Benchmarking Performance of an MPI-based Solver for 3D Elasticity Problems

Ivan Lirkov, Svetozar Magenov
Institute for Parallel Processing
Bulgarian Academy of Sciences
Acad. G. Bonchev, Bl. 25A, 1113 Sofia, Bulgaria
ivan@parallel.bas.bg, margenov@parallel.bas.bg Poland

Marcin Paprzycki
Computer Science, SWPS
ul. Chodakowska 19/31, 03-815 Warszawa, Poland
marcin@cs.okstate.edu

Abstract

Numerical solution of 3D linear elasticity equations is considered. Problem is described by a coupled system of second order elliptic partial differential equations. This system is discretized by trilinear parallelepipedal finite elements. Preconditioned Conjugate Gradient iterative method is used for solving large–scale linear algebraic systems arising after the Finite Element Method (FEM) discretization of the problem. The displacement decomposition technique is applied at the first step to construct a preconditioner using the decoupled block–diagonal part of the original matrix. Then circulant block–factorization is used to precondition thus obtained block–diagonal matrix. Since both preconditioning techniques, displacement decomposition and circulant block–factorization, are highly parallelizable, a portable parallel FEM code based on MPI is developed. Results of numerical tests performed on a number of modern parallel computers using real–life engineering problems from the geomechanics in geosciences are reported and discussed.

1. Introduction

Our work concerns development and implementation of efficient parallel algorithms for solving elasticity problems arising from geomechanics in geosciences. In this area, typical application problems include, among others, a variety of simulations of foundations of engineering constructions, which transfer and distribute the total loading into the bed of soil (e.g. piles supporting bridges or central columns carrying construction of a tall building) and multi–layer media with strongly varying material characteristics. Here, the spatial framework of the construction produces a composed stressed–strained state in active interaction zones. A modern design of a cost–efficient construction with a sufficient guaranteed reliability requires determining parameters of this stressed–strained state.

Application problems, of the type that attempt at coming close to modeling actual real-life situations, are three-dimensional nonlinear elasticity problems, which are described mathematically by a system of partial differential equations. When applied, a finite element (or finite difference) discretization reduces problem described by partial differential equations to a system of linear/nonlinear equations. To make a reliable prediction of the construction safety, which is sensitive to soil deformations, a very accurate model and thus a large system of sparse linear equations is required. Specifically, in real–life applications, such linear systems can be extremely large – containing up to several millions of unknowns. Hence, these problems have to be solved by robust and efficient parallel iterative methods on powerful multiprocessor computers.

Note that numerical solution of linear systems is a fundamental operation in solving elasticity problems. In fact, nonlinear equations generated from the discretization of the nonlinear elasticity problem have to be solved by an iterative procedure, in which a system involving millions of linear equations has to be solved in every iteration. Solving these linear systems is usually very time–consuming (consuming up to 90% of the total solution time). Hence, development of fast algorithms for solving systems of linear equations is essential. Furthermore, observe that solution of real-life problems under consideration occur usually in the context of engineering simulations. In this case multiple instances of the problem (for varying design parameters) have to be solved.

Therefore efficient solution algorithms can significantly speed up simulations. In this context note also that, 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 should be efficiently implementable on a variety of modern parallel computer architectures, e.g. shared–memory, distributed memory, and mixed shared–distributed memory.

2. Elasticity problems

For simplicity, in this note we focus our attention on 3D linear elasticity problems that are guided by two basic assumptions: (1) displacements are small, and (2) material properties are isotropic.

Precise mathematical formulation of the considered problem can be found in [1]. Here let us only remind, that the 3D elasticity problem in the stressed–strained state can be described by a coupled system of three differential equations. After applying standard linearization, nonlinear equations can be simplified to a system of three linear differential equations, which is often referred to as Lamé equations.

Let us restrict our considerations to the case where the computational domain Ω has the form of a cube $[0, x_1^{\max}]\times[0, x_2^{\max}]\times[0, x_3^{\max}]$, and where boundary conditions on each side of Ω are of a fixed type.

