Performance Evaluation of MPI Implementations and MPI Based Parallel ELLPACK Solvers

S. Markus, S.B. Kim, K. Pantazopoulos, A.L. Ocken,
E.N. Houstis, P. Wu and S. Weerawarana
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA.

D. Maharry
Department of Mathematics and Computer Science
Wabash College
Crawfordsville, IN 47933, USA.

Abstract

In this study, we are concerned with the parallelization of finite element mesh generation and its decomposition, and the parallel solution of sparse algebraic equations which are obtained from the parallel discretization of second order elliptic partial differential equations (PDEs) using finite difference and finite element techniques. For this we use the Parallel ELLPACK (//ELLPACK) problem solving environment (PSE) which supports PDE computations on several MIMD platforms. We have considered the ITPACK library of stationary iterative solvers which we have parallelized and integrated into the //ELLPACK PSE. This Parallel ITPACK package has been implemented using the MPI, PVM, PICL, PARMACS, nCUBE Vertex and Intel NX message passing communication libraries. It performs very efficiently on a variety of hardware and communication platforms. To study the efficiency of three MPI library implementations, the performance of the Parallel ITPACK solvers was measured on several distributed memory architectures and on clusters of workstations for a testbed of elliptic boundary value PDE problems. We present a comparison of these MPI library implementations with PVM and the native communication libraries, based on their performance on these tests. Moreover we have implemented in MPI, a parallel mesh generator that concurrently produces a semi–optimal partitioning of the mesh to support various domain decomposition solution strategies across the above platforms. The results indicate that the MPI overhead varies among the various implementations without significantly affecting the algorithmic speedup even on clusters of workstations.

1 Introduction

Computational models based on partial differential equation (PDE) mathematical models have been successfully applied to study many physical phenomena. The overall quantitative and qualitative accuracy of these computational models in representing the physical situations or artifacts that they are supposed to simulate, depends very much on the computer resources available. The recent advances in high performance computing technologies have provided an opportunity to significantly speed up these computational models and dramatically increase their numerical resolution and complexity. In this paper, we focus on the parallelization of PDE computations based on the message passing paradigm in high performance distributed memory environments.

We use the Parallel ELLPACK (//ELLPACK) PDE computing environment to solve PDE models consisting of a PDE equation ($Lu = f$) defined on some domain Ω and subject to some auxiliary condition ($Bu = g$) on the boundary of Ω (= $\partial\Omega$). This continuous PDE problem is reduced to a distributed sparse system of linear equations using a parallel finite difference or finite element discretizer and solved using a parallel iterative linear solver. We compare the performance of these parallel PDE solvers on different hardware platforms using native and portable message passing communication systems. In particular, we evaluate the performance of three implementations of the portable Message Passing Interface (MPI) standard in solving a testbed of PDE problems within the //ELLPACK environment.

In [4] the authors study the performance of four different public domain MPI implementations on a cluster of

DEC Alpha workstations connected by a 100Mbps DEC GIGAswitch using three custom developed benchmarking programs (ping, ping–pong and collective). In [14] the authors study the performance of MPI and PVM on homogeneous and heterogeneous networks of workstations using two benchmarking programs (ping and ping–pong). While such analyses are important, we believe that the effective performance of an MPI library implementation can be best measured by benchmarking application libraries which are in practical use. In this work we report the performance of MPI library implementations using the Parallel ITPACK (//ITPACK) iterative solver package in //ELLPACK. We also evaluate the performance of a parallel finite element mesh generator and decomposition library which was implemented using MPI in the //ELLPACK system.

This paper is organized as follows. In the next section we describe the //ELLPACK problem solving environment (PSE), which is the context in which this work was done. In section 3 we present the PDE problem that was used in our tests and explain the parallel computations that were measured. In section 4 we present the experimental performance results and analyze them. Finally, in section 5 we present our conclusions.

