Distribution Category:
Mathematics and
Computer Science (UC-405)

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439-4801

**ANL-93/41**

# Early Experiences with the IBM SP1 and the High-Performance Switch

*Edited by*

William Gropp

November 1993

This work was supported in part by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

## Contents

# Early Experiences with the IBM SP1 and the High-Performance Switch

Edited by

*William Gropp*

**Abstract**

The IBM SP1 is IBM's newest parallel distributed-memory computer. As part of a joint project with IBM, Argonne took delivery of an early system in order to evaluate the software environment and to begin porting programming packages and applications to this machine. This report discusses the results of those efforts once the high-performance switch was installed. An earlier report (ANL/MCS-TM-177) emphasized software usability and the initial ports to the SP1. This report contains performance results and discusses some applications and tools not covered in TM 177.

**Picture**

The picture on the title page shows the early time dynamics of vortex trapping by twin boundary defects in a high-temperature, Type-II superconductor. See Section 3.7 for more information. Thanks to David Levine for this picture.

**Contributors**

| | |
|---|---|
| Christian Bischof | Kimmo Forsman |
| Lori Freitag | William Gropp |
| Lauri Kettunen | Gary Leaf |
| David Levine | Ewing Lusk |
| William McCune | John Michalakes |
| Ross Overbeek | Mario Palumbo |
| Steven Pieper | Paul Plassmann |
| Xiaobai Sun | Steven Tuecke |
| Robert Wiringa | |

## 1 Introduction

The IBM SP1 is a new parallel computer designed to make the best use of IBM's powerful RISC technology combined with a high-speed switch.

Special features of this machine are (numbers are for the ANL machine)

- large memory per node (128 MBytes),
- local disks on each node (1 GByte),
- full Unix on each node (IBM AIX 3.2.4),
- high-performance nodes,
- high-performance switch,
- high I/O bandwidth off nodes, and
- relatively mature software environment.

This report describes the applications and programming packages that researchers at Argonne National Laboratory ported to the SP1 after the high-performance switch was delivered. A previous report [8] discussed the SP1 without the high-performance switch. This report discusses early experiences with the SP1; most of the tools and applications discussed in this paper have not yet been tuned for the SP1. Performance results are included for some of the applications; these should not be considered indicative of anything other than what can be accomplished with an initial port. See Section 1.1 below for more discussion on the benchmarks.

The software packages and tools are shown in Table 1. The applications (all parallel) are shown in Table 2

Three "transport layers" are available for use with the high-performance switch (henceforth just "switch"). The first of these is IP, providing enhanced performance to code written using Unix sockets for interprocessor communication. The second is EUI, IBM's message-passing system. IP and EUI applications may share the switch; multiple IP applications may share both nodes and the switch. IP and EUI run under the "Parallel Operating Environment," or POE. POE includes a number of tools, such as a parallel debugger and ParaGraph-like [12] visualization tool (`vt`). These two transport layers share a common interface to the switch known as *lightspeed*.

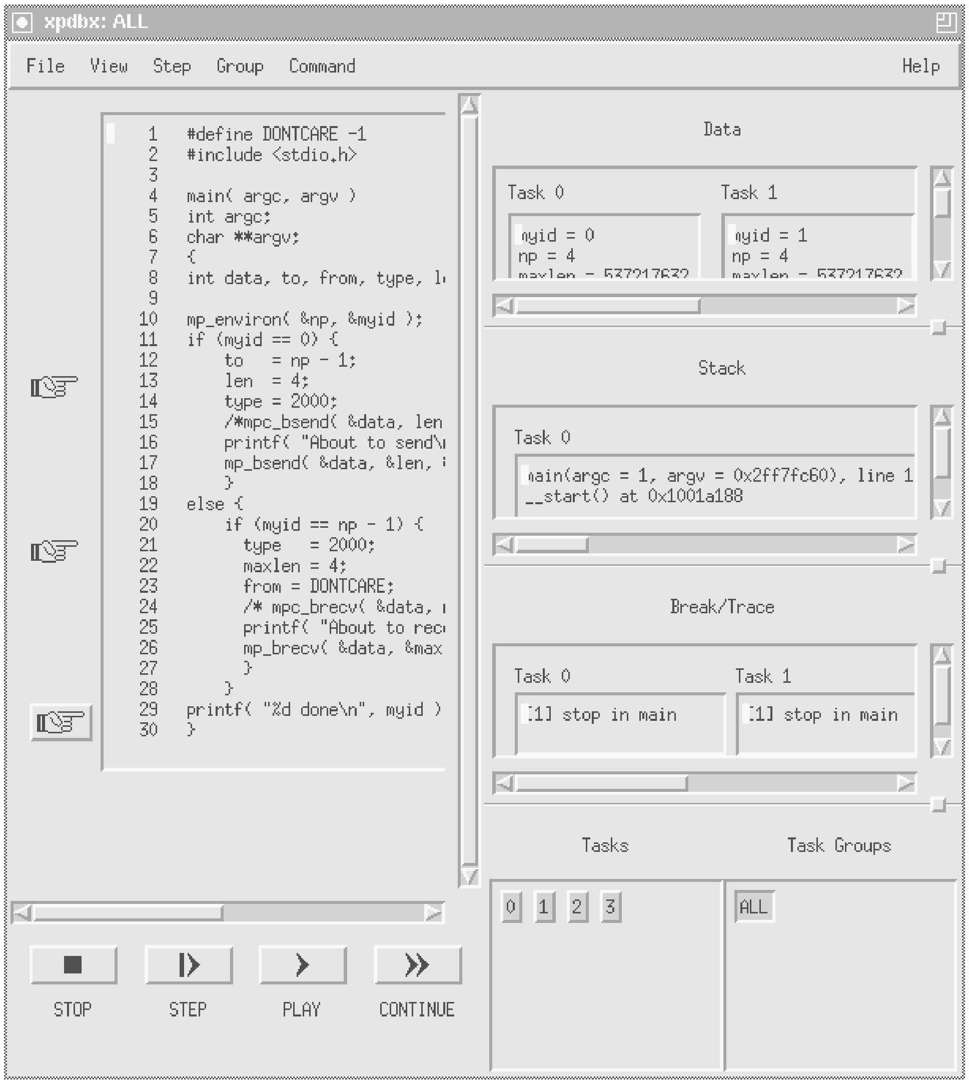
Figure 1 shows a display from `xpdbx`, the parallel debugger for the IBM SP1. A valuable feature of this debugger is that it displays the location in the code of each of the processors. In the example shown here, four processors are executing a program, with "hands" pointing

Table 1: Parallel tools described in this paper

| | |
|---|---|
| BlockSolve | Parallel sparse, symmetric linear systems |
| Chameleon | Lightweight and portable message-passing system |
| Fortran M | Parallel extensions to Fortran |
| Graphical display of system | Show usage of EUIH message-passing system |
| MPI | Message-passing interface draft standard |
| Parallel UNIX tools | Parallel versions of `cp`, `kill`, etc. |
| PCN | Program Composition Notation (a coordination language) |
| PETSc | Portable, extensible tools for scientific computing |
| PRISM | Parallel Linear Algebra |
| P4 | Portable message-passing and shared-memory library |

