1989

Benchmarking of MIMD Hardware on Subdomain Splitting Elliptic PDE solvers.

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

E. A. Vavalis

Report Number:
89-874

BENCHMARKING OF MIMD HARDWARE ON SUBDOMAIN SPLITTING ELLIPTIC PDE SOLVERS

E. N. Houstis
J. R. Rice
E. A. Vavalis

CSD-TR-874
March 1989

BENCHMARKING OF MIMD HARDWARE ON SUBDOMAIN SPLITTING ELLIPTIC PDE SOLVERS

E.N. Houstis, J.R. Rice and E.A. Vavalis

Computer Sciences Department
Purdue University
Technical Report CSD-TR-874
CAPO Report CER-89-13
March 1989

Work supported in part by NSF grand CCR-8619817 and AFSOR grant 88-0243.

BENCHMARKING OF MIMD HARDWARE ON SUBDOMAIN SPLITTING ELLIPTIC PDE SOLVERS

E.N. Houstis, J.R. Rice and E.A. Vavalis
Purdue University
Department of Computer Science
West Lafayette, IN 47907

ABSTRACT

In this paper we model and evaluate the performance of a number of parallel machines that are commercially available over a subdomain splitting method (SPM) for solving elliptic partial differential equations (PDEs). The basic idea of this method is to subdivide the domain of the PDE problem into a number of overlapping subdomains and decompose the problem into one that involves solution of PDE boundary value problems on the subdomains. The interaction among subdomains is implemented according to Jacobi and Gauss-Seidel iterative schemes. For the bus architectures we use the loop-based approach to specify the inherent parallelism and the Argonne SCHEDULE package to implement it. For the local memory machines, we use block and substracturing data structures for the implementation of SPM and we map the underlying computation onto a 2-dimensional grid. Preliminary results indicate that both the shared and non-shared memory machines behave equally well for the various configurations.

1. INTRODUCTION

The primary objective of this study is to study the relative performance of PDE-solver/machine pairs for a number of commercially available multiprocessor machines. It turns

Work supported in part by NSF grant CCR-8619817 and AFSOR grant 88-0243.

out that there exist several benchmarking tests for serial computers, while no real synthetic benchmark tests are available for analyzing the performance of parallel machines are available [Duni 88]. In this paper we use a PDE solver consisting of a Cubic Spline Collocation discretization method coupled with a Schwarz domain splitting methodology [Hous 88a], [Hous 88b] to exploit the characteristics of commercially available hardware. The basic characteristics of SCSC that makes it a good synthetic parallel benchmarking test are:

- Double precision float and integer arithmetic.
- Bottle-neck and pipelined parallel structures.
- Nearest neighbor communication (on hypercubes).
- Small efficiency ratio ($\frac{communication\ time}{computation\ time}$)

It should be pointed out that effective use of multiprocessors for solving elliptic PDEs especially with irregular domains, requires proper partition, allocation and load balancing. In fact we are building a *Domain Decomposition Tool* [Hous 89] that solves the domain decomposition problem and provides us with allocation and load balancing strategies. This tool will be incorporated into the //ELLPACK system (a parallel implementation of ELLPACK [Rice 85]). In this study we restrict ourselves to rectangular PDE domains and we assume decompositions into rectangular subdomains of the same size.

In Section 2 we describe briefly the Schwarz variant of the Cubic Spline Collocation method. In Section 3 we present a transportable parallel implementation of the SCSC method on bus-based architectures with shared memory using the SCHEDULE package. Timing results from the BALANCE SYMMETRY, BALANCE SEQUENT and ALLIANT FX/8 are also

given. The implementation on hypercube architectures is presented in Section 4 together with measurements for three hypercube multiprocessors, namely the NCUBE, the iPSC/1 and the iPSC/2. In Section 5 we present our conclusions.