2 //ELLPACK PSE

//ELLPACK [15] is a problem solving environment for solving PDE problems on high performance computing platforms as well as a development environment for building new PDE solvers or PDE solver components. //ELLPACK allows the user to (symbolically) specify partial differential equation problems, specify the solution algorithms to be applied, solve the problem and finally analyze the results produced. The problem and solution algorithm are specified in a custom high level language through a complete graphical editing environment. The user interface and programming environment of //ELLPACK is independent of the targeted machine architecture and its native programming environment.

The //ELLPACK PSE is supported by a parallel library of PDE modules for the numerical simulation of stationary and time dependent PDE models on two and three dimensional regions. A number of well known "foreign" PDE systems have been integrated in the //ELLPACK environment including VECFEM, FIDISOL, CADSOL, VERSE, and PDECOL. //ELLPACK can simulate structural mechanics, semi–conductor, heat transfer, flow, electromagnetic, microelectronics, ocean circulation, bio–separation, and many other scientific and engineering phenomenon.

The parallel PDE solver libraries are based on the "divide and conquer" computational paradigm and utilize the discrete domain decomposition approach for problem partitioning and load balancing [11]. A number of tools and libraries exist in the //ELLPACK environment to support this approach and estimate (specify) its parameters. These include sequential and parallel finite element mesh generators, automatic (heuristic) domain decomposers, finite element and finite difference modules for discretizing elliptic PDEs, and parallel implementations of the ITPACK [12] linear solver library. The parallel libraries have been implemented in both the host–node and hostless programming models using several portable message passing communication libraries and native communication systems.

3 Benchmark Application and Libraries

We use the //ELLPACK system to compare the performance of different implementations of the Parallel ITPACK (//ITPACK) [10] sparse iterative solver package in solving sparse systems arising from finite difference PDE approximations. We also use the //ELLPACK system to evaluate the performance of an MPI–based parallel mesh generator and decomposer.

3.1 Benchmarked PDE Problem

The //ITPACK performance data presented in this paper are for the Helmholtz–type PDE problem,

$$u_{xx} + u_{yy} - [100 + \cos(2\pi x) + \sin(3\pi y)]u = f(x, y) \quad (1)$$

where $f(x, y)$ is chosen so that

$$u(x, y) = -0.31[5.4 - \cos(4\pi x)]\sin(\pi x)(y^2 - y)$$
$$[5.4 - \cos(4\pi y)]$$
$$[(1 + (4(x - 0.5)^2 + 4(y - 0.5)^2)^4)^{-1} - 0.5]$$

exactly satisfies (1) and with Dirichlet boundary conditions (see Figure 1).

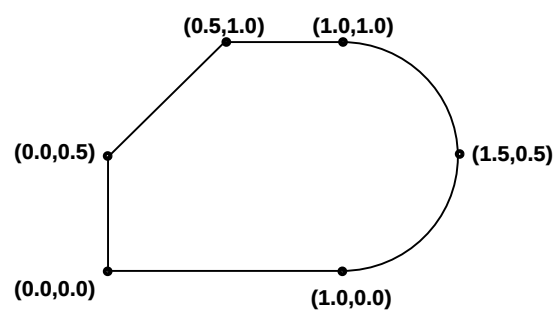

Figure 1. *Domain for the Helmholtz–type boundary value problem with a boundary consisting of lines connecting the points* $(1, 0), (0, 0), (0, 0.5), (0.5, 1)$ *and* $(1, 1)$ *and the half circle* $x = 1 + 0.5\sin(t), y = 0.5 - 0.5\cos(t), t \in [0, \pi]$.

We solve this problem using a parallel 5–point star discretization. The experimental results were generated with 150x150 and 200x200 uniform grids.

Instead of partitioning the grid points optimally, [11] proposed to extend the discrete PDE problem to the rectangular domain that contains the original PDE domain. Identity equations are assigned to the exterior grid points of the rectangular overlaying grid and these artificial equations are uncoupled from the active equations. The modified problem is solved in parallel by partitioning the overlayed rectangular grid in a trivial manner. We refer to this parallel discretization scheme as the *encapsulated 5–point star* method. Numerical results indicate that this approach outperforms all the ones that are based on an optimal grid partitioning [11]. The encapsulated 5–point star discretization on (1) results in a total of 18631 equations for a 150x150 grid and 33290 equations for a 200x200 grid.

