High Performance Solver for 3D Elasticity Problems

Ivan Lirkov[1], Yavor Vutov[1], Marcin Paprzycki[2], and Maria Ganzha[3]

[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] Institute of Computer Science, Warsaw School of Social Psychology, ul.
Chodakowska 19/31, 03–815 Warszawa, Poland,
marcin.paprzycki@swps.edu.pl http://mpaprzycki.swps.edu.pl
[3] Systems Research Institute, Polish Academy of Science,
ul. Newelska 6, 01-447 Warszawa, Poland
maria.ganzha@ibspan.waw.pl http://ganzha.euh-e.edu.pl

Abstract. In this paper we consider numerical solution of 3D linear elasticity equations 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 utilizing MPI for communication is implemented. Results of numerical tests performed on a number of modern parallel computers using real life engineering problems from the geosciences (geomechanics in particular) are reported and discussed.

1 Introduction

Our work concerns development and implementation of efficient parallel algorithms for solving elasticity problems arising in geosciences. Typical application problems include simulations of foundations of engineering constructions (which transfer and distribute the total loading into the 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.

These engineering problems are 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/nonlinear equations $K\mathbf{x} = \mathbf{f}$, where the stiffness matrix K is large, sparse and symmetric positive definite. The Conjugate Gradient (CG) type methods are recognized as the most cost-effective way to solve problems of this type [1]. To accelerate the iteration convergence a preconditioner M is combined with the CG algorithm. To make a reliable prediction of the construction safety, which is sensitive to soil deformations, a very accurate model and a large system of sparse linear equations is required. In the real-life applications, such system can be very 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 the numerical solution of linear systems is a fundamental operation in solving elasticity problems. Specifically, solving these linear systems is usually very time-consuming (requiring up to 90% of the total solution time). Hence, developing fast algorithms for solving linear equations is essential. Furthermore, such algorithms can 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.

2 Elasticity Problems

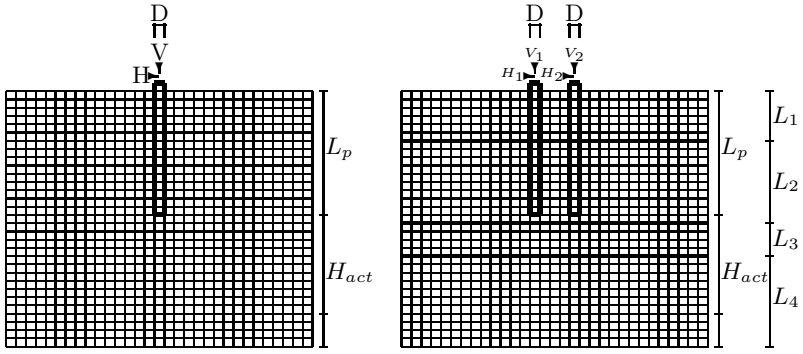
For simplicity, in this work we focus our attention on 3D linear elasticity problems following two basic assumptions: (1) displacements are small, and (2) material properties are isotropic. A precise mathematical formulation of the considered problem is described in [5]; the 3D elasticity problem in 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 parallelogram $\Omega = [0, x_1^{max}] \times [0, x_2^{max}] \times [0, x_3^{max}]$, where the boundary conditions on each wall of Ω are of fixed type.

Benchmark problems from [4] are used in numerical tests reported here. The engineering problems are as follows: a) single pile in a homogeneous sandy clay soil (see Fig. 1(a)) and b) two piles in an inhomogeneous sandy clay soil (Fig. 1(b)). 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 substantial body of work dealing with preconditioning of iterative solution methods for elasticity systems discretized using the Finite Element



(a) Problem 1; Cross section of the computational domain Ω. $E_{soil} = 10MPa$, $\nu_{soil} = 0.3$, $E_{pile} = 31500MPa$, $\nu_{pile} = 0.2$

(b) Problem 2; Cross section of the computational domain Ω. $E_{L_1} = 5.2MPa$, $\nu_{L_1} = 0.4$, $E_{L_2} = 9.4MPa$, $\nu_{L_2} = 0.35$, $E_{L_3} = 14.0MPa$, $\nu_{L_3} = 0.25$, $E_{L_4} = 21.4MPa$, $\nu_{L_4} = 0.2$.

Fig. 1. Benchmark problems

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 displacement decomposition (see, e.g., [3]). In [6] circulant block-factorization is used for preconditioning of the obtained block-diagonal matrix and a displacement decomposition circulant block factorization 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 [5, 6]. Moreover DD CBF solver has a good parallel efficiency (see, e.g., [5, 6]).