In the numerical tests reported here we utilize benchmark problems put forward in [2]. These benchmarks model (1) a situation where a single pile is placed in a homogeneous sandy clay soil (reported as *Benchmark 1*) and (2) where two piles are placed in an inhomogeneous sandy clay soil (reported as *Benchmark 2*); see [1, 2, 3] for more details. In the solution process, uniform grid is used with n_1, n_2 and n_3 grid points along the coordinate directions.

3. Displacement decomposition circulant block factorization preconditioner

There exists a large body of research dealing with preconditioning of iterative solution methods for the elasticity systems discretized using Finite Element Method. In [4] 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 a displacement decomposition (DD, for more details see, e.g., [5]). In [6] circulant block–factorization is used for preconditioning of the obtained block–diagonal matrix and a displacement decomposition circulant block factorisation preconditioner is constructed. The estimate of the condition number of the proposed preconditioner shows that DD CBF solver is asymptotically as fast as preconditioners based on the point–ILU factorization. Moreover the DD CBF solver has a very good parallel efficiency.

4. Computer Systems

The proposed approach has been tested on two cluster computers, located in the *Oklahoma Supercomputing Center for Education and Research* (OSCER) in Norman, Oklahoma, and two parallel machines located in the *High Performance Computing Center* of the University of Stuttgart, Stuttgart, Germany.

Boomer is a Pentium 4 Xeon Linux Cluster manufactured by Aspen Systems [9]. It has a total of 135 compute nodes. Each compute node consists of two Intel P4 Xeon processors running at 2.0 GHz and each having 512 KB of cache memory. Each computational node has 2 GB of memory. All nodes are connected by a Myrinet 2000 interconnect. We have been using Intel C compiler version 8.0 and compiled our code using the following options: -fast, -ptt7, -xW. MPI is supported by the MPICH software.

Shooner is an Itanium2 Linux Cluster also manufactured by Aspen Systems [10]. It has 33 compute nodes, each consisting of 2 Itanium Deerfield processors (64-bit architecture) running at 1.0 GHz; each with 1.5 MB of cache memory. Each compute node has 4 GB of memory. Nodes are connected through a SilverStorm 3000 interconnect (formerly known as InfiniCon Systems). We have used Intel C compiler version 8 and compiled the code with the following options: -O, -tpp2, -ftz. In both cases the compiler options have been suggested to us (by H. Neeman; director of OSCER) as one of the best standard optimizing configurations.

Azusa is a NEC server – Azusa Express5800/1160Xa computer [11]. It has 16 Intel Itanium processors running at 800 Mhz, with 4 MB of cache memory and 32 GB main memory. We have used a NEC C Compiler with -O3 option.

Strider is a Cray Strider Opteron Cluster [12]. It has 125 compute nodes with two AMD Opteron processors each; running at 2.0 GHz and having 516 GB of memory. Nodes are connected through a Myrinet 2000 interconnect. We have used a PGI C Compiler with -O3 option.

5. Benchmarking performance

Let us recall that the parallel code has been implemented in C and the parallelization has been facilitated using the MPI library [8]. Times have been collected using the MPI provided timer. In all cases we report the best results from multiple runs. Specifically, we report the elapsed time T_p on p processors (in seconds), the speed-up defined in a standard way $S_p=T_1/T_p$, and the parallel efficiency $E_p=S_p/p$. In our experiments we used discretizations with $n_1=n_2=n_3=n$ where n=32, 48, 64, and 96. Obviously, sizes of discrete problems are n^3.