3.2 //ITPACK Library

The //ITPACK system is integrated in the //ELLPACK PSE and is applicable to any linear system stored in //ELLPACK's distributed storage scheme. It consists of seven modules implementing SOR, Jacobi-CG, Jacobi-SI, RSCG, RSSI, SSOR-CG and SSOR-SI under different indexing schemes [12]. The interfaces of the parallel modules and the assumed data structures are presented in [11]. The parallel ITPACK library has been proven to be very efficient for elliptic PDEs [10].

Implementation

The code is based on the sequential version of ITPACK which was parallelized by utilizing a subset of level two sparse BLAS routines [11]. Thus the theoretical behavior of the solver modules remain unchanged from the sequential version.

The parallelization is based on the message passing paradigm. The implementation assumes a row–wise splitting of the algebraic equations (obtained indirectly from a non–overlapping decomposition of the PDE domain). Each parallel processor stores a row block of coupled and uncoupled algebraic equations and the requisite communication information, in its local memory. In each sparse solver iteration, a local matrix–vector multiplication is performed. On each processor, this involves the local submatrix A and the values of the local vector u, whose shared components are first updated with data received from the neighboring processors. Inner product computations also occur in each iteration. For this, first the local inner products are computed concurrently. Then these local results are summed up using a global reduction operation (Figure 2).

```
repeat
   .......
   .......
   for i = 1 to no_of_neighbors
      SEND shared components of vector u
      RECEIVE shared components of vector u
   for i = 1 to no_of_equations
      Perform Σ_{j=1}^{no_unknowns} A_ij u_j
   .......
   .......
   Compute local inner product
   GLOBAL REDUCTION to sum local results
   .......
   .......
   Check for convergence
until converged
```

Figure 2. *Communication operations in the core algorithm within the //ITPACK solvers.*

Communication Modules

The communication modules of the parallel ITPACK library have been implemented for several MIMD platforms using different native and portable communication libraries. The implementations utilize standard send/receive, reduction, barrier synchronization and broadcast communication primitives from these message passing communication libraries. No particular machine configuration topology is assumed in the implementation.

Parallel ITPACK implementations are available on the Intel Paragon, Intel iPSC/860 and nCUBE 2 parallel machines, as well as on workstation clusters. It has been implemented for these MIMD platforms using the MPI [8], PVM [5], PICL [6] and PARMACS [9] portable communication libraries as well as the nCUBE 2 VERTEX and Intel NX native communication libraries [3], [13].

3.3 Mesh Generator and Decomposer

The //ELLPACK system contains a natural "fast" alternative for the normally very costly mesh decomposition task [11]. It contains a library that integrates the mesh generation and partitioning steps and implements them in parallel [16]. This methodology is natural since most of the mesh generators already use some form of coarse domain decomposition as a starting point. The parallel library concurrently produces a semi–optimal partitioning of the mesh to support a variety of domain decomposition heuristics for two and three dimensional meshes. It supports both element–wise and node–wise partitionings. This parallel mesh generator and decomposer library has been implemented using MPI. Experimental results show that this parallel integrated approach can result in significant reduction of the data partitioning overhead [17].

Communication Library	Platform			
	P	N	I	W
MPICH v1.0.7	x		x	
MPICH v1.0.12		x		x
CHIMP v2				x
LAM v5.2				x
LAM v6.0				x
Vertex		x		
NX	x			
PICL v2.0	x	x	x	
PVM v3.3	x			x
PARMACS v5.1		x	x	

Table 1. *Communication libraries used on each hardware platform. The P column represents the Intel Paragon, N the nCUBE/2, I the Intel iPSC/860 and W the workstation cluster.*