Table 2: Applications described in this paper

| | |
|---|---|
| Computational Electromagnetics | Model 3-d, arbitrary geometry magnets |
| Mesoscale weather model | Continent-sized weather model |
| Nuclear Structure | Monte Carlo computation |
| Parallel Community climate model | Global climate model |
| Phylogenetic tree | Program to construct phylogenetic trees from sequence data |
| Superconductivity | Modeling of flux vortices in high-temperature superconductors (three applications) |
| Theorem prover | Distributed associative-commutative theorem prover |



```
#define DONTCARE -1
#include <stdio.h>

main( argc, argv )
int argc;
char **argv;
{
int data, to, from, type, l

mp_environ( &np, &myid );
if (myid == 0) {
    to   = np - 1;
    len  = 4;
    type = 2000;
    /*mpc_bsend( &data, len
    printf( "About to send\
    mp_bsend( &data, &len,
    }
else {
    if (myid == np - 1) {
      type   = 2000;
      maxlen = 4;
      from = DONTCARE;
      /* mpc_brecv( &data,
      printf( "About to rec
      mp_brecv( &data, &max
      }
    }
printf( "%d done\n", myid )
}
```

```
myid = 0
np = 4
maxlen = 537217632
```

```
myid = 1
np = 4
maxlen = 537217632
```

```
main(argc = 1, argv = 0x2ff7fc60), line 1
__start() at 0x1001a188
```

```
[1] stop in main
```

```
[1] stop in main
```

Figure 1: Display from `xpdbx` on the IBM SP1

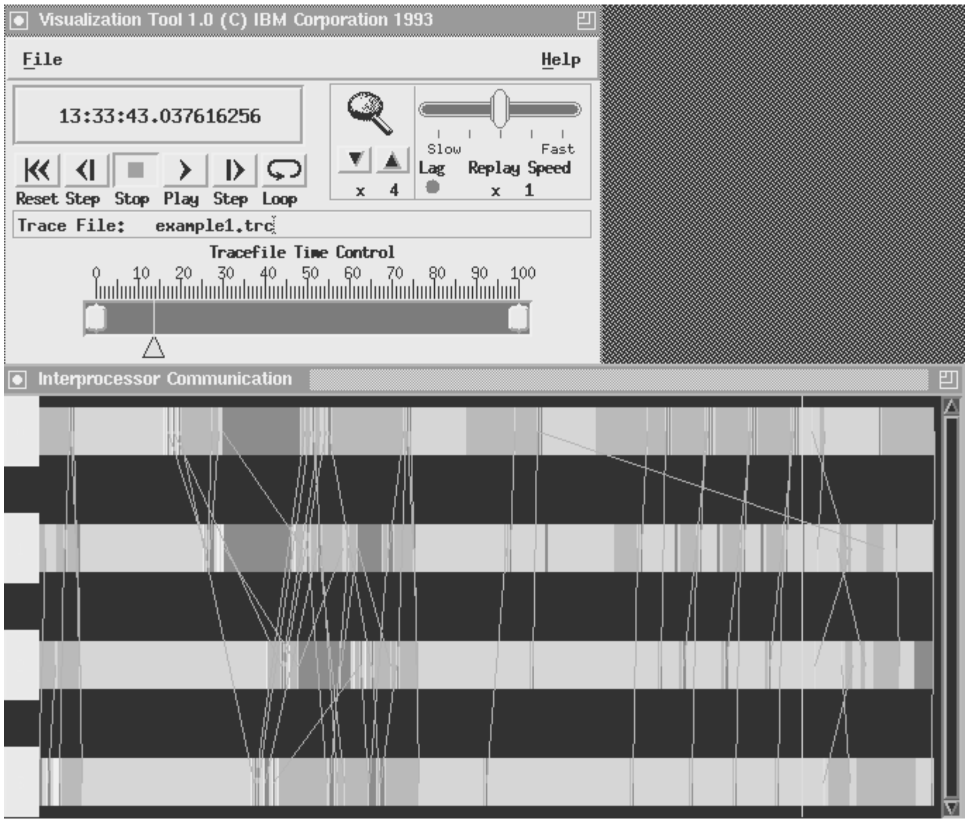


Figure 2: Display from `vt`

at the three locations in the code where each processor has stepped to. The hands in the "raised" box at the bottom indicate that several processors are at the same line.

Figure 2 shows a display from `vt`, the performance visualization tool for the IBM SP1. Note the control box that allows the user to move the display back and forth and to zoom in or out.

The third transport layer is a an experimental low-overhead version of EUI, called EUIH; it is not available as a product. This layer does *not* support any of the POE tools and is incompatible with IP and EUI executables. However, EUIH does offer some performance advantages for applications that need high-speed communications (see Section 2.5). We emphasize that this is a research tool and was made available to Argonne under a special joint-research arrangement.

Each of these subsections was contributed by the author named in the section; minor editing has been done, and any errors are the responsibility of the editor.

### 1.1 Comments on Timings

All timings and performance results in this document are preliminary. Because the IBM SP1 is running a full Unix on each node, it is more difficult than on MPPs that run a single-user operating system to insure that no processes other than the program being benchmarked are using resources. The performance figures given here were done without running on a stand-alone system.

Many of the results presented here are on relatively small numbers of processors; again, this is primarily because little time has been made available for single-user benchmarks.

## 2 Programming Packages and Tools

This section describes the programming packages that support the applications that have been ported to the SP1. The packages include several numerical libraries (BlockSolve, PETSc, and PRISM) and three programming packages (Chameleon, PCN, and `p4`). In addition, a port of part of the draft message-passing standard (MPI) has been made to the SP1.

### 2.1 Graphical Display of System Use

Contributed by **William Gropp and Ewing Lusk**

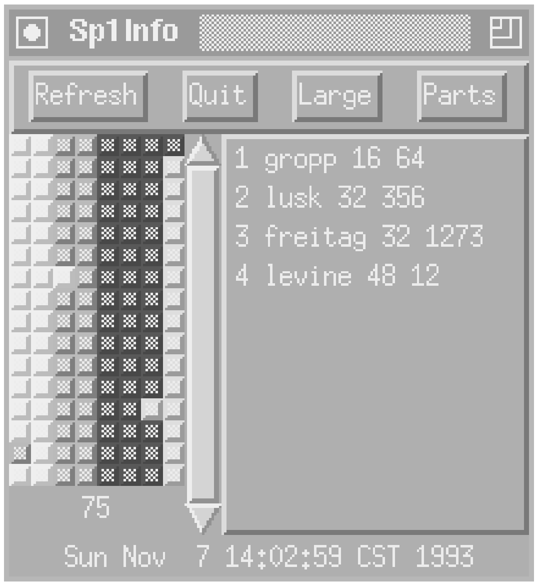
We have written a tool, `xsp1info`, that displays the usage of the SP1 when using the EUIH message-passing system. This tool, written using `tcl` and `tk`, displays

- the partitions in use. Each partition is displayed in a different color.
- a list of partitions, containing the username, size, and amount of time the partition has been in use.

An example display is shown in Figure 3.

In addition, each node is represented by a button. Pushing this button with the mouse can bring up an `xterm` on that node or show the load average (depending on which mouse button is used). The display also shows the number of the node the mouse is pointing to and the time when the display was last updated.