	Benchmark 1			*Benchmark 2*		
p	Time	S_p	E_p	Time	S_p	E_p
n=32						
1	26.78			105.75		
2	16.52	1.62	0.81	64.85	1.63	0.82
4	8.919	3.00	0.75	35.19	3.01	0.75
8	4.43	6.05	0.76	17.55	6.06	0.75
16	2.63	10.18	0.64	10.19	10.38	0.65
32	1.97	13.59	0.42	7.81	13.54	0.42
n=48						
1	603.15			1106.71		
2	354.07	1.70	0.85	648.96	1.71	0.85
3	208.62	2.89	0.96	354.41	3.12	1.04
4	189.02	3.19	0.80	346.15	3.20	0.80
6	132.24	4.56	0.76	223.55	4.95	0.83
8	99.02	6.09	0.76	181.35	6.10	0.76
12	63.99	9.43	0.79	116.33	9.51	0.79
16	48.48	12.44	0.78	82.74	13.38	0.84
24	34.97	17.25	0.72	64.97	17.03	0.71
48	24.15	24.98	0.52	41.12	26.91	0.56
n=64						
1	1499.02			2461.25		
2	975.12	1.54	0.77	1602.1	1.54	0.77
4	492.41	3.04	0.76	815.43	3.02	0.75
8	262.913	5.70	0.71	432.61	5.69	0.71
16	145.37	10.31	0.64	240.26	10.24	0.64
32	72.45	20.69	0.65	115.87	21.24	0.66
64	48.24	31.07	0.49	78.81	31.23	0.49
n=96						
1	9916.38			17011		
2	6428.79	1.54	0.77	10783.6	1.58	0.79
3	3359.32	2.95	0.98	5551.98	3.06	1.02
4	3188.2	3.11	0.78	5120.63	3.32	0.83
6	2084.96	4.76	0.79	3428.16	4.96	0.83
8	1535.64	6.46	0.81	2550.48	6.67	0.83
12	1050.39	9.44	0.79	1744	9.75	0.81
16	795.42	12.47	0.78	1332.45	12.77	0.80
24	561.88	17.65	0.74	940.77	18.08	0.75
32	435.61	22.76	0.71	720.302	23.62	0.74
48	292.2	33.94	0.71	484.13	35.14	0.73
96	230.17	43.08	0.45	386	44.07	0.46

Table 1. Results obtained on *Boomer*.

In Table 1 we present results of experiments executed on *Boomer*. The parallel efficiency on up to 48 processors is above 50%, which confirms our general expectations that the proposed approach parallelizes very well. As expected, as the size of the problem increases, parallel efficiency increases as well (for n=96, efficiency of 73% has been observed on 48 processors, and even on 96 processors efficiency of almost 50% was recorded).

	Benchmark 1			*Benchmark 2*		
p	Time	S_p	E_p	Time	S_p	E_p
n=32						
1	45.96			185.71		
2	24.1	1.91	0.95	96.88	1.63	0.96
4	12.35	3.72	0.93	49.19	3.01	0.94
8	6.28	7.32	0.91	25.18	6.06	0.92
16	3.48	13.21	0.83	13.84	10.38	0.84
32	1.7	27.04	0.84	11.2	13.54	0.52
n=48						
1	952.94			1744.28		
2	485.61	1.96	0.98	889.92	1.96	0.98
3	320.64	2.97	0.99	586.61	2.97	0.99
4	247.78	3.85	0.96	422.03	4.13	1.03
6	167.8	5.68	0.95	285.02	6.12	1.02
8	125.58	7.59	0.95	229.44	7.60	0.95
12	83.97	11.35	0.95	154.17	11.31	0.94
16	64.66	14.74	0.92	110.64	15.77	0.99
24	44.33	21.50	0.90	80.8	21.59	0.90
n=64						
1	2566.04			4247.67		
2	1336.46	1.92	0.96	2212.14	1.92	0.96
4	663.91	3.87	0.97	1099.84	3.86	0.97
8	339.25	7.56	0.95	559.704	7.59	0.95
16	175.32	14.64	0.91	289.24	14.69	0.92
32	95.57	26.85	0.84	158.3	26.83	0.84
n=96						
1	15257.2			25000		
2	7826.49	1.95	0.97	12976.8	1.93	0.96
3	5036.91	3.03	1.01	8384.87	2.98	0.99
4	3879.16	3.93	0.98	6450.12	3.88	0.97
6	2610.47	5.84	0.97	4336.2	5.77	0.96
8	1970.25	6.46	0.97	3279.75	7.62	0.95
12	1321.53	9.44	0.96	2198.85	11.37	0.95
16	1004.52	12.47	0.95	1671.92	14.95	0.93
24	681.54	17.65	0.93	1130.18	22.12	0.92
32	516.2	22.76	0.92	867.62	28.81	0.90

Table 2. Results obtained on *Shooner*.