4 Performance Analysis

4.1 Computing Environments

The experiments for this study were performed on four different hardware platforms: an nCUBE/2, an Intel iPSC/860, an Intel Paragon XP/S 10 and a network of Sun workstations. The nCUBE/2 is a 64–node system with 4MB of memory per node. The Intel iPSC/860 is a 16–node system with 16MB of memory per node. The Intel Paragon XP/S 10 is a 140–node system with 32MB of memory per node. The network of Sun workstations consists of a collection of SparcStation 2/5/10/20s and Sparc IPCs, IPXs and LXs. We treat this as two separate clusters by separating the SS20s running Solaris 2.4 from the other workstations (running SunOS 4.1.3). Henceforth we shall refer to these two clusters as SunOS4–workstation–network and Solaris–workstation–network. The SS20s (Model 61), each with 32MB memory, are connected to a 10Mbps Ethernet. The workstations running SunOS 4.1.3 include 50 MHz LXs each with 40MB memory, 40 MHz SS2s with 24MB to 48MB memory, a two–processor SS10 (Model 512) with 64MB memory, 40 MHz IPXs each with 16MB memory and 25 MHz IPCs with 24MB memory. They are all connected with a 10Mbps Ethernet.

In this study we consider the following public domain MPI standard implementations: MPICH [7], a joint project between Argonne National Labs and Mississippi State University, CHIMP [1], from the Edinburgh Parallel Computing Centre at the University of Edinburgh and LAM [2], from the Ohio Supercomputer Center.

//ITPACK's communication module has been implemented using nCUBE 2 Vertex, Intel NX, MPI (MPICH v1.0.12, MPICH v1.0.7, CHIMP v2, LAM v6.0 and LAM v5.2), PICL v2.0 and PVM v3.3. However, not all of these communication libraries are available on all the hardware platforms. Table 1 indicates the hardware platform and communication library combinations we used for this study.

4.2 Experimental Results

We use the //ITPACK Jacobi CG iterative solver to solve the finite difference equations arising from the encapsulated 5–point star discretization of the benchmark PDE problem on different hardware platform and communication library combinations. A convergence tolerance of $0.5 * 10^{-5}$ was specified as the stopping criterion for the Jacobi CG iterations. The Jacobi CG solver converged in 368 to 371 iterations for the 150x150 grid and in 365 to 169 iterations for the 200x200 grid. An error norm of less than $1.0 * 10^{-3}$ was obtained in the PDE problem solution for all the platforms. The timing data listed in the tables below, reflect the aggregate of the actual CPU usage and communication times and *not* the wall–clock time.

Tables 2, 4 and 3 list the //ITPACK Jacobi CG solver execution times (in seconds) for the benchmark problem, on the Intel Paragon, nCUBE 2 and iPSC/860 parallel platforms with different communication libraries. Since the problem size is fixed across all the processor configurations, the decline in the speedup as the number of processors increase can be mostly attributed to the decrease in computation and increase in communication per processor. This is evidenced by the better speedup obtained in the 200x200 grid problem in comparison with the 150x150 grid problem for the 16, 32 and 64 processor configurations on the Paragon (Table 2). We were unable to run the 200x200 grid problem on the nCUBE 2 machine due to insufficient memory on each node.

The performance measurements show that the MPICH MPI implementation for the Paragon delivers reasonable speedup for the smaller processor configurations (1, 2, 4, 8). The speedup achieved on the iPSC/860 for MPICH (Table 3) is slightly better for the same processor configurations. The speedup obtained for MPICH on the nCUBE 2 platform (Table 4) is clearly the best across all the parallel machines considered, despite its higher overall execution times. The good speedup achieved on the nCUBE 2 is partly because it is a very well balanced machine in terms of processor speed and communication latencies. Both the nCUBE 2 and iPSC/860 have an underlying hypercube interconnection network and the Paragon has a two–dimensional mesh interconnection network. Since the application was not programmed with a specific virtual topology, these performance measurements indicate that in general, MPI based application implementations map onto hypercube interconnection networks in the underlying hardware quite well, with good relative speedup. This is not surprising since hypercube net-

Configuration		150x150	200x200
1	time	80.24	141.41
	speedup	1.00	1.00
2	time	42.07	73.73
	speedup	1.91	1.92
4	time	25.09	43.52
	speedup	3.20	3.25
8	time	14.29	23.75
	speedup	5.61	5.95
16	time	8.85	13.59
	speedup	9.06	10.41
32	time	6.38	8.66
	speedup	12.57	16.33
64	time	7.59	6.60
	speedup	10.57	21.43

Table 2. *Performance measurements of the MPI based //ITPACK Jacobi CG solver (MPICH v1.0.7) on the Paragon.*

works have the shortest diameter, and thus generally deliver a better relative speedup.

