Comparative Analysis of High Performance Solvers for 3D Elasticity Problems

Ivan Lirkov[1], Yavor Vutov[1], Maria Ganzha[2], and Marcin Paprzycki[2]

[1] Institute for Parallel Processing, Bulgarian Academy of Sciences,
Acad. G. Bonchev, Bl. 25A, 1113 Sofia, Bulgaria,
ivan@parallel.bas.bg yavor@parallel.bas.bg
http://parallel.bas.bg/~ivan/ http://parallel.bas.bg/~yavor/
[2] Systems Research Institute, Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland,
paprzyck@ibspan.waw.pl maria.ganzha@ibspan.waw.pl
http://mpaprzycki.swps.edu.pl http://www.ganzha.euh-e.edu.pl

Abstract. We consider the numerical solution of 3D linear elasticity equations. The investigated problem is described by a coupled system of second order elliptic partial differential equations. This system is then discretized by conforming or nonconforming finite elements. After applying the Finite Element Method (FEM) based discretization, a system of linear algebraic equations has to be solved. In this system the stiffness matrix is large, sparse and symmetric positive definite. In the solution process we utilize a well-known fact that the preconditioned conjugate gradient method is the best tool for efficient solution of large-scale symmetric systems with sparse positive definite matrices. In this context, the displacement decomposition (DD) technique is applied at the first step to construct a preconditioner that is based on a decoupled block diagonal part of the original matrix. Then two preconditioners, namely the Modified Incomplete Cholesky factorization MIC(0) and the Circulant Block-Factorization (CBF) preconditioning, are used to precondition thus obtained block diagonal matrix.
As far as the parallel implementation of the proposed solution methods is concerned, we utilize the Message Passing Interface (MPI) communication libraries. The aim of our work is to compare the performance of the two proposed preconditioners: the DD MIC(0) and the DD CBF. The presented comparative analysis is based on the execution times of actual codes run on modern parallel computers. Performed numerical tests demonstrate the level of parallel efficiency and robustness of the proposed algorithms. Furthermore, we discuss the number of iterations resulting from utilization of both preconditioners.

1 Introduction

Our work concerns development and implementation of efficient parallel algorithms for solving elasticity problems arising in geosciences. Typical application problems include simulation of foundations of engineering constructions (that

transfer and distribute the total load into a bed of soil) and multilayer media with strongly varying material characteristics. Here, the spatial framework of the construction produces a complex stressed-strained state in the active interaction zones. The modern design of cost-efficient construction with a sufficient guaranteed reliability requires determining parameters of this stressed-strained state.

This type of engineering problems is described mathematically by a system of three-dimensional nonlinear partial differential equations. A finite element (or finite difference) discretization reduces the partial differential equation problem to a system of linear equations $K\mathbf{x} = \mathbf{f}$, where the stiffness matrix K is large, sparse and symmetric positive definite. It is a well known fact that Conjugate Gradient (CG) type methods are claimed to be the most cost-effective way to solve problems of this type, c.f. [1]. Furthermore, to accelerate convergence of the iterative process, a preconditioner M is combined with the CG algorithm.

To make a reliable prediction of the safety of the construction, which is sensitive to soil deformations, a very accurate model and thus a large system of sparse linear equations has to be solved. In real-life applications, such system can contain up to several millions of unknowns. Note that the numerical solution of linear systems is a fundamental operation in computer modeling of elasticity problems. Specifically, solving these linear systems is usually very time-consuming (requiring up to 90% of the total solution time). Hence, developing fast solvers for linear equations is essential. Furthermore, such algorithms should be expected to significantly speed up the simulation processes of real application problems. Due to the size of the system, an efficient iterative solver should not only have a fast convergence rate but also high parallel efficiency. Moreover, the resulting program has to be efficiently implementable on modern shared-memory, distributed memory, and shared-distributed memory parallel computers.

The remaining part of the paper is organized as follows. The considered elasticity problems are described in the next section. In section 3 we present the developed parallel solvers. The results of test experiments on parallel computer systems are provided in section 4.

2 Elasticity Problems

For simplicity, in this work we focus our attention on 3D linear elasticity problems following two basic assumptions: (1) displacements are small, and (2) material properties are isotropic. A precise mathematical formulation of the considered problem has been described in [5]. The 3D elasticity problem in the stressed-strained state can be described by a coupled system of three differential equations. Applying linearization, the nonlinear equations can be transformed into a system of three linear differential equations, which is often referred to as Lamé equations.