4 Benchmarking Performance

To solve the above described problems, a portable parallel FEM code was designed and implemented in C, while the parallelization has been facilitated using the MPI library [7, 8]. The parallel code has been tested on cluster computers located in the National Energy Research Scientific Computing Center (NERSC), Oklahoma Supercomputing Center (OSCER), and in Bologna, Italy (CINECA). In our experiments, times have been collected using the MPI provided timer and report the best results from multiple runs. We report the elapsed time T_p in seconds on p processors, the speed-up $S_p = T_1/T_p$, and the parallel efficiency $E_p = S_p/p$. For the benchmark problems described in Section 2, we used discretization with $n_1 = n_2 = n_3 = n$ where $n = 32, 48, 64$, and 96, while sizes of discrete problems were $3n^3$.

Table 1. Experimental results on Jacquard.

		Benchmark 1			Benchmark 2				Benchmark 1			Benchmark 2		
p	n	T_p	S_p	E_p	T_p	S_p	E_p	n	T_p	S_p	E_p	T_p	S_p	E_p
1	32	12.5			50.4			64	812.9			1277.8		
2		6.6	1.90	0.950	25.7	1.96	0.981		416.8	1.95	0.975	675.2	1.89	0.946
4		3.5	3.55	0.886	14.3	3.53	0.883		217.2	3.74	0.936	351.3	3.64	0.909
8		1.8	6.81	0.852	7.4	6.78	0.848		111.7	7.28	0.910	185.1	6.90	0.863
16		1.2	10.64	0.665	4.7	10.82	0.676		56.4	14.42	0.901	92.7	13.79	0.862
32		0.8	15.93	0.498	3.1	16.28	0.509		35.2	23.11	0.722	57.7	22.16	0.692
64									25.4	32.01	0.500	43.1	29.68	0.464
1	48	326.5			608.7			96	5259.8			8702.3		
2		165.9	1.97	0.984	303.1	2.01	1.004		2704.7	1.94	0.972	4503.9	1.93	0.966
3		115.1	2.84	0.946	212.2	2.87	0.956		1833.1	2.87	0.956	3083.6	2.82	0.941
4		87.0	3.75	0.939	158.3	3.85	0.961		1388.3	3.79	0.947	2331.9	3.73	0.933
6		59.3	5.51	0.918	107.9	5.64	0.940		952.7	5.52	0.920	1588.3	5.48	0.913
8		44.2	7.39	0.924	80.5	7.56	0.945		714.8	7.36	0.920	1188.9	7.32	0.915
12		30.1	10.85	0.904	54.9	11.09	0.924		480.3	10.95	0.913	796.3	10.93	0.911
16		25.9	12.62	0.789	47.0	12.96	0.810		358.1	14.69	0.918	590.5	14.74	0.921
24		17.7	18.47	0.769	32.1	18.94	0.789		240.1	21.91	0.913	399.9	21.76	0.907
32									182.8	28.77	0.899	299.8	29.03	0.907
48		12.5	26.21	0.546	23.6	25.80	0.537		177.2	29.69	0.618	293.0	29.70	0.619
96									140.0	37.58	0.391	231.5	37.58	0.392

In Table 1 we present results of experiments performed on Jacquard (see http://www.nersc.gov/nusers/resources/jacquard/). It is a 712-CPU (356 dual-processor nodes) Opteron Linux cluster. Each processor runs at 2.2 GHz, and has a theoretical peak performance of 4.4 GFlop/s. Processors on each node share 6 GB of memory. The nodes are interconnected with a high-speed InfiniBand network. Shared file storage is provided by a GPFS file system. We have used the ACML Optimized Math Library and compiled the code using "mpicc -Ofast $ACML" command. The "-Ofast" option is a generic option leading to vendor suggested aggressive optimization.

As expected, parallel efficiency improves with the size of the discrete problems. For the largest problems in this set of experiments ($n = 96$), parallel efficiency is above 90% on up to 32 processors which confirms our general expectations that the proposed approach parallelizes very well.

Table 2 shows execution time on Topdawg. It is Dell Pentium4 Xeon64 Linux cluster (see http://www.oscer.ou.edu/resources.php). It has 512 dual-processor nodes. Each processor runs at 3.2 GHz and has a theoretical peak performance of 6,4 GFlop/s. Processors within each node share 4 GB of memory, while nodes are interconnected with a high-speed InfiniBand network. We have used Intel C compiler and compiled the code with the following options: "-O3 -parallel -ipo -tpp7 -xP" (collection of options for aggressive optimization suggested by Henry Neeman of OSCER).