Configuration		PARMACS	MPICH	PICL
1	time	109.97	79.75	110.13
	speedup	1.00	1.00	1.00
2	time	57.95	41.71	57.95
	speedup	1.90	1.91	1.90
4	time	30.27	25.11	30.40
	speedup	3.63	3.18	3.62
8	time	34.20	14.11	17.33
	speedup	3.22	5.65	6.35

Table 3. *Performance measurements of the PARMACS (v5.1), MPI (MPICH v1.0.7) and PICL (v2.0) based //ITPACK Jacobi CG solver for a 150x150 grid on the iPSC/860.*

The timing data in Table 4 shows that the overhead for the PICL and MPI portable communication library implementations on the nCUBE 2 is fairly low in comparison with the native communication system (Vertex). Our results indicate that PICL library implementation has less overhead than the MPICH library implementation on the nCUBE 2. However, Figure 3 shows that the speedup achieved for MPICH and PICL are approximately equal. On the iPSC/860, our results (Table 3) indicate that the MPICH communication library has less overhead in comparison with the PICL communication library. However, the benchmark application achieved slightly better speedup with the PICL communication library than with the MPICH library for this parallel

Configuration		Vertex	MPICH	PICL
1	time	496.97	501.92	496.96
	speedup	1.00	1.00	1.00
2	time	253.92	257.25	254.31
	speedup	1.96	1.95	1.95
4	time	147.59	150.32	148.09
	speedup	3.37	3.34	3.36
8	time	77.07	79.87	77.81
	speedup	6.45	6.28	6.39
16	time	40.90	43.94	41.81
	speedup	12.15	11.42	11.89
32	time	21.81	25.30	23.19
	speedup	22.78	19.84	21.43
64	time	12.35	16.15	14.00
	speedup	40.23	31.07	35.50

Table 4. *Performance measurements of the Vertex (native), MPI (MPICH v1.0.12) and PICL (v2.0) based //ITPACK Jacobi CG solver for a 150x150 grid on the nCUBE 2.*

platform. Results for the NX library on the Paragon and iPSC/860 are not yet available as we are currently evaluating this implementation.

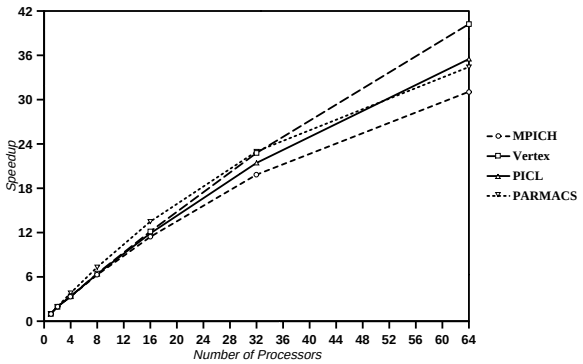

Figure 3. *Speedup Comparison of different communication library implementations of the //ITPACK Jacobi CG solver on the nCUBE 2.*

Tables 5, 6, 7 and 8 list the performance measurements for the workstation clusters for different portable communication library packages. On the Solaris–workstation–network, the execution times are approximately equal for MPICH, CHIMP and LAM portable communication library implementations. However, the MPICH communication library delivers slightly better speedup than the LAM and CHIMP libraries for both the 150x150 and 200x200 grid sizes in the benchmark application. In Tables 5 and 6 we compare

		Configuration				
		1	2	4	8	16
M	time	205.58	108.94	78.18	53.34	N/A
	spd	1.00	1.89	2.63	3.85	N/A
C	time	196.34	114.90	99.50	82.32	82.19
	spd	1.00	1.71	1.97	2.39	2.39
L	time	238.29	132.35	122.94	138.29	224.16
	spd	1.00	1.80	1.94	1.72	1.06
P	time	159.23	146.89	83.29	59.69	67.44
	spd	1.00	1.08	1.91	2.67	2.36