We restrict our considerations to the case when the computational domain Ω is a rectangular parallelepiped $\Omega = [0, x_1^{max}] \times [0, x_2^{max}] \times [0, x_3^{max}]$, where the boundary conditions on each face of Ω are of fixed type.

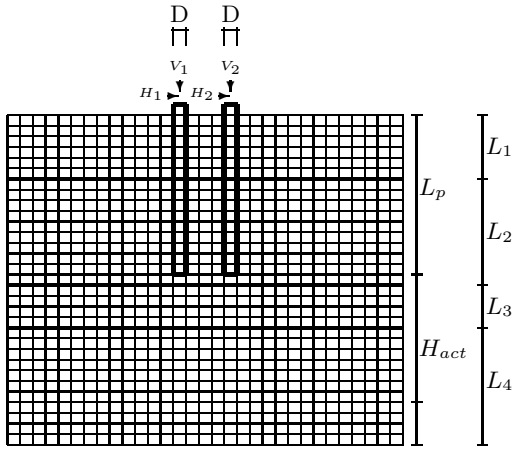

Fig. 1. Cross section of the computational domain Ω. $x_1^{max} = x_2^{max} = 37.2m$, $x_3^{max} = 31.0m$. $|H_1| = |H_2| = 150kN$, $|V_1| = 4000kN$, $|V_2| = 2000kN$, $E_{pile} = 31500MPa$, $\nu_{pile} = 0.2$, $E_{L_1} = 5.2MPa$, $\nu_{L_1} = 0.4$, $E_{L_2} = 9.4MPa$, $\nu_{L_2} = 0.35$, $E_{L_3} = 14.0MPa$, $\nu_{L_3} = 0.25$, $E_{L_4} = 21.4MPa$, $\nu_{L_4} = 0.2$.

A benchmark problem from [4] is used in numerical tests reported here. The engineering problem describes two piles in an inhomogeneous sandy clay soil (see Fig. 1). In the solution process, uniform grid is used with n grid points along each coordinate direction. In our experiments we used two kind of meshes: coarse mesh with step sizes $1.2 \times 1.2 \times 1$ and fine mesh with step sizes $0.6 \times 0.6 \times 0.5$.

3 Parallel Displacement Decomposition Solvers

There exists a substantial body of work dealing with preconditioning of iterative solution methods for elasticity systems discretized by using the Finite Element Method. For instance, in [2] Axelsson and Gustafson construct their preconditioners based on the point-ILU (Incomplete LU) factorization of the displacement decoupled block-diagonal part of the original matrix. This approach is known as the displacement decomposition (see, e.g., [3]).

Our first approach uses the DD MIC(0) (Modified Incomplete Cholesky) factorization, presented in [7]. It uses nonconforming Ranacher-Turek elements for the discretization. Modifying the block-diagonal displacement decomposed matrix, an auxiliary matrix B is obtained. This matrix has a block structure, with diagonal blocks being diagonal matrices. Then the MIC(0) factorization of the matrix B is used as a preconditioner.

The other parallel preconditioning technique used in this work is the circulant block-factorization used for preconditioning of the obtained block-diagonal matrix, c.f. [6]. Here, a displacement decomposition circulant block factorization preconditioner is constructed.

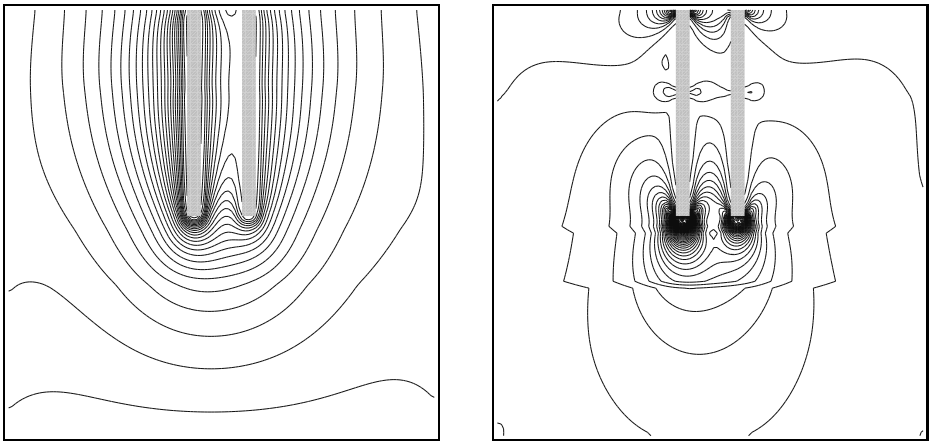

Fig. 2. Vertical displacements on the left, vertical strains on the right.