Using `xsp1info`, users can see how much of the machine is in use with EUIH jobs and see where their EUIH jobs are running. It also provides a convenient way to open `xterm`s on the SP1.



```
1 gropp 16 64
2 lusk 32 356
3 freitag 32 1273
4 levine 48 12
```

```
Sun Nov  7 14:02:59 CST 1993
```

Figure 3: A typical display of the EUIH usage of the SP1

### 2.2 Parallel Unix Tools

Contributed by **William Gropp and Ewing Lusk** Because each node of the SP1 is running a separate copy of AIX and contains a private disk (`/tmp` in our system), the user quickly discovers the need to (a) run various Unix tools (such as `ps`, `ls`, and `cp`) on a set of nodes and (b) to filter the output to show just the needed data. For example, a typical query is "is a file present?" On a uniprocessor, this is usually answered by using `ls filename`. On 128 processors, however, running `ls` generates so much output that it is difficult to be sure that the file was present on all processors (piping into `wc` is too severe; if the file is missing somewhere, the user wants to know where). We have provided programs that can help answer such questions in a scalable way, by providing both an easy way to make a specific inquiry across the parallel machine (e.g., `ppred`) and a graphical display of the output of these parallel commands (`pdisp`). We have produced prototype implementations of some parallel versions of popular Unix commands (see Table 3).

These routines (actually shell scripts) use recursive subdivision to execute the Unix commands in parallel. They are particularly important in distributing executable and shared input data to the local disks on the nodes. In the POE environment, these scripts can use the high-performance switch.

Table 3: Parallel unix commands

| Unix | Parallel |
|---|---|
| `cp` | `pcp` |
| `ps` | `pps` |
| `ls` | `pls` |
| `find` | `pfind` |
| `if (‘test‘) action` | `ppred` |
| `kill` | `pkill` |
| `a.out` | `prun ...  a.out` |

## 2.3 BlockSolve

Contributed by **Paul Plassmann and Lori Freitag**

BlockSolve [13] is a software library for solving large, sparse systems of linear equations on massively parallel computers. The matrices must be symmetric but may have an arbitrary sparsity structure. BlockSolve is a portable package that is compatible with several different message-passing paradigms including EUIH but not EUI.

## 2.4 Fortran M

Contributed by **Steven Tuecke**

Fortran M [6] is a small set of extensions to Fortran that supports a *modular* approach to the construction of sequential and parallel programs. Fortran M programs use *channels* to plug together *processes* that may be written in Fortran M or Fortran 77. Processes communicate by sending and receiving messages on channels. Channels and processes can be created dynamically, but programs remain deterministic unless specialized nondeterministic constructs are used.

Fortran M has been ported to the SP1, with runtime support added to allow communication via TCP/IP over either the switch or the ethernet. Currently we are limited to running on 64 nodes, since IP over the switch supports only up to 64 nodes.

Since Fortran M is a preprocessor that produces Fortran 77 code, it is heavily dependent upon the target machine's Fortran 77 compiler. On the SP1 we had few difficulties beyond the normal small differences in Fortran compilers.

Several Fortran M applications have been run on the SP1, including a parallel chromatography simulation and a parallel smog model.

## 2.5 Chameleon

Contributed by **William Gropp** Message passing is a common method for writing programs for distributed-memory parallel computers. Unfortunately, the lack of a standard for message passing has hampered the construction of portable and efficient parallel programs. In an attempt to remedy this problem, a number of groups have developed their own message-passing systems, each with its own strengths and weaknesses. Chameleon [10] is a second-generation system of this type. Rather than replacing these existing systems, Chameleon is meant to supplement them by providing a uniform way to access many of these systems. Chameleon's goals are to (a) be very lightweight (low overhead), (b) be highly portable, and (c) help standardize program startup and the use of emerging message-passing operations such as collective operations on subsets of processors. Chameleon also provides a way to port programs written using PICL or Intel NX message passing to other systems, including collections of workstations. This feature was used by the global climate model (see Section 3.4) to port to the SP1.

Chameleon ported to the SP1 with no problems other than the need to statically link Fortran programs. Both an EUI and EUIH port have been provided, as well as an IP port (using `p4` for the IP transport). The EUIH port provides a simplified startup mechanism that eliminates the need for having the user invoke the program with the shell script `cotb0`.

Chameleon includes a set of programs that test the communications performance of the system. The `twin` program (written by Scott Berryman and William Gropp) tests communication between pairs of processors, using a number of techniques to remove "occasional effects" such as timer interrupts and the effect of network activity from the timings. The message sizes tested are also chosen adaptively to capture discontinuities in the behavior of the message-passing system. One such discontinuity is shown in the EUI results at 128 bytes; EUI switches to a different protocol for longer messages that significantly adds to the latency of longer messages. This effect is shown in Figure 4. The DELTA results also show a discontinuity (at 480 bytes); this reflects the message packet size (minus the header) used on the DELTA. The performance for long messages for a variety of machines is shown in Figure 5. These results show that we can expect good performance compared to other MPPs for communication-intensive programs that use EUIH. The EUI data is from a preproduction version of EUI.

Additional programs test collective operations (`gop`), compute the relative speed of communication links (`tcomm`), test all of the links for correct operation (`stress`), and compute the bisection bandwidth (`bisect`). An option allows the programs to display in an X-window the amount of communication activity on a per-processor basis.

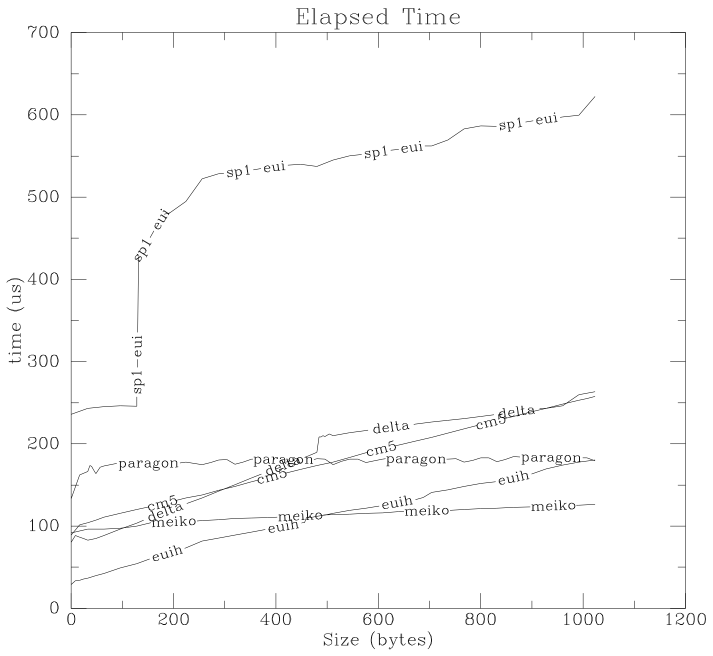


Figure 4: Communication performance for small messages for a variety of machines. All SP1 results use the switch.

## 2.6 Parallel Research on Invariant Subspace Methods

Contributed by **Christian Bischof and Xiaobai Sun**

Successive Band Reduction (SBR) is an approach for the orthogonally reduction of matrices to condensed form allowing for the use of matrix-matrix (BLAS-3) kernels. Special instances of SBR are the tridiagonal and Hessenberg reductions used in various eigensolvers. In addition, SBR supports general bandreduction, which is needed for banded eigenvalue scenarios. A modified SBR approach is also used in the PRISM (Parallel Research on Invariant Subspace Methods) project for the development of a scalable parallel eigenvalue solver.

The SBR code is intended for distributed-memory MIMD parallel machines. It uses the Chameleon programming system, and its support for parallel operations on disjoint nodes subsets is critical in exploiting the multiple levels of parallelism in the algorithm. The SBR code had initially been implemented on the DELTA.