The execution time on Topdawg is substantially smaller than that on Jacquard (in computations that are primarily floating point arithmetic, Xeon64

Table 2. Experimental results on Topdawg.

		Benchmark 1			Benchmark 2				Benchmark 1			Benchmark 2		
p	n	T_p	S_p	E_p	T_p	S_p	E_p	n	T_p	S_p	E_p	T_p	S_p	E_p
1	32	10.0			38.0			64	536.0			852.0		
2		6.0	1.67	0.83	22.0	1.73	0.86		359.0	1.49	0.75	592.0	1.44	0.72
4		3.1	3.23	0.81	11.0	3.45	0.86		180.0	2.98	0.74	293.0	2.91	0.73
8		1.8	5.56	0.69	6.3	6.03	0.75		82.0	6.54	0.82	236.0	3.61	0.45
16		1.2	8.47	0.53	3.9	9.84	0.62		44.0	12.18	0.76	71.0	12.00	0.75
32		1.1	9.09	0.28	3.5	10.86	0.34		24.0	22.33	0.70	39.0	21.85	0.68
64									18.0	29.78	0.47	29.0	29.38	0.46
1	48	244.0			444.0			96	4074.0			6766.0		
2		146.0	1.67	0.84	267.0	1.66	0.83		2353.0	1.73	0.87	3817.0	1.77	0.89
3		96.0	2.54	0.85	177.0	2.51	0.84		1538.0	2.65	0.88	2557.0	2.65	0.88
4		71.0	3.44	0.86	131.0	3.39	0.85		1207.0	3.38	0.84	1996.0	3.39	0.85
6		46.0	5.30	0.88	84.0	5.29	0.88		805.0	5.06	0.84	1344.0	5.03	0.84
8		34.0	7.18	0.90	62.0	7.16	0.90		602.0	6.77	0.85	999.0	6.77	0.85
12		24.0	10.17	0.85	41.0	10.83	0.90		406.0	10.03	0.84	675.0	10.02	0.84
16		19.0	12.84	0.80	34.0	13.06	0.82		307.0	13.27	0.83	509.0	13.29	0.83
24		13.0	18.77	0.78	24.0	18.50	0.77		207.0	19.68	0.82	343.0	19.73	0.82
32									158.0	25.78	0.81	262.0	25.82	0.81
48		9.7	25.15	0.52	18.0	24.67	0.51		115.0	35.43	0.74	190.0	35.61	0.74
96									70.0	58.20	0.61	115.0	58.83	0.61

processors running at 3.2 GHz are more efficient than Opteron processors at 2.2 GHz; which can be also seen comparing their theoretical peak performance). The communication time on both clusters is approximately the same (they both use InfiniBan network) and this is one of the reasons for higher parallel efficiency of Jacquard (slower processors combined with equally fast network). Again, parallel efficiency increases with the size of the discrete problems and for the largest problems reaches 60% on 96 processors.

Table 3 contains execution times collected on an IBM Linux Cluster 1350 made of 512 2-way IBM X335 nodes. Each computing node contains 2 Xeon Pentium IV processors running at 3 GHz and 2 GB of RAM. Nodes are interconnected via a Myrinet network with a maximum bandwidth of 256 Mb/s. We have used IBM Visual Age compiler and a "-O3" option.

The execution time on one processor is larger than the results from earlier mentioned computer systems. While the run-time on IBM Linux cluster is much longer than on Jacquard and Topdawg, its parallel efficiency is higher — it is higher than 50% for full set of experiments reported here. This indicates that the decrease in processor speed offsets the slower interconnection network.

Finally, Table 4 reports execution times collected on an IBM SP Cluster 1600 made of 64 nodes p5-575 (see http://www.ibm.com/servers/eserver/pseries/library/sp_books/). A p5-575 node contains 8 IBM Power5 processors running at 1.9 GHz and has 16 GB of RAM. Nodes are interconnected with a pair of connections to the Federation High Performance Switch (HPS). The HPS interconnect is capable of a unidirectional bandwidth of up to 2 Gb/s. We have used the

Table 3. Experimental results for the IBM Linux Cluster.