The super-linear speed-up was been observed in some of the runs (for 3 or 4 processors). There are two reasons for this fact: (1) splitting the entire problem into subproblems of what could be considered an "optimal" size for the computer architecture in question and (2) following from it better usage of cache memories of individual parallel processors.

Table 2 shows results obtained on *Shooner*. These results are quite interesting. Here we can "compare" execution time on (and thus performance of) a 2.0 GHz P4 processor (32-bit architecture) with that of the 1.0 GHz Itanium processor (64-bit architecture). The execution time on *Schooner* is substantially longer than on *Boomer*. This indicates that the very fact that Itanium processors are a 64-bit architecture and, overall, were introduced with promise of higher performance does not compensate for a 50% drop in clock speed. This also seems to indicate that while tools like compilers have been around long enough to optimize performance of 32-bit processors (in the case of a realistic application, rather than a synthetic benchmark), their 64-bit counterparts still have long way to go to reach the same level of code optimization.

	Benchmark 1			*Benchmark 2*		
p	Time	S_p	E_p	Time	S_p	E_p
n=32						
1	117.37			480.55		
2	60.26	1.95	0.97	246.89	1.95	0.97
4	30.34	3.87	0.97	123.32	3.90	0.97
8	15.38	7.63	0.95	62.63	7.67	0.96
16	8.42	13.94	0.87	44.18	10.88	0.68
n=48						
1	2520.68			4315.42		
2	1273.73	1.98	0.99	2321.31	1.86	0.93
3	835.88	3.02	1.01	1444.70	2.99	1.00
4	638.68	3.95	0.99	1170.25	3.69	0.92
6	445.49	5.66	0.94	813.99	5.30	0.88
8	318.97	7.90	0.99	584.28	7.39	0.92
12	215.70	11.69	0.97	368.12	11.72	0.98
n=64						
1	6885.60			11216.40		
2	3345.32	2.06	1.03	5648.49	1.99	0.99
4	1689.63	4.08	1.02	2706.94	4.14	1.04
8	867.27	7.94	0.99	1437.97	7.80	0.98
16	664.12	10.37	0.65	1081.92	10.37	0.65
n=96						
1	42632.1			68353.40		
2	20613.8	2.07	1.03	34507.20	1.98	0.99
3	14140.6	3.01	1.01	22984.70	2.97	0.99
4	10342.3	4.12	1.03	17066.10	4.01	1.00
6	7140.59	5.97	1.00	11652.60	5.87	0.98
8	5354.38	7.96	1.00	8793.15	7.77	0.97
12	3570.23	11.94	1.00	5935.70	11.52	0.96

Table 3. Results obtained on *Azusa*.

Furthermore, when comparing the results obtained on *Boomer* and *Shooner* one can see that, in general, parallel efficiency is substantially higher on *Shooner*. In this context let us note that, overall, it is easier to obtain high level of efficiency on a machine with relatively slower processors. However, efficiency on *Shooner* is extremely good – note that in the case of n=96 it stays above 90% for all numbers of processors used. This can be further explained by the fact that the network parameters start-up time and time for transferring of single word are smaller on the second cluster. Again one can see that the speed-up and efficiency increase with the increase in the size of the discrete problem.

Two cases of super-linear speed-up (for n=48 and 3 and 4 processors) can be explained by the problem size division hitting a sweet-spot of memory usage per processor and its interaction with local cache memory size.