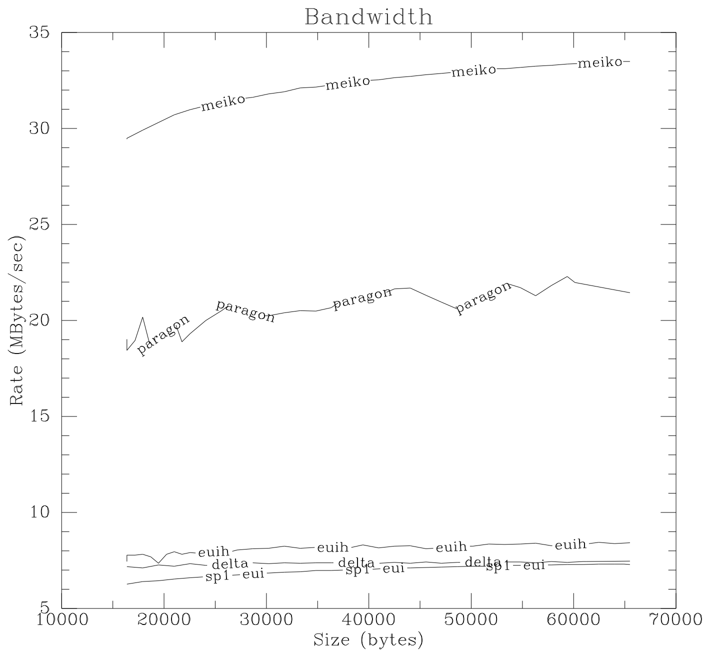


Figure 5: Communication performance for long messages for a variety of machines. All SP1 results use the switch. Data for the CM-5 is unavailable for messages of this length.

Table 4 shows some early performance results of the SBR code when applied to the Hessenberg reduction of a full matrix. "N" is the matrix size, "nb" is the block size used in the reduction of the matrix to "b" subdiagonal bands, "time" is the elapsed time in seconds, and "Gflops" is the sustained double-precision performance in Gflops/sec on the 16-node IBM SP1 system. These runs used the EUIH transport layer and the vendor-supplied 'blas.a' library.

Note that codes exploiting matrix-matrix kernels ($nb > 1$) perform much better than codes based on matrix-vector kernels ($nb = 1$). In particular, using a blocked algorithm, we can reduce a $3000 \times 3000$ matrix to a bandwidth of 15 roughly in the same time that we can tridiagonalize a $2000 \times 2000$ matrix, even though the former operation takes three times as many floating-point operations.

Table 4: Results for SBR code applied to a full matrix

| N | nb | b | Time | Gflops |
|---|---|---|---|---|
| 1000 | 1 | 20 | 19.9 | 0.233 |
| 1020 | 15 | 15 | 11.5 | 0.412 |
| 2000 | 1 | 20 | 164.1 | 0.227 |
| 2000 | 10 | 10 | 63.1 | 0.582 |
| 3000 | 1 | 10 | 512.5 | 0.246 |
| 3000 | 15 | 15 | 165.4 | 0.798 |
| 4000 | 20 | 20 | 346.0 | 0.849 |

## 2.7 MPI

Contributed by **William Gropp and Ewing Lusk** MPI (Message-Passing Interface) is an intended message-passing standard that is currently being developed by a broad group of massively parallel processor (MPP) vendors and users. A partial implementation of the standard as it stood in May 1993 was implemented and run on the SP1. In August we met with IBM researchers in Yorktown to propose a new implementation design [9]. The new design specifies a low-level device interface on which the MPI can be implemented portably. We have begun doing this, and at this stage (early September 1993) have implemented the basic point-to-point message-passing functionality of MPI. Meanwhile, IBM-Yorktown is proceeding with a high-performance implmentation of the device interface on the SP1 switch. This collaborative project is intended to provide (by November 1993) a high-performance implementation of major parts of the MPI specification on the SP1. To provide wide availability of MPI, we have also implemented the device interface portably, using our Chameleon and `p4` tools. These provides portability of parallel applications written with MPI from a variety of machines to the SP1, and in particular between the SP1 and workstation networks.

## 2.8 PCN

Contributed by **Steven Tuecke**

PCN [5, 7] is a system for developing and executing parallel programs. It comprises a high-level programming language, tools for developing and debugging programs in this language, and interfaces to Fortran and C that allow the reuse of existing code in multilingual parallel programs.

The network version of PCN (net-PCN) has been ported to the SP1. Runtime support has been added to PCN to allow communication to use TCP/IP over either the switch or the ethernet. Currently we are limited to running on 64 nodes over the switch, since IP over the switch only supports up to 64 nodes.

Several PCN applications have been run on the SP1, including the Massively Parallel Mesoscale Model (MPMM).

## 2.9 Portable, Extensible Tools for Scientific Computing (PETSc)

Contributed by **William Gropp**

PETSc is a package of routines aimed primarily at the solution of partial differential equations. PETSc is designed to match advanced algorithms to new and existing applications by taking an object-oriented approach to the design of the routines. For example, the iterative accelerators that are part of PETSc [11] have been designed to allow the user to specify all of the vector operations as well as matrix-vector product and preconditioning. Thus, these iterative methods can be used with nontraditional vectors, such as oct-trees or vectors distributed across a distributed-memory parallel computer. PETSc also includes a number of packages that aid in writing parallel programs. One of these is BlockComm, a package for communicating blocks of data between processors. Another is a parallel general (nonsymmetric) linear system solver using iterative methods [11]. Currently, parallel versions of conjugate gradient, conjugate gradient squared, BiCG-Stab, Freund's transpose-free QMR, generalized minimum residual (GMRES), Chan's transpose-free QMR, Chebychev, and Richardson are supported, along with a variety of preconditioners. A parallel nonlinear system solver is also available.

All of the parallel communication in PETSc is done with Chameleon. Porting PETSc, with the exception of the Fortran library problem, required no special effort. A version of PETSc that can take advantage of IBM's ESSL (when available) has been developed; the object-oriented nature of PETSc means that users can take advantage of these changes by relinking rather than rewriting their code.

## 2.10 p4

Contributed by **Ewing Lusk**

The p4 parallel programming system [1, 2, 3] currently runs on nearly all existing parallel computers and workstations. It has been used routinely on networks of RS/6000's. It was hoped that the RS/6000 version of p4 could be built unchanged on the SP1. At the moment, two different installations of p4 are maintained, one for the RS/6000's and one for the SP1.

The C part of p4 compiled and linked the first time on the SP1, using all parameters from the RS/6000 version. C programs compiled and linked for the RS/6000 network have run unchanged on the SP1. The phylogenetic tree application (see Section 3.5) is in this category.

After the switch and related software were installed, existing `p4` programs could immediately use the switch via the IP interface, even without EUI. This approach works in most situations, but some programs fail because of a bug in the IP/switch interface. This problem is currently being worked on by IBM.

When EUI arrived, `p4` was quickly ported to EUI, and then was ported easily to EUIH. The only changes necessary to move from the EUI version involved switching to Fortran calling sequences.

## 3 Applications

Successful port of a programming package to a parallel machine was once considered a sufficient test of the machine. However, as parallel machines are increasingly being acquired for production computing, it is more important to test them with ports of actual (as opposed to model) applications. Using our carefully developed portablilty tools, we were able to quickly port and run a wide variety of applications.

### 3.1 Computational Electromagnetics

Contributed by **Kimmo Forsman, William Gropp, Lauri Kettunen, and David Levine**

We have developed a parallel code to solve computational electromagnetic problems. Computational electromagnetics is widely used in industrial, research, and defense applications. However, many important problems are intractable with conventional techniques and vector supercomputers. For practical applications, the problem size must be dramatically increased, turnaround time must be reduced, and solution accuracy must be improved. A promising approach for overcoming these limitations is the use of integral equation methods (IEMs) implemented on massively parallel computers.