Suitable modification of the DD MIC(0) algorithm allows parallelization of the preconditioning, but results more communication steps in comparison with the DD CBF preconditioner. The estimate of the condition number of the DD CBF preconditioner shows that the convergence is asymptotically as fast as preconditioners based on the point-ILU factorization, c.f. [5, 6]. Moreover the DD CBF solver has a good parallel efficiency (see, e.g., [5, 6]).

4 Experimental Results

Before proceeding with describing results of performed benchmarking runs let us illustrate the nature of the solved problem. The obtained solution of the elasticity problem is used for computation of the vertical strain in the computational domain. Thus, in Fig. 2 vertical displacements and vertical strains are depicted in a cross section of the domain. An isoline connects points with equal values.

To efficiently solve the problem, portable parallel FEM codes were designed and implemented in C (the DD CBF code) and C++ (the DD MIC(0) code). In both cases, parallelization has been facilitated using the MPI library, c.f. [8, 9]. The parallel code has been tested on cluster computers located in the National Energy Research Scientific Computing Center (NERSC). In our experiments, times have been collected using the MPI provided timer and for each problem size and number of processors we report the best results from multiple runs. For linear system with N unknowns we represent the number of iterations as N_{it}, the elapsed time as T_p (in seconds; obtained on p processors), the speed-up as $S_p = T_1/T_p$, and the parallel efficiency as $E_p = S_p/p$. Because of the NERSC-imposed limitations in available computational time and memory in some cases we were not able to establish single-processor performance for the largest problem. Therefore, for the largest problems we report parallel efficiency related to results collected on 2 processors.

Table 1. Experimental results on Bassi.

p	N	N_{it}	T_p	S_p	E_p	N	N_{it}	T_p	S_p	E_p
DD MIC(0)										
1	2 179 548	1 533	1 514.3			17 298 000	2 840	23 101.0		
2		1 533	795.3	1.90	0.952		2 840	11 584.3	1.99	0.997
4		1 533	414.9	3.65	0.912		2 840	5 973.1	3.87	0.967
8		1 533	217.7	6.95	0.869		2 840	3 219.0	7.18	0.897
16		1 533	125.9	12.02	0.752		2 840	1 684.5	13.71	0.857
32		1 533	74.1	20.43	0.638		2 840	913.2	25.30	0.791
64							2 840	544.4	42.43	0.663
DD CBF										
1	786 432	1 297	1 912.6			6 291 456	2 641	41 075.5		
2		1 297	955.6	2.00	1.001		2 633	17 378.5	2.36	1.182
4		1 291	467.0	4.10	1.024		2 625	8 633.0	4.76	1.189
8		1 290	235.1	8.14	1.017		2 617	4 330.6	9.49	1.186
16		1 252	115.7	16.53	1.033		2 612	2 191.6	18.74	1.171
32		1 282	62.4	30.63	0.957		2 609	1 079.7	38.04	1.189
64		1 247	33.3	57.51	0.899		2 600	575.8	71.34	1.115

Table 1 summarizes results collected on the IBM p575 POWER 5 system, named Bassi [10]. Bassi is a distributed memory computer with 888 IBM POWER 5 processors (running at 1.9 GHz) distributed among 111 compute nodes with 8 processors per node. Each Bassi processor has a theoretical peak performance of 7.6 GFlop/s. Processors within each node have a shared memory pool of 32 GB. Bassi's network switch is the IBM "Federation" HPS switch which is connected to a two-link network adapter on each node. We have used IBM C and C++ compilers with options "-O3 -qstrict -qarch=auto -qtune=auto". According to the best of our knowledge, this and compiler switches used on other machines should result in maximal performance optimization.

Table 2 shows execution time on the NERSC Cray XT4 system, named Franklin [11]. Franklin is a massively parallel processing (MPP) system with 9 660 compute nodes, and the entire system has a total of 19 320 processor cores. Specifically, each of Franklin's compute nodes consists of a 2.6 GHz dual-core AMD Opteron processor with a theoretical peak performance of 5.2 GFlop/s. Each compute node has 4 GB of memory and is connected to a dedicated SeaStar2 router through the Hypertransport with a 3D torus topology. We have used the PGI C and C++ compilers with options "-fast -O3 -Minline".

The memory available on a single node of Franklin is not large enough to run our experiments for the fine mesh and we reported here the execution time starting from two processors on two different nodes.