2. SCHWARZ CUBIC SPLINE COLLOCATION.

2.1. The Cubic Spline Collocation Method.

In this section we give a brief description of the Cubic Spline Collocation discretization method in two-dimensional rectangular domains $\Omega = [a,b] \times [c,d]$. We assume a uniform rectangular mesh $\Delta \equiv \{(x_i,y_j)$: $i = 0$ to N and $j = 0$ to $M\}$ of Ω, and define the tensor product of one-dimensional cubic splines

$$S_{3,\Delta} \equiv S_{3,\Delta_x} \otimes S_{3,\Delta_y} \equiv \mathbf{P}_{3,\Delta} \cap C^{k-1}(\Omega)$$

where $\mathbf{P}_{3,\Delta}$ denotes the space of piecewise cubic polynomials with respect to Δ. Throughout we assume a second order elliptic PDE problem $Lu = f$ defined in Ω with Dirichlet homogeneous boundary conditions ($u = 0$). According to cubic spline collocation methodology, we seek an approximation u_Δ ?? $S_{3,\Delta}$ that satisfies exactly th e boundary conditions and it is determined by the equations:

(*I*) $Lu_\Delta + P_L u_\Delta = f$ at interior grid points,

(*II*) $Lu_\Delta = f$ at the boundary grid points other than corners

where $P_L u_\Delta \equiv \frac{1}{12}(\Gamma_x + \Gamma_y)u_\Delta$ with $\Gamma_z \equiv D^2_{z-h} - 2D^2_z + D^2_{z+h}$.

The general formulation of cubic spline collocation methods can be found in [Hous 88a].

Without loss of its convergence properties, we set the four corner degrees of freedom of u_Δ equal to zero, i.e. $\alpha_{0,0} = \alpha_{0,M+1} = \alpha_{N+1,0} = \alpha_{N+1,M+1} = 0$.

2.2. The Schwarz Cubic Spline Collocation method.

The basic idea of the Schwarz splitting methods [Mill 65] is to subdivide the domain into a number of overlapping subdomains and decompose the problem into one that involves solution of boundary value problems on the subdomains. For example, assuming a partition of two overlapping subdomains Ω_1, Ω_2 with interior interfaces Γ_1 and Γ_2, then for the considered PDE probelm the method can be described as follows:

Step 1

Specify $u_2^{(0)}$ on Γ_1 (initial guess).

Step 2:

Solve $Lu_1 = f$ in Ω_1, with $u_1 = g$ on

$\partial\Omega_1 \cap \partial\Omega$, and $u_1 = u_2^{(0)}$ on Γ_1.

Step 3:

Solve $Lu_2 = f$ in Ω_2, with $u_2 = g$ on

$\partial\Omega_2 \cap \partial\Omega$, and $u_2 = u_1$ on Γ_2.

where $\partial\Omega$, $\partial\Omega_1$ and $\partial\Omega_2$ are the boundaries of Ω, Ω_1 and Ω_2. The Schwarz Cubic Spline Collocation method consists of discretizing the subproblems using the Cubic Spline Collocation scheme introduced in section 2.1 while the interaction between the subproblems is computed by an iterative method. For both the convergence analysis and the implementation of the SCSC

method, it is more convenient to formulate it at the solution phase of spline collocation equations.

2.3. The Model Problem.

For the performance evaluation analysis we have considered the self adjoint elliptic equation

$$D_x(e^{xy}D_x u) + D_y(e^{-xy}D_y u) - \frac{c(x,y)}{(1+x+y)}u = f(x,y) \tag{2.3}$$

with Dirichlet boundary conditions on the unit square. The right hand side f corresponds to the exact solution $u = 0.75e^{xy}\sin(\pi x)\sin(\pi y)$. The function $c(x,y)$ in equation (3.2) is used as a parameter to control the rate of convergence of the SCSC method. A recent study [Hous 88c] has shown that for moderate mesh sizes direct methods and in particular the DGBFA/DGBSL routines from LINPACK produce an efficient solution to cubic spline collocation equations. In this paper we have selected these routines to solve the system locally to each processor. The convergence tolerance for each grid size is picked in such a way that guarantees the fourth order convergence of the Cubic Spline Collocation. The code for the parallel SCSC is written in FORTRAN and the computations were performed in double precision. All the presented here timings are in seconds.

3. PARALLEL IMPLEMENTATION OF SCSC ON BUS ARCHITECTURES.

In this section we discuss the implementation of the parallel SCSC algorithm on bus-based multiprocessor machines with shared memory. To produce an efficient parallel implementation of SCSC method on such machines the data dependencies and the parallel structures of the

method should be fully understood and implemented using a high level language with parallel extensions. The lack of standardized parallel extension of FORTRAN make portability among parallel machines difficult while high level parallelism leads to non-trivial implementation questions in expressing the associated data dependencies. For the implementation of SCSC algorithm we have used a package called SCHEDULE [Dong 87] that allowes to define data dependencies and parallel structures in an easy way and assure the portability of our code to several machines. Following the philosophy and data structures of the SCHEDULE we have broken our FORTRAN program into a set of calls of the self-contained routine *schwarz_colloc* that operates on shared data structures consisting of a workspace array for storing the matrix and the pivots, a vector array that contains the values of the degrees of freedom and the right hand side vector. This subroutine basically solves the discrete systems associated with each subdomain provided that certain data dependancies are satisfied. These dependencies are needed for the correct updating of the right hand side.