Integral methods offer several advantages over traditional finite-element techniques. Integral methods require no discretization of the nonactive regions, thus saving considerable time. Integral methods also take into account far-field boundary conditions automatically. In addition, integral methods remove the need for keeping track of which elements are moving or stationary.

However, integral equation methods give rise to dense matrices for which the solution time with direct methods scales as $O(n^3)$ and the memory requirements scale as $O(n^2)$, where $n$ is the number of degrees of freedom in the integral equation formulation. In addition, IEMs have a matrix definition time that scales as $O(n^2)$ and can be significant. Historically, this has meant that in practice IEMs have either taken too much computation time compared with finite-element programs to achieve the same level of accuracy, or have not used enough elements to obtain accurate solutions. It is our feeling, however,

Table 5: Solution time for four problems

| No. Proc. | 579 | 972 | 1629 | 2278 |
|---|---|---|---|---|
| 1 | 952 | 3127 | — | — |
| 2 | 507 | 1245 | — | — |
| 4 | 279 | 767 | 2571 | — |
| 6 | 236 | — | — | — |
| 8 | — | 454 | 1637 | 3416 |

that the advent of parallel processing computers will lead to renewed interest in IEMs for computational electromagnetic problems.

CORAL is a program that has been used to solve nonlinear three-dimensional magnetostatics problems using integral equation methods. The sequential version of CORAL uses LU decomposition (Crout's algorithm) with back substitution to solve the linear systems that arise each nonlinear iteration.

We have developed a parallel version of CORAL. The two key steps that we have parallelized are the matrix generation and the linear equation solver. We use the `Chameleon` message-passing system, developed at Argonne, for the message-passing parts of the program. The systems of linear equations are solved using the parallel iterative methods in the Parallel Simplified Linear Equation Solvers package (`PSLES`), which provides easy access to state-of-art methods for solving systems of linear equations (see Section 2.9). All of the results reported here were calculated using the Generalized Minimal Residual method with block diagonal preconditioning.

Preliminary timings on the IBM SP1 are shown in Table 5. This table shows solution time in seconds as a function of the number of processors for four different problems. The first column is the number of processors used. The other columns report total solution time (seconds) as a function of the number of processors used for solving four different nonlinear problems with matrices of order 579, 972, 1629, and 2278, respectively. These runs were made using an *unoptimized* version of the program and running the EUIH message-passing software with the high-performance switch.

## 3.2 Massively Parallel Mesoscale Model

Contributed by **John Michalakes**

MPMM is a fine-grained dynamic decomposition of the Penn State/NCAR Mesoscale Model version 5. Each set of four horizontal grid points is represented as a parallel process running under PCN (see Section 2.8), providing a transparent mechanism for redistributing load between physical processors. The work is being done in collaboration with the developers of the original Cray model (who are at NCAR). This program is used for real-time forecasting and climate prediction. Work on this code is continuing; the SP1 provides a better development environment than other parallel systems, in part because each node provides a full Unix environment.

## 3.3 Monte Carlo Calculations of Nuclear Ground States

Contributed by **Steven Pieper and Robert Wiringa**

With V. R. Pandharipande of the University of Illinois, Urbana, we are computing the properties of light (up to 40 neutrons and protons) nuclei using realistic two- and three-nucleon interactions. This involves developing many-body methods for reliably computing the properties of a nucleus for complicated forces that are strongly dependent on the spins and charge states of the nucleons. Unlike the Coulomb force used in atomic or condensed-matter calculations, there is no useful fundamental theory that tells us what this force is. We can partially constrain the two-body force by fitting nucleon-nucleon scattering data, but many-body calculations are required to test other properties of this force as well as the three-body interaction. Thus we are at the same time refining our knowledge of the forces and using it to make predictions about nuclei.

We make variational calculations, in which one assumes a form for the quantum-mechanical wave function describing a nucleus and then computes the energy of the nucleus for a given force model. This work involves computing multidimensional (12 to 120 dimensions) integrals using Monte Carlo methods. The integrand is expressed in terms of large complex vectors describing the spin and charge states of the nucleons. These calculations must be repeated many times to find the best set of parameters for the assumed form of the wave function. The longer a given calculation is allowed to proceed, the smaller the statistical error from the Monte Carlo integration, and hence the more refined the determination of the best parameters.

While the energy calculation is our main test of both the quality of the wave function and of the correctness of the force, we can also calculate various other properties, including low-lying excited states and a variety of low-energy reactions. Many of these are relevant to experiments that will be carried out at CEBAF, while others are predictions of astrophysically interesting cross sections that cannot be measured in the laboratory.

The first calculations being done on the SP1 are using a new nuclear interaction and are obtaining much better (when compared to experiment) results for the binding energy and density profile of oxygen than we had previously obtained. The better density results are specifically attributable to the detailed variational searches made possible by the SP1. The speed of the SP1 is, for the first time, making possible calculations of calcium (40 nucleons); work on this is in progress.

The SP1 is the first parallel processor that we have been able to use for our calculations. Previously we used single processors of the most powerful Cray computers available. Earlier parallel computers could not be used because of their small memories; our calculations require up to 65 megabytes of memory. The RS/6000 compiler provides good speeds on each node, allowing us to reach speeds of 48 Mflops on a single processor.

The Argonne package `p4` is used to implement the message-passing part of the program. The calculations proceed by having the master node do a random walk and send positions to the slaves. The calculation of the integrand at one position is very lengthy (up to 4 minutes) so it is easy for one master to keep all the slaves busy. The communication load is very low. Runs on 128 nodes achieve speedups of 123, or computational rates of 5.9 Gflops. A run using 160 nodes achieved 6.5 Gflops, but there were also other users on some of the nodes.

## 3.4 Parallel Community Climate Model

Contributed by **John Michalakes**

PCCM2 is a message-passing implementation of the NCAR Community Climate Model 2 (CCM2). In September 1993, PCCM was officially validated with respect to the sequential version of CCM. The SP1 was used extensively in the validation work because its nodes are identical to workstation platforms running the previously validated sequential version. The first validated version of PCCM ran on the SP1.

The model is patch decomposed in two horizontal dimensions. Spectral transport of all prognostic variables except moisture is accomplished by parallel FFTs in the zonal dimension and Gaussian quadrature in the meridional dimension, approximating Legendre transforms. The spectral transport mechanism of CCM2 is communication intensive because interchange of data is not confined to nearest neighbor. A semi-Lagrangian transport scheme is used for transport of moisture. Modules that compute atmospheric processes such as convection, radiative transfer, and precipitation are collectively known within the model as *physics*. Physics is perfectly parallel in PCCM because there are no horizontal data dependencies; however, physics does present the largest source of inefficiency from load imbalance.

PCCM2 is implemented using Chameleon through a compatibility library to PICL, the message-passing package under which the code was originally developed. Before being run on the IBM SP1, PCCM2 was run on the Intel Touchstone DELTA and Paragon computers.

PCCM currently runs at approximately 650 Mflops on the full SP1 (128 processors) communicating over the EUIH switch interface. Figure 6 shows the distribution of run time over the three main components of the code: spectral transport, semi-Lagrangian transport, and physics. Interprocessor communication accounts for most of the time spent in the forward and inverse FFTs, the parallel vector-sum (part of the Legendre approximation), and initialization for the semi-Lagrangian transport (SLTINI). The largest computational part of the code is physics, and the effect of load imbalance can be seen as well. Nearly a quarter of the elapsed time, however, is not at present accounted for. Work to identify the source of this problem, to tune computational performance, and to improve interprocess communication is continuing.