Table 5. *Performance measurements of the MPI and PVM based //ITPACK Jacobi CG solver implementations on the SunOS4–workstation–network for a 150x150 grid. (N/A = Not Available). Row labelling: M (MPICH v1.0.7), C (CHIMP v2.0), L (LAM v5.2), P (PVM v3.3).*

		Configuration				
		1	2	4	8	16
M	time	360.61	187.87	137.84	82.76	N/A
	spd	1.00	1.92	2.62	4.36	N/A
C	time	353.32	189.33	143.73	105.99	94.34
	spd	1.00	1.87	2.46	3.33	3.75
L	time	413.29	208.57	171.13	165.90	230.79
	spd	1.00	1.98	2.42	2.49	1.79
P	time	275.55	147.72	158.76	76.50	53.07
	spd	1.00	1.87	1.74	3.60	5.19

Table 6. *Performance measurements of the MPI and PVM based //ITPACK Jacobi CG solver implementations on the SunOS4–workstation–network for a 200x200 grid. (N/A = Not available). Row labelling: M (MPICH v1.0.7), C (CHIMP v2.0), L (LAM v5.2), P (PVM v3.3).*

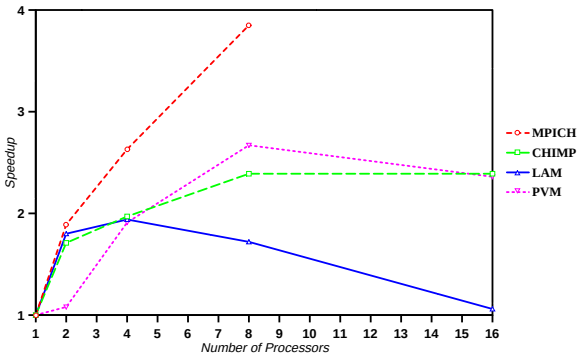

Figure 4. *Speedup Comparison of different portable communication library based //ITPACK Jacobi CG solver implementations on the SunOS4–workstation–network*

		Configuration			
		1	2	4	8
M	time	74.63	41.16	28.33	21.74
	spd	1.00	1.81	2.63	3.43
C	time	74.91	42.56	32.37	22.76
	spd	1.00	1.76	2.31	3.29
L	time	75.56	42.30	33.90	22.49
	spd	1.00	1.79	2.23	3.36

Table 7. *Performance measurements of the MPI based //ITPACK Jacobi CG solver implementation on the Solaris–workstation–network for a 150x150 grid. Row labelling: M (MPICH v1.0.12), C (CHIMP v2.0), L (LAM v6.0).*

		Configuration			
		1	2	4	8
M	time	131.91	70.65	45.49	30.40
	spd	1.00	1.87	2.90	4.34
C	time	133.07	72.86	48.02	31.25
	spd	1.00	1.83	2.77	4.26
L	time	132.74	72.63	50.41	30.57
	spd	1.00	1.83	2.63	4.34

Table 8. *Performance measurements of the MPI based //ITPACK Jacobi CG solver implementation on the Solaris–workstation–network for a 200x200 grid. Row labelling: M (MPICH v1.0.12), C (CHIMP v2.0), L (LAM v6.0).*

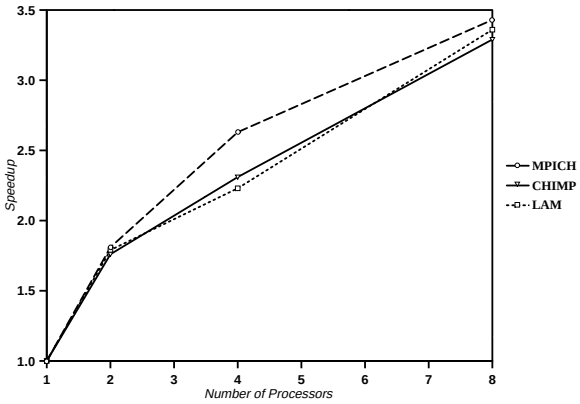

Figure 5. *Speedup Comparison of different portable communication library based //ITPACK Jacobi CG solver implementations on the Solaris–workstation–network*

the performance of the three MPI library implementations with the PVM portable communication library. It should be noted that the timing data listed in these two tables were obtained for older versions of the communication library implementations. The current versions of these libraries will probably deliver better performance. Considering these older library implementation versions on the SunOS4–workstation–network, the PVM communication library obtained the relatively lowest execution times and the best relative speedup. Figures 4 and 5 depict the relative speedup achieved by the benchmark application on the SunOS4–workstation–network and the Solaris–workstation–network for different communication libraries.