In this implementation the main program of parallel SCSC is responsible for the formulation of the global coefficient matrix and the right hand side the initialization of the Schwarz process and the timming of various segments and calls the SCHED routine from the SCHEDULE package with arguments the number of processors *nprocs*, the parallel subroutine *parscsc* and the list of its arguments. In routine *parscsc* we specify the data dependencies associated with each process which consists of a call to *schwarz_colloc* unique for each subdomain and each iteration. In this way the SCHEDULE package will create a pool of processes and and a scheduling mechanism for executing the units of computation on a multiprocessor system. In all our experiments the measured overhead due to the use of the SCHEDULE software was

negligible.

In Figure 3.1 we present the relative speedup of SCSC method using the Jacobi iteration scheme for the PDE problem given in Section 2.2 with $c(x,y)=-1$ and a 41x41 global discretization grid. The speedup for n processors is computed by the ratio

$$\frac{\textit{Time for the SCSC on } n \textit{ processors}}{\textit{Time for the SCSC on } 1 \textit{ processors}}.$$

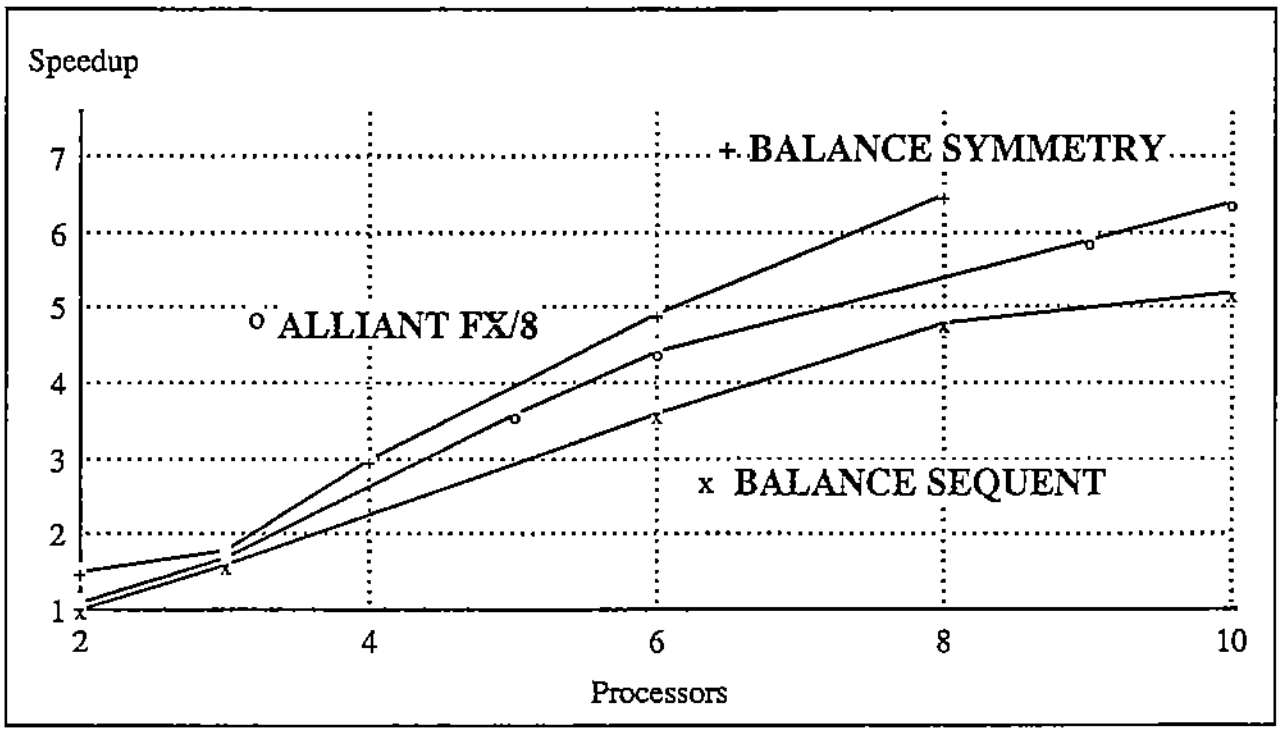

Figure 3.1. Relative speedups of the Jacobi SCSC on bus architectures.