The work is being performed under the directed portion of the Department of Energy CHAMMP initiative and is the collaborative effort of Argonne, Oak Ridge National Laboratory, and NCAR. The model is used for climate prediction.

## 3.5 Phylogenetic Trees

Contributed by **Ross Overbeek**

Gary Olsen, along with a group at ANL and Hideo Matsuda of Kobe University, decided to create a fast implementation of a maximum likelihood algorithm for constructing phylogenetic trees from an alignment of sequence data. This program, called fastDNAml, now runs on a wide class of uniprocessors, on networks of workstations, and on several of the massively parallel systems (most notably, the DELTA).

The algorithm used in this application uses automatic load balancing and very large-grain parallelism, so that the efficiency of the message-passing system in not significant. It currently does not use any of the switch-based transport mechanisms. This application has consumed by far the largest number of hours on the SP1 since its installation.

The Ribosomal Database Project at the University of Illinois at Urbana-Champaign [15] distributes alignments of small subunit ribosomal RNA (rRNA) sequences from both prokaryotic microorganisms and eukaryotes. In order to better understand the organisms themselves and the evolution and function of the ribosome, we wished to infer a consistent, high-quality phylogenetic tree relating all of these sequences. Such a phylogenetic tree plays a fundamental role in supporting interpretation of molecular sequence data.

Phylogenetic tree inference based on maximum likelihood is appealing from both biological and statistical perspectives. Felsenstein [4] has written a computer program that implements such a method. However, maximum likelihood is computationally demanding, so the program had been used only for inferring relationships among small numbers of sequences (up to about 20). Analyses indicated that execution time (for small trees) rises as the third power of the number of sequences.

Gary Olsen and Carl Woese of the Ribosomal Database Project at the University of Illinois at Urbana have been creating an alignment of the rRNA from the small subunit of the ribosome. This alignment has become one of the fundamental tools for phylogenetic research.

Working closely with Olsen and with Carl Woese (co-leader of the Ribosomal Database Project), we are trying to construct a phylogentic tree for all of the organisms included in the current alignments. Currently, this number is approaching 2000. Our research is focusing on developing on a number of critical issues:

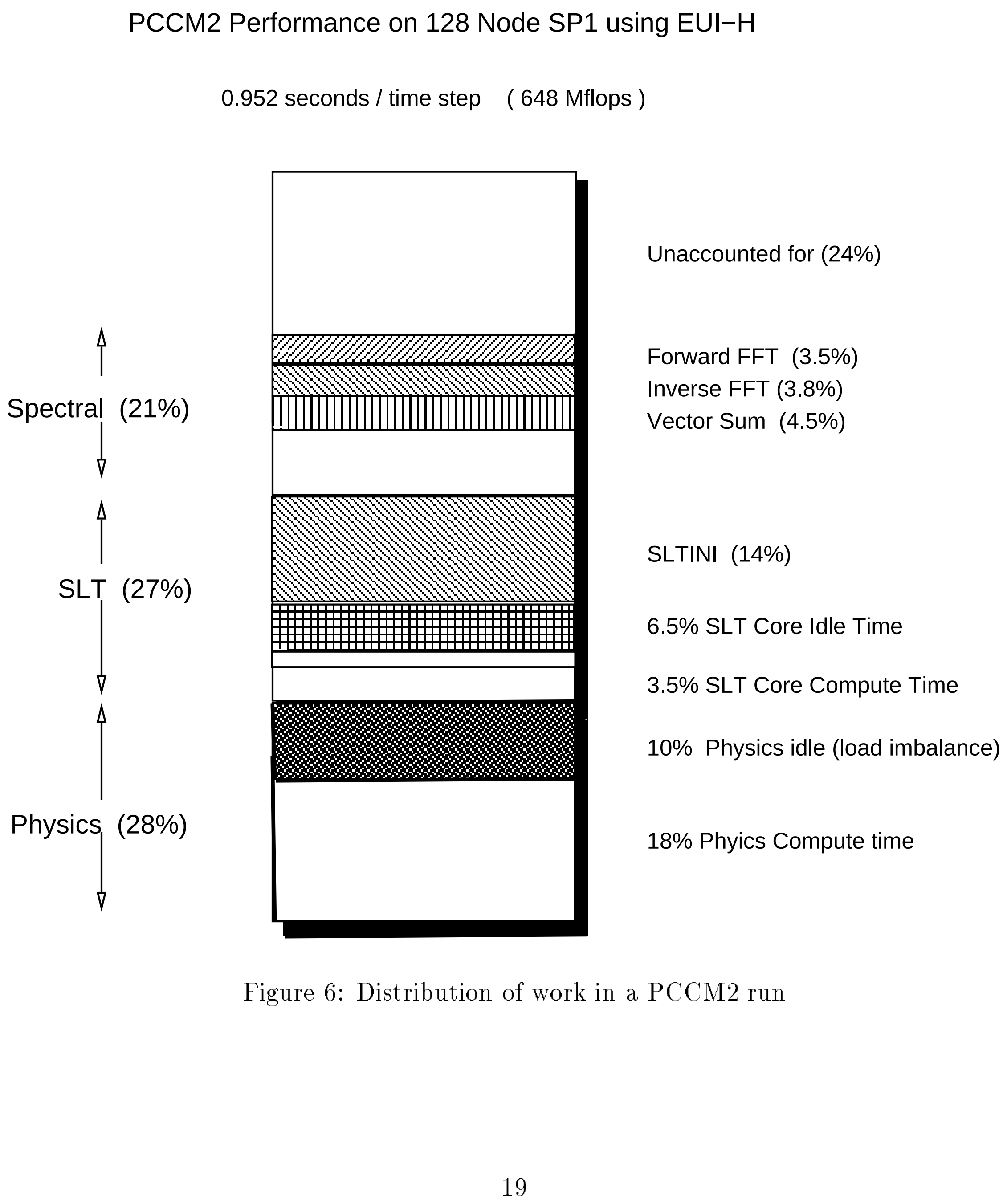


Figure 6: Distribution of work in a PCCM2 run

1. Phylogenetic computations are sensitive to the rate of change in specific columns. Gary Olsen has developed a set of tools to reflect varying rates of change. We need to first verify that this tool does, in fact, improve the sensitivity of a standard maximum likelihood computation. The task of gathering data to evaluate this question has just been completed on the SPI at Argonne. The computation required execution of over 450 runs, each of which consumed between 12 and 36 hours on single nodes. We are now evaluating this data and plan to write up the results as soon as possible. Our intent is to establish the basic utility of the approach that Olsen has implemented in his fastDNAml tools.

2. We are also establishing a basic algorithm for creating relatively huge trees. It is not practical to simply make one huge run and produce a reliable tree. Rather, the tree must be computed in steps. First, we have created an initial tree composed of 473 organisms. We then developed a tool for sequentially inserting new sequences into the tree. This produces an "initial tree" that must then be "optimized" by performing thousands of local maximum likelihood computations (which can produce local rearrangements within the tree).

3. Finally, the tree produced by maximum likelihood is subjected to critical analysis by other experts in the field, specific questionable areas are isolated, and a detailed analysis is done of these locations. This analysis is done using a variety of techniques and can be used to establish the limitations and advantages of the maximum likelihood tool that is in development.

We have developed both sequential and parallel versions of the fastDNAml package; the sequential version is now being distributed through the RDP server, and we have helped a limited set of institutions instal the parallel version (which is based on the `p4` package of routines for writing portable parallel programs [1]).