In Table 3 we present results of experiments performed on the Jacquard [12]. It is a 712-CPU (356 dual-processor nodes running at 2.2 GHz) Opteron Linux cluster. Each processor has a theoretical peak performance of 4.4 GFlop/s. Processors within each node share 6 GB of memory and are interconnected through a high-speed InfiniBand network. We have used C/C++ compilers produced by

Table 2. Experimental results on Franklin.

p	N	N_{it}	T_p	S_p	E_p	N	N_{it}	T_p	E_p
DD MIC(0)									
1	2 179 548	1 959	3 787.7			17 298 000			
2		1 959	1 935.9	1.96	0.978		3 404	26 593.8	
4		1 959	1 001.6	3.78	0.945		3 404	13 712.8	0.970
8		1 959	526.9	7.19	0.899		3 404	6 921.7	0.961
16		1 959	286.9	13.20	0.825		3 404	3 573.8	0.930
32		1 959	167.2	22.65	0.708		3 404	1 937.2	0.858
64		1 959					3 404	1 115.7	0.745
DD CBF									
1	786 432	1 298	1 392.6			6 291 456			
2		1 294	745.9	1.87	0.933		2 632	11 944.2	
4		1 291	380.9	3.66	0.914		2 630	6 443.3	0.927
8		1 292	184.5	7.55	0.943		2 621	3 243.1	0.921
16		1 251	88.8	15.69	0.980		2 612	1 600.8	0.933
32		1 286	50.4	27.63	0.864		2 613	800.1	0.933
64		1 281	33.8	41.16	0.643		2 608	420.8	0.887

PathScale with the ACML Optimized Math Library and compiled the code using "mpicc -Ofast $ACML" command. The "-Ofast" option is a generic option leading to a vendor suggested aggressive optimization.

Several jobs submitted on Jacquard are still waiting in the queue and this is the reason for the fine mesh to report only some results from DD CBF code.

First, let us note that the number of iterations for the DD CBF varies with the number of processors. This well known effect is caused by the different order of summations in the inner product computations involved in the PCG. The same effect is not observed in the DD MIC(0) code, because of a special precaution taken during the computation of the inner products. Here, the order of the additions is made independent of the number of processors.

An interesting phenomenon is observed concerning the number of iterations for the DD MIC(0). On Bassi they are notably smaller than those on both Franklin and Jacquard. For instance, for the smaller problem the number of iterations is 1533 on Bassi and 1959 on the other two clusters. For the larger problem these numbers are 2840 and 3404 respectively. This is probably caused by the difference in the processor architectures (Power 5 vs. x86).

The number of unknowns in the nonconforming discretization of the problem (used by the DD MIC(0) solver) is about three times greater than in the conforming one (used by the DD CBF solver). Furthermore, the number of iterations of the DD MIC(0) preconditioner is also greater than the number of iterations of the CBF one. Nevertheless, computing times of both solvers are comparable, with the DD MIC(0) solver being somewhat faster. A notable exception from this are runs on Franklin, where the DD MIC(0) code performs more than two times slower. We believe that the cause of this phenomenon is the PGI compiler and its inability to appropriately optimize the C++ code (vis-a-vis the C code).

Table 3. Experimental results on Jacquard.

p	N	N_{it}	T_p	S_p	E_p	N	N_{it}	T_p	E_p
DD MIC(0)									
1	2 179 548	1 959	1 083.6						
2		1 959	535.7	2.023	1.011				
4		1 959	301.4	3.594	0.899				
8		1 959	180.7	5.997	0.750				
16		1 959	117.0	9.264	0.579				
32		1 959	81.7	13.260	0.414				
DD CBF									
1	786 432	1 260	1 277.8			6 291 456			
2		1 297	675.2	1.89	0.946		2636	12 465.1	
4		1 259	351.3	3.64	0.909		2629	6 693.9	0.931
8		1 290	185.1	6.90	0.863		2617	3 354.3	0.929
16		1 287	92.7	13.79	0.862		2610	1 664.9	0.936
32		1 286	57.7	22.16	0.692		2605	805.0	0.968
64		1 283	43.1	29.68	0.464		2602	556.0	0.701

As expected the parallel efficiency of the DD CBF solver is generally better than this of the DD MIC(0). On Bassi, the efficiency of 64% is obtained for the smaller problem on 32 processors for the MIC(0) preconditioner. For the larger problem, on 64 processors, the efficiency is 66% for the same solver. For the CBF preconditioner the lowest efficiencies are 90% and 94%, reached on 64 processors, for the smaller and larger problems respectively. There could be both software and hardware causes for the super-linear speed-up observed on Bassi. On the software side, when using more processors, the number of iterations needed for some convergence steps are smaller. On the hardware side, cache effect is usually the cause. When the sub-domains are smaller, the data each processor owns are more easily fit into cache.