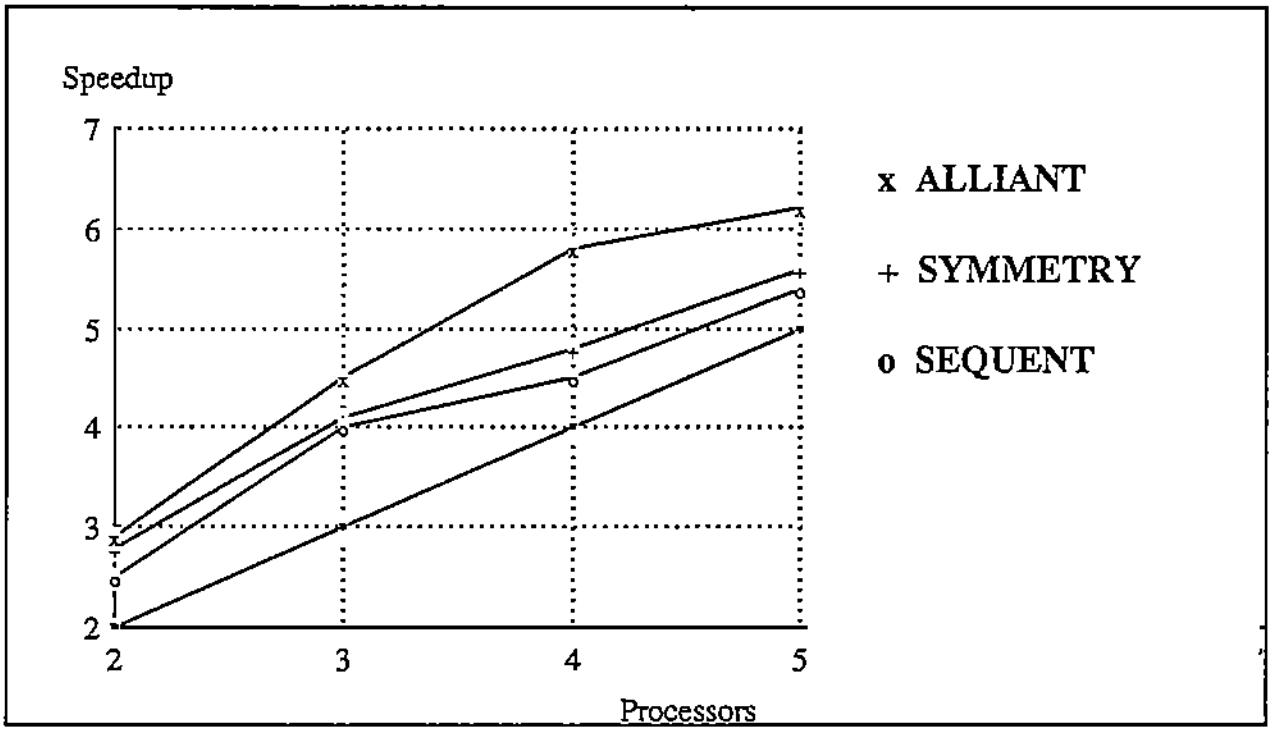

Figure 3.2. Per iteration speedups of SCSC of the Gauss-Seidel.

4. PARALLEL IMPLEMENTATION OF SCSC ON HYPERCUBES.

The parallel implementation of SCSC on hypercube systems differs significantly from the one for the bus-based architectures. The part of the code that runs on the host processor first maps the hypercube onto a 2-dimensional grid using a binary reflected Gray code. Then it forms the global spline collocation linear system and loads the node program. Finally, it sends the local coefficient matrices corresponding to various subdomains together with an initial guess of the corresponding degrees of freedom to the associated nodes. In this study, we do not consider the communication cost among the host and node processors since the discretization can take place at each node with minimum overhead. The local coefficient matrix is factored on each node and the Jacobi iteration scheme is employed to implement the interaction among sub-domains. At the beginning all processors become fully utilized by working on each subdomain. When they are finished they all send and receive the appropriate degrees of freedom that share

with the neighbors and start working on the next iteration. In the case of Gauss-Seidel each node gets the values for the degrees of freedom from its nearest neighbor in the west direction and the corresponding degrees of freedom from the nearest neighbor in the south direction before updating its right hand side and back solving. Next it sends the appropriate data to all its four nearest neighbors and waits to get data from north and west before starting the next iteration. After achieving convergence the node processors read their clocks and send the time together with the associated degrees of freedom to the host. The host reads the time send by each node and takes their average as the resulting time.