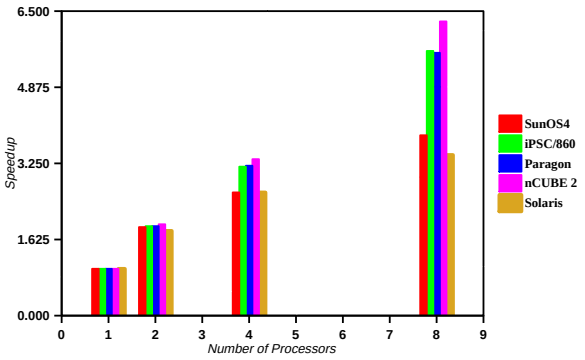

Figure 6. *Speedup Comparison of the MPI (MPICH) based parallel ITPACK Jacobi CG solver on different hardware platforms*

Figure 6 shows the speedup for the MPICH communication library implementation on all the hardware platforms under consideration, for the benchmark problem with a 150x150 grid size. This figure clearly indicates that the best speedup was achieved on the nCUBE 2 platform.

Configuration		Mesh Size	
		3684	14844
1	time	27.95	98.44
	speedup	1.00	1.00
2	time	8.83	38.77
	speedup	3.17	2.54
4	time	5.53	22.09
	speedup	5.05	4.46
8	time	4.94	22.93
	speedup	5.66	4.29

Table 9. *Performance of the MPI (MPICH) based parallel mesh/decomposition generator on a cluster of eight SparcStation 20s.*

Configuration		Mesh Size	
		3684	14844
1	time	168.16	595.72
	speedup	1.00	1.00
2	time	62.66	213.53
	speedup	2.68	2.79
4	time	27.40	91.90
	speedup	6.14	6.48
8	time	13.94	50.16
	speedup	12.06	11.88

Table 10. *Performance of the MPI (MPICH) based parallel mesh/decomposition generator on a cluster of eight Sun IPCs.*

Configuration		Mesh Size	
		3684	14844
1	time	31.79	109.52
	speedup	1.00	1.00
2	time	11.77	39.95
	speedup	2.70	2.74
4	time	4.91	17.27
	speedup	6.47	6.34
8	time	2.47	8.56
	speedup	12.87	12.79
16	time	1.88	6.26
	speedup	16.91	17.50
32	time	1.02	3.72
	speedup	31.17	29.44
64	time	0.70	2.92
	speedup	45.41	37.51

Table 11. *Performance of the MPI (MPICH) based parallel mesh/decomposition generator on the Intel Paragon.*

Configuration		Mesh Size	
		3684	14844
1	time	93.07	318.46
	speedup	1.00	1.00
2	time	32.69	111.61
	speedup	2.85	2.85
4	time	12.73	44.20
	speedup	7.31	7.20
8	time	6.23	21.66
	speedup	14.94	14.70

Table 12. *Performance of the MPI (MPICH) based parallel mesh/decomposition generator on the iPSC/860.*

Configuration	Time	Speedup
1	140.00	1.00
2	62.02	2.26
4	26.91	5.20
8	13.55	10.33
16	10.29	13.61
32	5.71	24.52

Table 13. *Performance of the MPI (MPICH) based parallel mesh/decomposition generator on the nCUBE 2.*

Tables 9, 10, 11, 12 and 13 list the performance measurements for the MPI based parallel finite element mesh/decomposition generator for different sized meshes (3684 elements and 14844 elements). The super linear speedup achieved is due to the behavior of the complexity of the underlying computation as a function of the number of splittings. Furthermore, the algorithm has low communication requirements. The tables indicate that the computation scales almost perfectly with the number of processors on all the platforms. The mesh with 14844 elements could not run on the nCUBE 2 machine due to memory resource limitations. The best speedup and execution times were obtained on the Intel Paragon machine.

