 15:241–258, 1978.

[3] R. Blaheta. Displacement decomposition–incomplete factorization preconditioning techniques for linear elasticity problems. *Num. Lin. Alg. Appl.*, 1:107–128, 1994.

[4] R.H. Chan and T.F. Chan. Circulant preconditioners for elliptic problems. *J. Num. Lin. Alg. Appl.*, 1:77–101, Mar. 1992.

[5] T.F. Chan, S.D. Margenov, and P.S. Vassilevski. Performance of block–ILU factorization preconditioners based on block–size reduction for 2D elasticity systems. *SIAM J.Sci.Comput.*, 18(5):1355–1366, 1997.

[6] T.F. Chan and T.P. Mathew. Domain decomposition algorithms. *Acta Numerica*, pages 61–143, 1994.

[7] T.F. Chan and P.S. Vassilevski. Convergence analysis of block-ILU factorization algorithms based on block–size reduction. Math. Dept. 95-46, UCLA, 1995. to appear in Math. Comp.

[8] T.F. Chan and P.S. Vassilevski. A framework for block-ILU factorizations using block-size reduction. *Math.Comp.*, 64(209):129–156, 1995.

[9] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994. Special issue on MPI.

[10] A. Georgiev, S. Margenov, P.S. Vassilevski, and L. Zikatanov. Application of the block-size reduction BILU algorithm for 3D simulation of pile foundations. In Zikmund Rakowski, editor, *Geomechanics 96*, pages 153–158, Rotterdam, 1997. A.A.Balkema. Proceedings of the conference Geomechanics'96, Rožnov, ČR.

[11] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* The MIT Press, Cambridge, Massachusetts, 1996. Third printing.

[12] S. Holmgren and K. Otto. Iterative solution methods for block-tridiagonal systems of equations. *SIAM J. Matr. Anal. Appl.*, 13:863–886, 1992.

[13] T. Huckle. Some aspects of circulant preconditioners. *SIAM J. Sci. Comput.*, 14:531–541, 1993.

[14] I. Lirkov and S. Margenov. Conjugate gradient method with circulant block-factorization preconditioners for 3D elliptic problems. In *Proceedings of the Workshop # 3 Copernicus 94-0820 "HIPERGEOS" project meeting, Rožnov pod Radhoštěm*, 1996.

[15] I. Lirkov and S. Margenov. Parallel complexity of conjugate gradient method with circulant preconditioners. In R. Volmar, W. Erhard, and V. Jossifov, editors, *Proceedings of the Parcella'96*, volume 96 of *Mathematical research*, pages 279–286, Berlin, 1996. Akademie Verlag.

[16] I. Lirkov, S. Margenov, and P.S. Vassilevski. Circulant block-factorization preconditioners for elliptic problems. *Computing*, 53(1):59–74, 1994.

[17] C. Van Loan. *Computational frameworks for the fast Fourier transform.* SIAM, Philadelphia, 1992.

[18] S.D. Margenov and I.T. Lirkov. Preconditioned conjugate gradient iterative algorithms for transputer based systems. In K.Boyanov, editor, *Parallel and distributed processing*, pages 406–415, Sofia, 1993.

[19] Y. Saad and M.H. Schultz. Data communication in parallel architectures. *Parallel Comput.*, 11:131–150, 1989.

[20] M. Snir, St. Otto, St. Huss-Lederman, D. Walker, and J. Dongara. *MPI: The Complete Reference.* Scientific and engineering computation series. The MIT Press, Cambridge, Massachusetts, 1997. Second printing.

[21] D. Walker and J. Dongara. MPI: a standard Message Passing Interface. *Supercomputer*, 63:56–68, 1996.