The above procedures can be described in Pseudo C code as follows:

HOST PROGRAM

```
call init;
call generate_collocation_matrix;

for ( node = 1; node < number_of_subdomains; node++ ) {
        call send_parameters(node);
        call send_local_matrix(node);
        call get_local_solution(node);
}

call form_solution;
```

NODE PROGRAM (GAUSS-SEIDEL)

```
call get_parameters;
call get_local_matrix;
call factor_matrix;

for ( iter = 1; iter < max_iterations; iter++ ) {
        call get_solution(south,west);
        call modify_rhs
        call backsolve;
        call send_solution(north,east,south,west);
        call get_solution(north,east);
}

call send_local_solution(host);
```

NODE PROGRAM (JACOBI)

```
call get_parameters;
call get_local_matrix;
call factor_matrix;

for ( iter = 1; iter < max_iterations; iter++ ) {
        call modify_rhs
        call backsolve;
        call send_solution(north,east,south,west);
        call get_solution(north,east,south,west);
}

call send_local_solution(host);
```

Next we present performance data from three hypercube machines, namely the NCUBE (128 processors), iPSC/1 (32 processors) and iPSC/2 (32 processors). The FORTRAN implementation for the three machines is almost identical. Specifically they differ only in the timing routine and the message-passing primitives. Its should be pointed out that the receive primitive blocks the execution until the desired data have been received while the send primitive is non-blocking.

In Figure 4.1 we present the performance of the parallel SCSC method implemented with the Jacobi iteration scheme on the three hypercubes. The PDE problem considered is the one given in Section 2.3 with $c(x,y) = -1$ and a uniform global grid of 41x41 lines. It has been observed that for many other discretization grid sizes, the iPSC/2 was approximately 3.8 times faster than the NCUBE while the later was 2.6 times faster than the iPSC/1.