5 Conclusions

In this paper we present a comparison of several MPI implementations on different hardware platforms based on the performance of PDE solvers from the //ELLPACK PSE. For our benchmark application and our parallel mesh generator/decomposer, we observed that the performance of the various portable communication libraries is mostly comparable and that reasonable speedup can be noticed even on workstation clusters connected via an Ethernet. In our experiments on the workstation clusters, the MPICH library implementation performed slightly better than the LAM and CHIMP implementations. Amongst the parallel machines, the best speedup for portable communication libraries was obtained on the nCUBE 2 machine, which is better balanced (in terms of computation and communication efficiency) than the others considered. The overhead of a portable communication library versus the native library was measured on the nCUBE 2 (Table 3) and our results indicate that although the overhead is negligible for small numbers of processors, this differential increases significantly for larger configurations (32 or 64 nodes). We are currently re–generating the performance data using the newest releases of all the communication libraries on all our target hardware platforms. We are also porting our software to an IBM SP/2.

References

[1] R. A. Bruce, J. G. Mills, and A. G. Smith. Chimp/mpi user guide. Technical Report EPCC-KTP-CHIMP-V2-USER 1.2, University of Edinburgh, UK.

[2] G. Burns, R. Daoud, and J. Vaigl. Lam: An open cluster environment for mpi. *Ohio Supercomputing Center, Ohio*.

[3] I. Corporation. iPSC/860: System user's guide. March 1992.

[4] E. Dillion, C. G. D. Santos, and J. Guyard. Homogeneous and heterogeneous networks of workstations: Message passing overhead. In *Proc. MPI Developers Conference*, 1995.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. *PVM: A Users' Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.

[6] G. A. Geist, M. T. Heath, B. Peyton, and P. Worley. A user's guide to picl: A portable instrumented communication library. Technical Report ORNL/TM-11616, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, 1992.

[7] B. Gropp, R. Lusk, T. Skjellum, and N. Doss. Portable MPI Model Implementation. Technical Report No., Argonne National Laboratory, July 1994.

[8] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, October 1994.

[9] R. Hempel, H.-C. Hoppe, and A. Supalov. A proposal for a parmacs library interface. Technical Report GMD, Postfach 1316, D-5205 Sankt Augustin 1, Germany, October 1992.

[10] S. Kim, E. N. Houstis, and J. R. Rice. Parallel stationary iterative methods and their performance. In D. Marinescu and R. Frost, editors, *INTEL supercomputer users group conference*, San Diego, 1994.

[11] S.-B. Kim. Parallel Numerical Methods for Partial Differential Equations. Ph.D. Thesis, 1993, Department of Mathematics, Purdue University. Technical Report CSD-TR-94-000, pp.1-00, Purdue University, Computer Science, 1993.

[12] D. Kinkaid, J. Respess, and R. Grimes. Algorithm 586: Itpack 2c: A fortran package for solving large linear systems by adaptive accelerated iterative methods. *ACM Tran. Math. Soft.*, 8(0):302–322, 1982.

[13] nCUBE Corporation. nCUBE 2 programmer's guide. *Release 2.0*, 1990.

[14] N. Nupairoj and L. Ni. Performance evaluation of some mpi implementations on workstation clusters. Technical report, Dept. of Computer Science, Michigan State University, 1995.

[15] S. Weerawarana, E. Houstis, A. Catlin, and J. Rice. //ellpack: A system for simulating partial differential equations. In *Proceedings of IASTED International Conference on Modelling and Simulation*, 1995. to appear.

[16] P. Wu and E. N. Houstis. Parallel mesh generation and decomposition. *Computer Systems in Engineering*, (CSD-TR-93-075, pp. 1-49), 1994.

[17] P.-T. Wu. *Parallel Mesh Generation and Domain Decomposition*. PhD thesis, Department of Computer Sciences, Purdue University, 1995.