In Table 3 we present results of experiments executed on the NEC Server Azusa Express5800/1160Xa. This machine is an SMP (Symmetric Multiprocessor) and thus it is a very different architecture that the previously considered two – classical cluster computers.

The first important observation is that the execution time on *Azusa* is substantially longer than in the case even of *Shooner*. This reduction in speed cannot be simply explained by 20% reduction in clock speed of the processor. The results on the Azusa were run about a year ago and we used a current for that time version of the NEC compiler. This being the case, combined with the fact that the Itanium technology is very new can explain some inefficiencies of the compiler. Furthermore, since we did not have any direct support and thus we used the simplest possible optimizations.

While the run-time on *Azusa* is much longer than on *Shooner*, its parallel efficiency is higher and a super-linear speed-up is observed very often. While this could be explained by the fact that its processors are even slower, we have to keep in mind that *Azusa* not only has slower processors, but also appropriately older remaining components of the system (e.g. the *Azusa* has SDRAM, while *Shooner* has DDR RAM memory). We would like to conjecture that what matters more is the fact that *Azusa* is an SMP computer and thus its network parameters are substantially better (smaller data access and transfer time) than in the case of a network-based cluster. It is the SMP architecture that is responsible for the extremely good efficiency of 96% when running the problem of size n=96 on 32 processors.

Table 4 shows results obtained on Cray Opteron cluster (*Strider*). The results are very similar to the results obtained on *Boomer*, but there are some interesting differences.

	Benchmark 1			*Benchmark 2*		
p	Time	S_p	E_p	Time	S_p	E_p
n=32						
1	24.29			100.67		
2	12.58	1.93	0.97	51.40	1.96	0.98
4	6.598	3.68	0.92	26.99	3.73	0.93
8	3.44	7.05	0.88	13.94	7.22	0.90
16	1.85	13.16	0.82	7.49	13.45	0.84
32	1.275	19.06	0.60	5.25	19.17	0.60
n=48						
1	666.45			1143.17		
2	331.12	2.01	1.01	607.57	1.88	0.94
3	226.48	2.94	0.98	417.84	2.74	0.91
4	171.12	3.89	0.97	313.39	3.65	0.91
6	118.17	5.64	0.94	216.00	5.29	0.88
8	90.01	7.40	0.93	164.82	6.94	0.87
12	59.80	11.14	0.93	109.18	10.47	0.87
16	45.39	14.68	0.92	77.30	14.79	0.92
24	30.990	21.51	0.90	57.07	20.03	0.84
48	18.198	36.62	0.76	33.35	34.28	0.71
n=64						
1	1533.96			2533.75		
2	756.18	2.03	1.01	1250.47	2.03	1.01
4	391.53	3.92	0.98	648.45	3.91	0.98
8	202.50	7.58	0.95	337.85	7.50	0.94
16	112.74	13.61	0.85	182.89	13.85	0.87
32	56.13	27.33	0.85	92.92	27.27	0.85
64	32.26	47.55	0.74	53.15	47.67	0.75
n=96						
1	11101.9			18430.80		
2	5541.60	2.00	1.00	9204.99	2.00	1.00
3	3722.11	2.98	0.99	6210.83	2.97	0.99
4	2800.50	3.96	0.99	4665.51	3.95	0.99
6	1913.57	5.80	0.97	3182.44	5.79	0.97
8	1435.91	7.73	0.97	2393.06	7.70	0.96
12	976.18	11.37	0.95	1618.29	11.39	0.95
16	741.59	14.97	0.94	1238.29	14.88	0.93
24	515.44	21.54	0.90	862.23	21.38	0.89
32	388.67	28.56	0.89	649.49	28.38	0.89
48	270.01	41.12	0.86	451.46	40.82	0.85
96	149.26	74.38	0.78	246.16	74.87	0.78

Table 4. Results obtained on *Strider*.