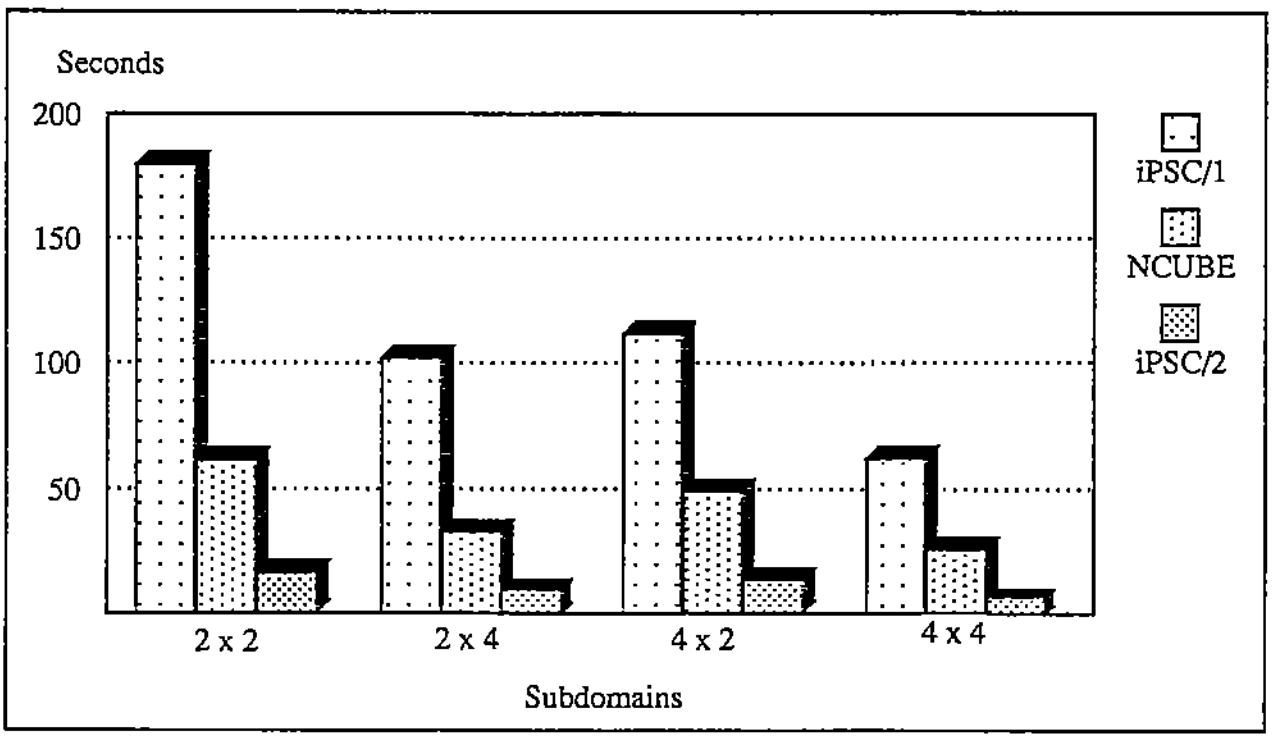

Figure 4.1. Performance of SCSC with Jacobi iteration scheme for various domain decompositions on NCUBE, iPSC/1 and iPSC/2.

In Figure 4.2 we give the measured speedup for the parallel SCSC method with both Jacobi and Gauss Seidel iteration schemes applied to two PDE problems on NCUBE and iPSC/2. Two PDE problems were consider and they correspond to the one given given in Section 2.3 with $c(x,y) = -1$ (PDE 1) and $c(x,y) = -e^{3+x+y}$ (PDE 2) and a 65x65 global discretization grid was used. Speedup has been obtained with respect to sequential Cubic Spline Collocation where the global coefficient system was solved by the fastest sequential method (Gauss elimination).

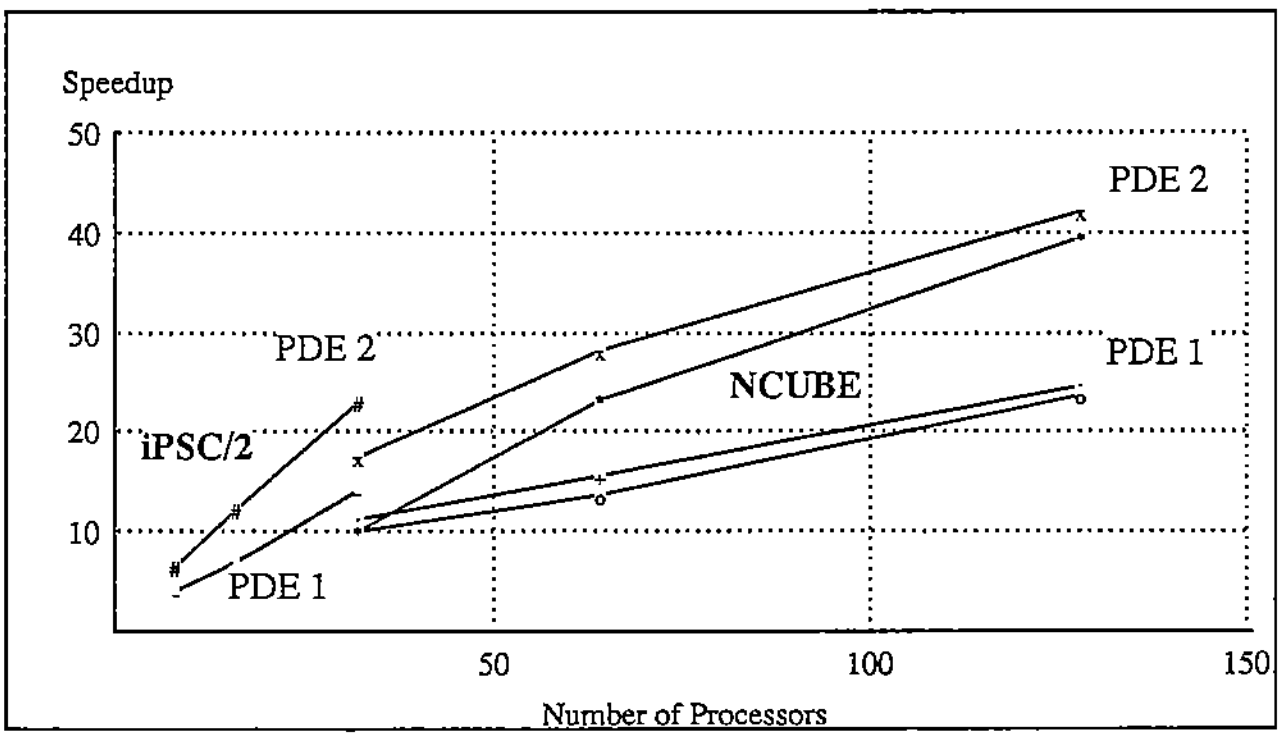

Figure 4.2. Speedups of SCSC with Jacobi (legends x, + and #) and Gauss-Seidel (legends *, o and -) iteration schemes for two PDEs on the NCUBE and iPSC/2.

