Project Report HG CP 94–0820–CICT–4–2

DD–BSR-MILU preconditioner for 3-D linear elasticity problems on a cluster of workstations

Krassimir GEORGIEV
Bulgarian Academy of Sciences, Sofia, Bulgaria

1 Introduction

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. Among the reasons underly the interest in these techniques 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. 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 supperlinear speed-up that is some times observed.

We consider an overlapping domain decomposition preconditioner for solving a 3-D linear elasticity problem on a cluster of workstations working under MPI (Message Passing Interface).

Let $\mathcal{B}$ be an elastic body occupying bounded polyhedral domain $\Omega \in I\!R^3$, 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

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

where

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

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_Ω(**u,v**) is a second order, symmetric and coercive bilinear form.

2 Statment 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 nonoverlapping subdomain approach.

Now, we will stop at Schwarz algorithms based on many overlapping subdomains [4].

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 subdomain 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 $V^H(\Omega) \subset (H_0^1(\Omega))^3$ and $V^h(\Omega) \subset (H_0^1(\Omega))^3$ be the spaces of continuous piecewise linear functions, on the triangulations, which vanish on the boundary $\partial\Omega$. Then, the discrete variational problem is:

Find: $u_h \in V^h(\Omega)$, such that

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

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 $K_{ij} = a_\Omega(\Phi_i, \Phi_j)$.

Let K_i be a local stiffness matrix according to the subdomain $\widehat{\Omega}_i$. 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 subdomain $\widehat{\Omega}_i$ 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 K_i corresponding to the subdomain $\widehat{\Omega}_i$ is

$$K_i \;=\; R_i\, K\, R_i^T.$$

In the case of many subdomains, the additive Schwarz algorithm is a generalization of the well known two-subdomain 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

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

As $R_i^T\ K_i^{-1}\ R_i\ \mathbf{y}$ can be computed in parallel for the different subdomains, this leads to a coarse grain parallelism. Taking into account the definition of the resticton 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.

3 The Solver

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

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

for $k = 0, 1, \ldots$ *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* ;
- $\alpha_k = \dfrac{(\mathbf{r}_k, \mathbf{z}_k)}{(A\mathbf{p}_k, \mathbf{p}_k)}$;
- $\mathbf{U}_{k+1} = \mathbf{U}_k + \alpha_k \mathbf{p}_k$ – *next iterate*;;
- $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$– *next quasiresidual*;

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 [5] (see also: [3] and [6]) is used to solve the systems in each subdomain.

The computational domain is the parallelepiped

$$\Omega = [0, x_{max}] \times [0, y_{max}] \times [0, z_{max}].$$

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 $z = 0$, i.e the displacements are equal to zero;
- Homogeneous Neumann boundary conditions on the *vertical sides* of the parallelepiped
$$x = 0, \quad x = x_{max}, \quad y = 0, \quad y = y_{max},$$
i.e the stresses are equal to zero;
- Nonhomogenuous Neumann boundary conditions on the *top side* of the parallelepiped $z = z_{max}$, where the load from the upper construction is applied.

4 MPI

MPI is standard message passing interface for applications and libraries running on concurent computers with a logicaly distributed memory [7]. It is used successivly 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 cals 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 3-D linear elasticity problem on a cluster of workstations. As one can improve performance on many systems by overlapping communications and computations, in this 0–version of our computer code we have used **nonblocking communications** as well as some **collective communications**. The nonblocking communications use opaque **request** objects to idetify 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 erach 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 erach 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 convinient 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)

5 Short description of the code

The parallel code follows the main ideas in the sequential code for the same problem. As was mentioned in Section 3 of this Report, 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 PCGM in each subdomain in parallel. The outer iteration procedure only is parallelized. An outline of the code is 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)
```

$\Downarrow$

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 subdomain in parallel. No communications at this stage.

$\Downarrow$

⇓

Solving the system (1) via PCGM with DD preconditioner

Computing the initial iterate in parallel
PCGM 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 neighbours. 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 quasidefect in parallel
PCGM with BSR-MILU preconditioner.
Each processor consists of its own part of the vector of the initial quasidefect.
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_ALLREDUCE, MPI_SUM.

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

⇓

$\Downarrow$

Output data

$\Downarrow$

call MPI_FINALIZE(ierr)

6 Conlusion and outlook

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

The existing now 0-version of the code for solving a 3-D linear elasticity problem on a cluster of workstations working under MPI is only a first stage in our experience in doing parallel codes. Many of the good news from MPI have not been used yet. Our idea is to continue work in this direction and in the near future to have an working version of this code and some numerical tests on a real cluster of workstations. On the other hand we are going to try some other MPI primitives in order obtain beter ratio between computations and communications. Up to now we have some numerical experiments for the classical Schwarz algorithms based on many overlapping subdomains on a cluster of four workstations located at ULB–Brussels, but for the Poisson equation in a 2-D domain as well as some simulations on SUN Sparc 20.

References

[1] O. Axelsson, *Iterative solution methods*, Cambridge Univ.Press, (1994)

[2] O. Axelsson, V. A. Barker, *Finite element solution of boundary value problems: theory and applications*, Orlando: Academic Press, (1983)

[3] T. Chan, S. Margenov, P. Vassilevski, *Performance of Block-ILU factorization preconditioning techniques for linear elasticity problems*, SIAM J. Sci.Comput. (to appear)

[4] T. Chan, T. Mathew, *Domain decomposition algorithms*, Acta Numerica (1994), 61-143

[5] T. Chan, P. Vassilevski, *A framework for block-ILU factorizations using block–size reduction*, Math. Comp. (to appear)

[6] T. Chan, P. Vassilevski, *Convergence analysis of block–ILU factorization algorithms based on block–size reduction*, UCLA CAM Report, Math. Comp. (to appear)

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