		Benchmark 1			Benchmark 2				Benchmark 1			Benchmark 2		
p	n	T_p	S_p	E_p	T_p	S_p	E_p	n	T_p	S_p	E_p	T_p	S_p	E_p
1	32	22.7			90.3			64	1384.1			2232.3		
2		12.5	1.81	0.906	49.5	1.82	0.911		730.2	1.90	0.948	1195.9	1.87	0.933
4		6.5	3.50	0.876	25.7	3.51	0.877		393.3	3.52	0.880	633.5	3.52	0.881
8		3.4	6.75	0.843	13.2	6.84	0.855		208.8	6.63	0.829	339.6	6.57	0.822
16		1.9	12.03	0.752	7.3	12.33	0.771		99.0	13.99	0.874	164.9	13.54	0.846
32		1.4	16.02	0.501	5.6	16.21	0.507		54.1	25.59	0.800	86.5	25.80	0.806
64									33.6	41.20	0.644	54.5	40.96	0.640
1	48	600.6			1104.2			96	10080.4			17648.8		
2		323.6	1.86	0.928	594.3	1.86	0.929		5401.3	1.87	0.933	8953.1	1.97	0.986
3		220.2	2.73	0.909	399.0	2.77	0.922		3654.3	2.76	0.919	6061.5	2.91	0.971
4		168.4	3.57	0.892	311.0	3.55	0.888		2794.2	3.61	0.902	4633.8	3.81	0.952
6		115.8	5.19	0.864	214.2	5.15	0.859		1900.2	5.30	0.884	3158.8	5.59	0.931
8		84.7	7.09	0.887	155.6	7.10	0.887		1454.9	6.93	0.866	2415.8	7.31	0.913
12		57.5	10.44	0.870	105.1	10.50	0.875		972.7	10.36	0.864	1604.4	11.00	0.917
16		43.7	13.75	0.860	80.2	13.78	0.861		754.2	13.37	0.835	1249.0	14.13	0.883
24		30.2	19.86	0.827	55.5	19.91	0.830		477.2	21.13	0.880	793.5	22.24	0.927
32									355.7	28.34	0.886	589.3	29.95	0.936
48		18.9	31.72	0.661	35.1	31.46	0.655		248.0	40.65	0.847	411.8	42.86	0.893
96									151.3	66.64	0.694	251.8	70.08	0.730

IBM Visual Age compiler and compiled the code using "-O4 -qipa=inline" options. One can see that for relatively large problems the speed-up is close to the theoretical limit — the number of processors. This result was expected because communications between processors is not only very fast, but also its start-up time is faster than in the case of other machines. Interestingly, a super-linear speed-up is observed in some cases. The main reasons for this fact can be related to splitting the entire problem into subproblems which helps memory management in the case of 8 processor nodes; in particular allows for better usage of cache memories of individual parallel processors. Interestingly, this machine is only slightly faster that the IBM Linux Cluster, bu remains slower than the first two clusters. This seems also to show the age of this machine, which is the oldest of the four.

A comparison of parallel performance of the developed C+MPI code obtained on all four above mentioned computer systems can be seen in Figures 2 and 3. In Figure 2 we depict the execution time of a single PCG iteration of our code (computer as an average of the total time divided by the number of iterations), while in Figure 3 we represent parallel speed-up of a single iteration. What is particularly revealing is the fact that all four systems have very similar speed-up.

However, the fact that the largest speed-up was obtained on the IBS SP machine indicates that as far as large clusters are concerned it is till the processing power that is winning the race with the network throughput. It is much easier to solve problems fast on a single processor than build a well-balanced parallel computer.

Table 4. Experimental results for the IBM SP cluster.