As it was pointed out in Section 2, the Gauss-Seidel scheme requires fewer iterations to converge thus less communication and computation than the Jacobi scheme. On the other hand it leads to a significantly lower utilization of the node processors. Figure 4.2 confirms the above observations.

5. CONCLUSIONS.

We have described the implementation of a PDE solver with inherent high level parallelism on multiprocessor systems and use that implementation as a synthetic benchmark test to measure the performance of several commercially available parallel machines. We have seen that, despite differences all the test machines exhibit similar performance. On bus machines no serious effects due to overflow of the cache memory and to the saturation of the bus have been observed, as in other parallel PDE solvers [Hous 87]; it is our believe that the good efficiency ratio of SCSC method should be the primary reason for that. Results on hypercubes exhibit good speed up while the second generation hypercube iPSC/2 turns to be significantly more efficient with respect to both cpu and communication time.

Finally it is worth to make a remark on the performance of chaotic iterations for SCSC on parallel architectures. More specifically we removed the synchronization from the SCSC algorithm by using non-blocking communication primitives on the hypercubes and by removing the data dependencies on the shared memory machines. This asynchronous iterative scheme converges very slow and the cpu time increases significantly on all the machines. It turns out that this is a characteristic of the Schwarz procedure. In earlier work [Hous 87] we developed an asynchronous iterative method for the solution of elliptic PDEs on bus architectures with success. The characteristic of that algorithm was the need of updating the whole unknown vector before start working on the next iteration. In contrast the SCSC needs updated values only for unknowns near the boundaries and that probably leads us to inefficient asynchronous schemes.

ACKNOWLEDGMENTS

We are grateful to Advanced Computing Research Facility at Argonne Laboratories for allowing us to use their iPSC/1, ALLIANT FX/8 and to Advance Computing Facility at Cornell's Theory Center for allowing us to use their iPSC/2.

REFERENCES

[Dang 87] J. Dongarra and D. Sorensen, *A Portable Environment for Developing Parallel Fortran Programs*, Parallel Computing, Vol 5, No 1&2, 175–186.

[Hous 87] E. N. Houstis, J. R. Rice and E. A. Vavalis, *Parallelization of a new class of cubic spline collocation methods*, in: *Advances in Computer Methods for Partial Differential Equations*, Vol. VI, (R. Vichnevetsky, R. S. Stepleman, editor), 167–174.

[Hous 88a]E. N. Houstis, J. R. Rice and E. A. Vavalis, *Convergence of an $O(h^4)$ Cubic Spline-Collocation Method for Elliptic Partial Differential Equations*, SIAM J. Num. Anal., Vol 25, No 1, 54-73.

[Hous 88b]E. N. Houstis, J. R. Rice and E. A. Vavalis, *A Schwarz Splitting Variant of Cubic Spline Collocation Methods for Elliptic PDEs*. HCCA3 Proceedings, ACM Press, 1746–1754.

[Hous 88c]Houstis, E.N., J.R. Rice, C.C. Christara and E.A. Vavalis, *Performance of Scientific Software*, in: *Scientific Software*, (J.R. Rice, ed.), Springer Verlag, 123–155.

[Hous 89] E.N. Houstis, P.N. Papachiou, J.R. Rice and M.K. Samartzis, *Domain Decomposer: A Software Tool for Partitioning PDE Computations Based on Geometry Decomposition Strategies*, Purdue University Technical Report, in preparation.

[Duni 88] T. H. Dunigan, *Performance of a Second Generation Hypercube*, Tech. Report. ORNL/TM-10881, Oak Ridge National Laboratory, Mathematical Science Section.

[Mill 65] K. Miller, *Numerical Analogs to the Schwarz Alternating Procedure*, Numer. Math., 7, 91–103.

[Rice 85] J. R. Rice and R. F. Boisvert, *Solving Elliptic Problems Using Ellpack*, Springer-Verlag, New York.