Let us consider the largest problem studied (Benchmark 2; n=96). Here, for a single processor *Boomer* is faster (execution time of 17011 second, while *Strider* time is 18430 seconds – the difference of about 8%). For 96 processors situation is exactly the opposite. Here, *Boomer* time is 386 seconds, while *Strider* time is only 246 seconds (difference of about 36%). This result is quit rather unexpected. It shows, that in the case of the real-life problem considered here – when code is executed on a single processor – the two 2.0 GHz processors from Intel and AMD behave very similarly (performance difference of 8% can be associated with compiler quality and options used during compilation), the big difference is in parallel performance. Let us stress that both systems use "the same" Myrinet 2000 switch and have a very similar total number of computational nodes. Thus the only way to explain the difference seems to be by pointing to the computer vendor. Cray surely knows supercomputers and it shows in this case through a 36% performance gain.

Finally, we compare results reported here with earlier results collected on a Sun Sunfire 6800. In Figure 1 we depict parallel speed-up of execution of a single iteration of our code obtained on different parallel systems in the case of *Benchmark* 2, for n=96.

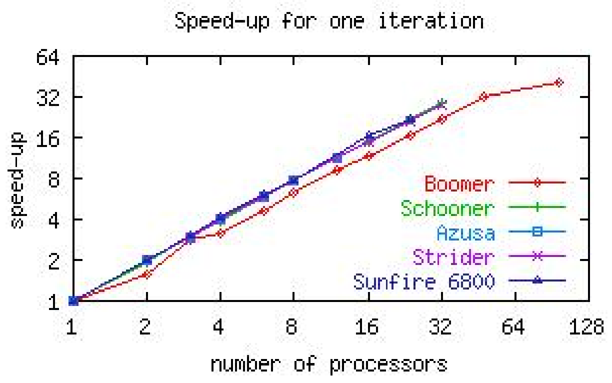

Fig.1 Speed-up for one iteration

What is particularly revealing is the fact that most systems have very similar speed-up. However *Boomer*, definitely lags behind and this is related to the fact that it is the fastest of the machines we experimented with when a single-processor performance is considered, but when it comes to parallel performance (and thus issues related to data transmission), then even though it is utilizing a very good switching infrastructure, its performance falls behind.

7. Concluding remarks

In this note we have used a real-life geomechanics modeling code to compare performance of four parallel computers (2 clusters and one SMP). The main finding is that the general behavior of the code on all machines is very good and in agreement with theoretical estimates. At the same time we were able to make a few interesting observations about each of computers used in our experiments.

Acknowledgments

This research was supported by the bilateral IPP–BAS – ICS–WSSP interacademy exchange grant "Parallel and Distributed Computing Practices". It was supported also by the Bulgarian NSF grant I–1402/2004. Computer time grant from the Oklahoma Supercomputing Center is kindly acknowledged. The results on Azusa and Strider were obtained during the HPC-Europa research visit at the University of Stuttgart supported by the European Commission through grant number R113–2003–506079.

References

[1] I. Lirkov, MPI solver for 3D elasticity problems, Mathematics and computers in simulation, 61 3–6, 2003, 509–516.

[2] A. Georgiev, A. Baltov, and S. Margenov, Hipergeos benchmark problems related to bridge engineering applications, project report HG CP 940820MOST4.

[3] I. Lirkov, Parallel Performance of an MPI Solver for 3D Elasticity Problems, Numerical methods and applications, (I. Dimov, I. Lirkov, S. Margenov, Z. Zlatev eds.), Lecture Notes in Computer Sciences, 2542, Springer Verlag, 2003, 527–535.

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

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

[6] I. Lirkov, S. Margenov, MPI parallel implementation of CBF preconditioning for 3D elasticity problems, Mathematics and computers in simulation, 50 1–4, 1999, 247–254.

[7] 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.

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

[9] http://www.oscer.ou.edu/resources.php#boomer

[10] http://www.oscer.ou.edu/resources.php#schooner

[11] http://www.hlrs.de/hw-access/platforms/azusa/

[12] http://www.hlrs.de/hw-access/platforms/strider/