The superiority of the parallel properties of the CBF solver can be tracked on Jacquard and Franklin as well, although they are not as pronounced.

To summarize, in Fig. 3 computing times on different clusters are shown for both algorithms. The left picture well illustrate the above mentioned phenomenon with slower execution of the DD MIC(0) solver on Franklin. Also, the theoretical peak performance of Bassi is the highest but with respect to the execution time on one processor Jacquard is the fastest machine.

Acknowledgments

Computer time grant from the National Energy Research Scientific Computing Center is kindly acknowledged. This research was partially supported by grant I-1402/2004 from the Bulgarian NSF. Work presented here is a part of the Poland-Bulgaria collaborative grant: “Parallel and distributed computing practices”.

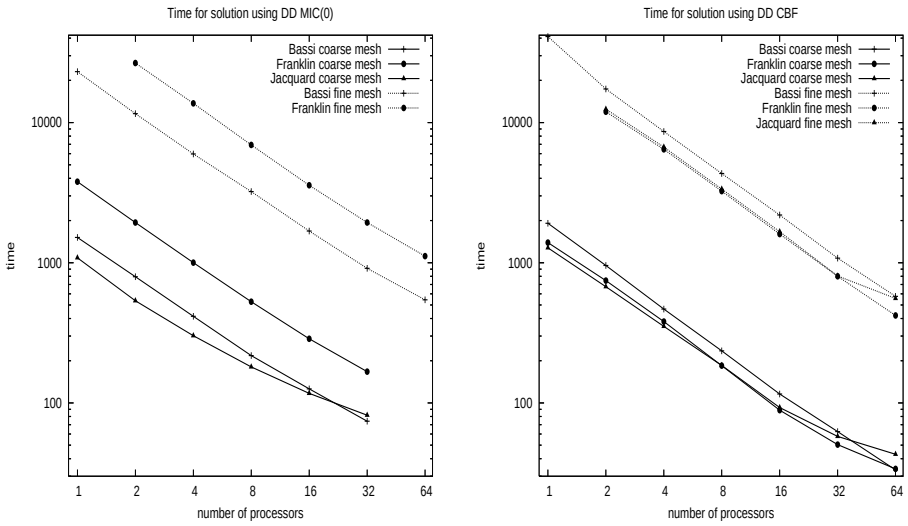

Fig. 3. Execution times for the coarse and fine mesh

References

1. O. Axelsson, *Iterative solution methods*, Cambridge Univ. Press, Cambridge, 1994.
2. O. Axelsson, I. Gustafsson, Iterative methods for the solution of the Navier equations of elasticity, *Comp.Meth.Appl.Mech.Eng.* **15**, 1978, 241–258.
3. R. Blaheta, Displacement decomposition-incomplete factorization preconditioning techniques for linear elasticity problems, *Num. Lin. Alg. Appl.* **1**, 1994, 107–128.
4. A. Georgiev, A. Baltov, S. Margenov, Hipergeos benchmark problems related to bridge engineering applications, REPORT HG CP 94–0820–MOST–4.
5. I. Lirkov, MPI solver for 3D elasticity problems, *Math. and computers in simulation*, **61** 3–6, 2003, 509–516.
6. I. Lirkov, S. Margenov, MPI parallel implementation of CBF preconditioning for 3D elasticity problems, *Math. and computers in simulation*, **50** 1–4, 1999, 247–254.
7. Y. Vutov, Parallel DD-MIC(0) Preconditioning of Nonconforming Rotated Trilinear FEM Elasticity Systems, *Large-Scale Scientific Computing*, I. Lirkov, S. Margenov, J. Waśniewski eds., *Lecture notes in computer sciences*, **4818**, Springer, 2008, 745–752.
8. M. Snir, St. Otto, St. Huss-Lederman, D. Walker, J. Dongara, *MPI: The Complete Reference*, Scientific and engineering computation series, The MIT Press, Cambridge, Massachusetts, 1997, Second printing.
9. D. Walker, J. Dongara, MPI: a standard Message Passing Interface, *Supercomputer* **63**, 1996, 56–68.
10. Bassi — IBM POWER 5, http://www.nersc.gov/nusers/systems/bassi/
11. Franklin — Cray XT4, http://www.nersc.gov/nusers/systems/franklin/
12. Jacquard — Opteron Cluster, http://www.nersc.gov/nusers/resources/jacquard/