The phylogenetic tree that we are developing is of major scientific interest. It is the culmination of decades of work to develop the technology, gather the rRNA sequences, and careful align them for analysis. If it can be established that the maximum likelihood approach is actually more accurate, then bringing the required computational resources will produce a clear instance in which massive computational resources do advance a fundamental area of science.

### 3.6 Superconductivity—Elastic String Model

Contributed by **David Levine and Gary Leaf**

We have developed a code for the numerical simulation of the planar motion of a one-dimensional elastic filament (single vortex) under tension, to investigate the properties of the vortex-glass state in high-temperature, Type-II superconductors. The computational problem requires the time integration of a stochastic evolution equation; ensemble averages are obtained by considering the long-time behavior of the solution for a large number of realizations. The objective of the numerical simulations is to measure the resulting "average" velocity of the filament as a function of the applied force.

In the study of the elastic filament model, we observed avalanche-type behavior when the applied forces are in the neighborhood of a critical transition value. Such behavior is characteristic of self-organized criticality phenomena. We have developed other elastic filament models to explore this phenomenon. These studies also require the accumulation of statistics from a large number of events. Each event involves the solution of a stochastic differential equation subject to a random initial perturbation. In the appropriate parametric state space, the system will enter a steady state for a sufficiently large number of events. The calculations are characterized by a large number of independent calculations that can occur simultaneously, a situation ideally suited for coarse-grained parallelism.

The most difficult calculations for the elastic filament model occurred for very small applied forces when the system is in a "glassy" or "creep" state characterized by very slow dynamics which require extremely large amounts of computer time to establish the asymptotic behavior. To further study this, we developed an alternative model based on a static tilted potential, characteristic of creep motion. The calculation is characterized by large numbers of ensembles ($\approx$ 12,000) each corresponding to a random realization of a potential tilted by an applied force. For small forces, a very large number of spatial points ($\approx$ 75,000) are needed to resolve the potentials.

In all three of these cases, the parallel approach used is coarse grained. Since each realization is both time consuming and independent of the other realizations, we have been able to run a large number of these jobs in parallel. This work was initially started using a BBN TC2000 and a Sun SPARC workstation network. We have since ported the programs to the SP1 system, where we have found a significant improvement in execution time and the number of processors available. For some of the most difficult calculations, with very small applied forces, we have reduced solution time from approximately five days on Sun SPARC workstations to approximately 17 hours on the SP1 system.

## 3.7 Vortex Dynamics in High-Temperature Superconductors

Contributed by **Gary Leaf and David Levine**

We are using the time-dependent Ginzburg-Landau (TDGL) equation for the numerical simulation of vortex dynamics and phase transitions in high-temperature, Type-II superconductors. Effects of external currents, material defects, and thermal fluctuations are incorporated into our model. We are interested in the formation and subsequent evolution of magnetic flux vortices and the influence of random impurities on vortex pinning.

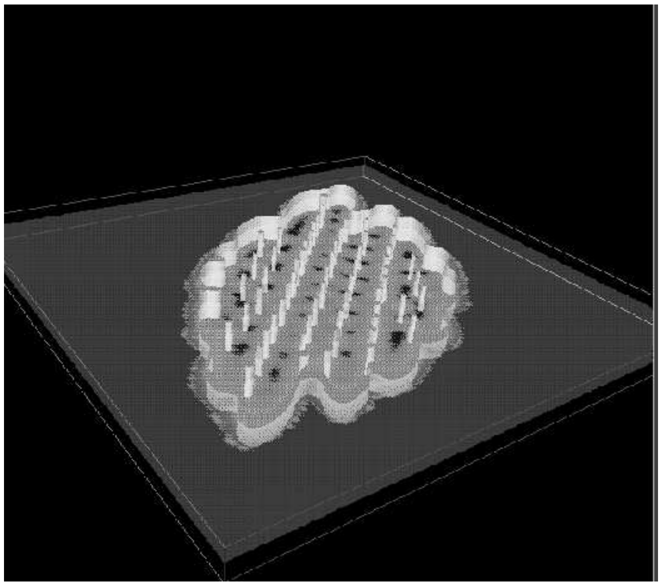


Figure 7: Solution at 5000 steps

We have developed a program to solve the three-dimensional TDGL equations for a superconducting cube in a fixed external magnetic field on bounded domains. The code solves numerically for the complex-valued order parameter as well as the magnetic field within a bulk superconducting material.

The three-dimensional domain is subdivided into an array of cells. We identify the order parameter with the vertices of each cell and the gauge field with the edges (links) of each cell. The resulting equations of motion are solved by using a single time step, forward Euler procedure. The primary data structures used are four complex, three-dimensional arrays whose values are updated each time step according to the equations of motion.

Figure 7 and Figure 8 show the early time dynamics of vortex trapping by twin boundary defects in a high-temperature, Type-II superconductor. The dynamics were initiated from a doped state in the presence of an external magnetic field whose strength was adjusted so that the material is in the mixed state. The material was modeled with a pair of planar defects (twin boundaries) running diagonally through the sample. Inhomogeneties in the twin boundaries were modeled with random point defects imbedded in the twin boundaries. The phenomenological model used was a three-dimensional, time-dependent Ginzburg-Landau system.

The figures show the early time evolution of the vortices from an initial seed. We see the twin boundaries pinning the vortices and the consequent alignment of vortices trapped

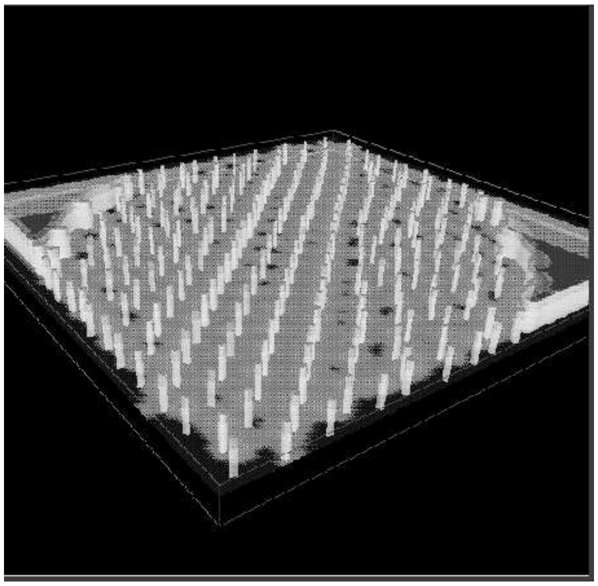


Figure 8: Solution at 10000 steps

between the twin boundaries.

To parallelize the program, we partition the array of cells (grid) among the processors. Each processor is responsible for updating all the cells in the subgrid contained in its memory. The update step for each cell requires values from neighboring cells. Because of the array decomposition, neighbors of some of the cells that a processor has require values from cells stored in other processor's memories. To communicate these values between processors, we use the BlockComm package developed by William Gropp (see Section 2.9). With BlockComm, the programmer specifies, via function calls, the decomposition that will be used. BlockComm produces an internal description of this decomposition. Whenever data from other processors is needed, the programmer calls the BlockComm routine `BCexec`; this manages all of the communication needed to provide the data.

Preliminary timings on the IBM SP1 are shown in Table 6. The first column is the number of processors used. The other columns report time per iteration (seconds) as a function of the number of processors for grid sizes of $130 \times 130 \times 10$, $258 \times 258 \times 10$, and $514 \times 514 \times 10$, respectively. EUIH in a column indicates IBM's EUIH message-passing software was used in conjunction with the high-performance switch. For comparative purposes, the smallest problem was also run using a version of the `p4` message-passing software that used Unix sockets to communicate over an external Ethernet network that also connects the processors.