		Benchmark 1			Benchmark 2				Benchmark 1			Benchmark 2		
p	n	T_p	S_p	E_p	T_p	S_p	E_p	n	T_p	S_p	E_p	T_p	S_p	E_p
1	32	21.8			86.8			64	1257.8			2056.8		
2		10.7	2.03	1.015	43.1	2.01	1.007		670.2	1.88	0.938	989.9	2.08	1.039
4		5.4	4.03	1.006	21.1	4.11	1.027		313.0	4.02	1.005	527.4	3.90	0.975
8		2.7	8.04	1.005	10.6	8.22	1.027		152.1	8.27	1.034	252.4	8.15	1.019
16		1.5	15.01	0.938	5.9	14.70	0.919		76.6	16.43	1.027	126.2	16.29	1.018
32		1.0	21.68	0.677	3.1	28.18	0.881		39.3	31.98	0.999	65.1	31.58	0.987
64									21.0	60.01	0.938	34.2	60.19	0.940
1	48	541.7			993.6			96	9100.5			12338.7		
2		278.2	1.95	0.974	500.7	1.98	0.992		4501.5	2.02	1.011	6771.2	1.82	0.911
3		182.4	2.97	0.990	337.5	2.94	0.981		3001.4	3.03	1.011	3988.2	3.09	1.031
4		137.9	3.93	0.982	252.7	3.93	0.983		2313.5	3.93	0.983	2982.7	4.14	1.034
6		90.1	6.01	1.002	159.3	6.24	1.039		1477.4	6.16	1.027	1961.8	6.29	1.048
8		67.3	8.05	1.006	122.9	8.08	1.010		1095.2	8.31	1.039	1473.9	8.37	1.046
12		45.1	12.00	1.000	82.4	12.06	1.005		740.3	12.29	1.024	1016.5	12.14	1.012
16		34.2	15.84	0.990	58.6	16.96	1.060		560.5	16.24	1.015	774.3	15.94	0.996
24		24.2	22.35	0.931	43.8	22.67	0.945		382.2	23.81	0.992	512.6	24.07	1.003
32									283.6	32.09	1.003	383.5	32.17	1.005
48		12.4	43.78	0.912	21.2	46.79	0.975		193.8	46.96	0.978	258.0	47.82	0.996
96									100.9	90.20	0.940	146.5	84.21	0.877

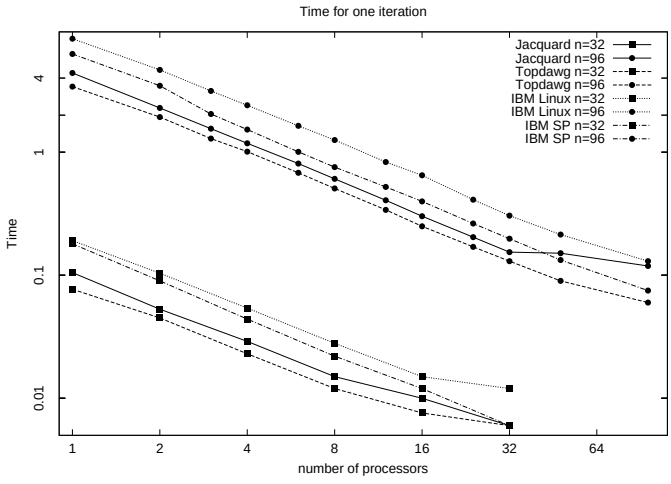


Fig. 2. Time for one iteration on the parallel computer systems

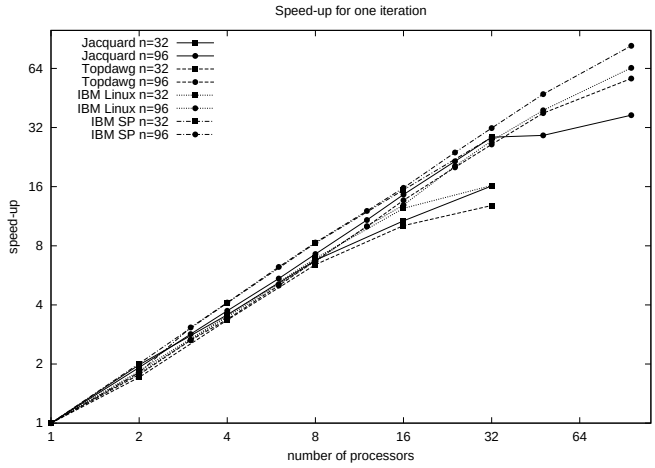


Fig. 3. Speed-up for one iteration on the parallel computer systems

Acknowledgments

Computer time grants from the National Energy Research Scientific Computing Center and the Oklahoma Supercomputing Center are kindly acknowledged. The parallel numerical tests on two clusters in Bologna were supported via the EC Project HPC-EUROPA RII3-CT-2003-506079. This research was partially supported by grant I-1402/2004 from the Bulgarian NSF and by the project BIS-21++ funded by FP6 INCO grant 016639/2005. Work presented here is a part of the Poland-Bulgaria collaborative grant: "Parallel and distributed computing practices".

References

1. O. Axelsson, *Iterative solution methods*, Cambridge Univ. 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**, 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, and 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. 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.
8. D. Walker and J. Dongara, MPI: a standard Message Passing Interface, *Supercomputer* **63**, 1996, 56–68.