Table 6: Solution times (sec/iter) for three problems

| No. | $130^2 \times 10$ | | $258^2 \times 10$ | $514^2 \times 10$ |
|---|---|---|---|---|
| Procs | EUIH | Sockets | EUIH | EUIH |
| 1 | 7.67 | 15.3 | — | — |
| 2 | 4.00 | 8.19 | — | — |
| 4 | 2.16 | 4.88 | 8.36 | — |
| 8 | 1.22 | 3.89 | 4.66 | 18.67 |
| 16 | .60 | 3.69 | 2.19 | 8.27 |
| 32 | .34 | 4.11 | 1.18 | 4.43 |
| 64 | .20 | 4.36 | .65 | 2.23 |
| 96 | .17 | — | .48 | 1.57 |

Table 7: Comparison of the CPU and elapsed times for three different cases: (1) the Intel Touchstone DELTA, (2) the IBM SP1 running EUIH, and (3) the IBM SP1 running `p4`. All cases are for 100 BFGS iterations with a constant global domain size of $32 \times 32 \times 32$.

| Number of | Intel DELTA | | IBM SP1 (EUIH) | | IBM SP1 (`p4`) | |
|---|---|---|---|---|---|---|
| Processors | CPU | Elapsed | CPU | Elapsed | CPU | Elapsed |
| 1 | – | – | 203.89 | 205.46 | 203.90 | 205.29 |
| 2 | 307.67 | 308.00 | 86.09 | 86.75 | 81.26 | 118.44 |
| 4 | 160.26 | 160.00 | 37.54 | 37.61 | 33.29 | 112.46 |
| 8 | 79.33 | 80.00 | 21.33 | 21.50 | 17.94 | 196.09 |
| 16 | 43.13 | 43.00 | 12.75 | 12.97 | – | – |

### 3.8 Superconductivity—Vortex Structures

Contributed by **Mario Palumbo and Paul Plassmann**

We have developed a parallel code that uses the limited-memory BFGS algorithm [14] to find optimal vortex solutions within the three-dimensional anisotropic Ginzburg-Landau model. Our implementation is capable of considering arbitrary field orientation as well as various types of random and correlated disorder. This code is currently being used to study various properties of uniaxial superconductors such as the lower critical field and the anomalous "vortex-chain" state.

The parallelization was achieved through a simple three-dimensional domain decomposition scheme in which the global domain is partitioned across an arbitrary number of processors. The communication between processors is carried out using the Chameleon par-

Table 8: Comparison of the CPU and elapsed time for the cases: (1) the Intel Touchstone DELTA, (2) the IBM SP1 running EUIH, and all cases are for 100 BFGS iterations with a constant local domain size of $16 \times 16 \times 16$.

| Number of | Intel DELTA | | IBM SP1 (euih) | |
|---|---|---|---|---|
| Processors | CPU | Elapsed | CPU | Elapsed |
| 1 | 73.71 | 74.00 | 15.91 | 16.01 |
| 2 | 76.27 | 76.00 | 17.71 | 17.90 |
| 4 | 77.85 | 78.00 | 19.33 | 19.49 |
| 8 | 79.33 | 80.00 | 21.46 | 21.57 |
| 16 | 80.57 | 81.00 | 22.75 | 22.98 |

allel software package (see Section 2.5). The portability of the Chameleon primitives has allowed us to run the code on a variety of parallel platforms, using several different parallel communication paradigms, without any coding changes. Performance comparisons for a selection of these cases are provided in Tables 7 and 8. Note the superlinear speedup in the SP1 results in Table 7; this is most likely caused by cache effects. Also note the CPU time column from the SP1 (`p4`) results. These show very good performance in a time-shared environment, even though the elapsed time performance is relatively poor.

Table 8 shows the performance as the local domain size is held fixed and the number of unknowns grows proportionally with the number of processors. These suggest that $16 \times 16 \times 16$ (only 4096 mesh points) local domain is too small for the SP1. This result is consistent with the faster speed of the processors with respect to the communication than for the Intel DELTA, and emphasizes why the large per-node memory is an important feature of the SP1.

### 3.9 Parallel Theorem Prover

Contributed by **William McCune and Ewing Lusk** We were able to port our parallel distributed-memory theorem prover `dac` (distributed associative-commutative theorem prover) to the SP1 with no problems. We initially developed `dac` on the Symmetry and a Sun network using `p4`. Single-node performance was excellent, despite the lack of floating-point operations in `dac`.

## 4 Summary

Having a full, running Unix OS on each node allows for easy ports. We have also taken advantage of the ability to reboot individual nodes without disturbing the others. Furthermore, the use of portability layers (Chameleon, PCN, and `p4`) let us port applications quickly and allowed us to become familar with the programming environment on the SP1 in the context of significant applications.

We have been able to get a wide variety of applications running with some impressive performance and with relatively little additional work, demonstrating that it is possible to write portable, efficient parallel programs.

## References

[1] James Boyle, Ralph Butler, Terrence Disz, Barnett Glickfeld, Ewing Lusk, Ross Overbeek, James Patterson, and Rick Stevens. *Portable Programs for Parallel Processors.* Holt, Rinehart, and Winston, 1987.

[2] Ralph Butler and Ewing Lusk. Monitors, messages, and clusters: The p4 parallel programming system. *Journal of Parallel Computing.* To appear (Also Argonne National Laboratory Mathematics and Computer Science Division preprint P362-0493).

[3] Ralph Butler and Ewing Lusk. User's guide to the p4 parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, October 1992.

[4] J. Felsenstein. Phylip manual version 3.3. Technical report, University of California, Berkeley, Calif., 1990.

[5] Ian Foster, Robert Olson, and Steven Tuecke. Productive parallel programming: The PCN approach. *Scientific Programming*, 1(1):51–66, Fall 1992.

[6] Ian Foster, Robert Olson, and Steven Tuecke. Programming in Fortran M. Technical Report ANL-93/26, Revision 1, Argonne National Laboratory, 1993.

[7] Ian Foster and Steven Tuecke. Parallel programming with PCN. Technical Report ANL-91/32, Rev. 2, Argonne National Laboratory, 1991.

[8] William Gropp. Early experiences with the IBM SP-1. Technical Report ANL/MCS-TM-177, Argonne National Laboratory, 1993.

[9] William Gropp and Ewing Lusk. An abstract device definition to support the implementation of a high-level message-passing interface. Technical Report MCS-P342-1193, Argonne National Laboratory, 1993.

[10] William D. Gropp and Barry Smith. Chameleon parallel programming tools users manual. Technical Report ANL-93/23, Argonne National Laboratory, March 1993.

[11] William D. Gropp and Barry Smith. Users manual for KSP: Data-structure-neutral codes implementing Krylov space methods. Technical Report ANL-93/30, Argonne National Laboratory, August 1993.

[12] Michael T. Heath and Jennifer Etheridge Finger. Visualizing performance of parallel programs. *IEEE Software*, 8(5):29–39, September 1991.

[13] Mark T. Jones and Paul E. Plassmann. An efficient parallel iterative solver for large sparse linear systems. In *Proceedings of the IMA Workshop on Sparse Matrix Computations: Graph Theory Issues & Algorithms*, Minneapolis, 1991. University of Minnesota.

[14] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large-scale optimization. *Math. Prog.*, 45:503–528, 1989.

[15] G. J. Olsen, R. Overbeek, N. Larsen, and C. R. Woese. The ribosomal database project: Updated description. *Nucleic Acids Res.*, 19:4817–4817, 1991.