ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-177

Early Experiences with the IBM SP-1

Edited by

William Gropp

Mathematics and Computer Science Division

Technical Memorandum No. 177

June 1993

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

Contents

Early Experiences with the IBM SP-1

Edited by

William Gropp

Abstract

The IBM SP-1 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 early efforts. Despite the newness of the machine and the lack of a fast interprocessore switch (part of the SP-1 but not yet available for our machine), every code that we attempted to port ran on the SP-1 with little or no modification. The report concludes with a discussion of expectations for the fast interconnect.

Cover Picture

This is a density plot of the atmospheric pressure over the United States, produced by the Massively Parallel Mesoscale Model running on the SP-1. This program is a PCN implementation of the Penn State/NCAR Mesoscale Model version 5. The units on the histogram are millibars $\times 100.0$. The domain is the continental United States, plus southern Canada and northern Mexico. The Rockies stand out in dark grey. While it is a little difficult to make out the east coast, the Appalachians do appear in light grey. Thanks to John Michalakes for this picture.

Contributors

Richard Feldmann
William Gropp
Gary Leaf
William McCune
Ross Overbeek
Steven Tuecke
Lori Freitag
David Levine
Ewing Lusk
John Michalakes
Paul Plassmann

1 Introduction

The IBM SP-1 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

- large memory per node (128 MBytes),
- local disks on each node (1 GByte),
- full Unix on each node (IBM AIX),
- high-performance nodes,
- 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 SP-1 in the first few weeks after it was delivered. Since this early system did not include the fast interprocessor switch (to be delivered by the end of May), these results, particularly the performance results, should not be taken as representative of performance of the SP-1. Instead, these results indicate the state of software environments for the SP-1 and the power of the software packages that have been developed for portable parallel programming.

The software packages and tools are as follows:

BlockSolve Parallel sparse, symmetric linear systems

Chameleon Lightweight and portable message-passing system

MPI Message-passing interface draft standard

PCN Program Composition Notation (a coordination language)

PETSc Portable, extensible tools for scientific computing

P4 Portable message-passing and shared-memory library

The applications (all parallel) are as follows:

Community climate model Global climate model

Mesoscale weather model Continent-sized weather model

Phylogenetic tree Program to construct phylogenetic trees from sequence data

Protein folding Program to grow a protein and fold it

Superconductivity Modeling of flux vortices in high-temperature superconductors (two applications)

Theorem prover Distributed associative-commutative theorem prover

Because the ANL SP-1 does not yet have the fast interconnect, these application ports are primarily a test of the software environment. However, since all of these applications are built using one or more of the portable parallel programming packages, successful ports of those packages immediately give ports of these applications. In addition, these ports will allow us to address the question of how necessary a fast interconnect is (some workers have suggested that farms of machines are adequate), by comparing the results with applications before and after the fast switch is installed.

One common problem that many groups experienced had to do with linking Fortran programs on the SP-1. Because the Fortran run-time libraries were not available on each SP-1 node, each group had to link with the options `-bnso -bI:/lib/syscalls.exp`. Further, since these are not documented on the `xlf` man page, there was some delay in porting some applications. Another problem with Fortran was caused by the fact that, by default, the external names produced by Fortran are not distinguishable from those produced by C; using the command line switch to `xlf` to add the trailing underscore common on many Unix systems is not always a workable solution, particularly for library developers.

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.

2 Programming Packages and Tools

This section describes the programming packages that support the applications that have been ported to the SP-1. The packages include a numerical library (BlockSolve) 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 SP-1.

2.1 BlockSolve

Contributed by **Paul Plassmann and Lori Freitag**

BlockSolve [5] 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

Table 1: Results for BlockSolve on the SP-1 with 1-D partitioning (times are CPU times and do not include communication times)

1-D PARTITIONING

Num. Procs.	Local Grid Sz.	Total Grid Sz.	Time Init.	Time Fact.	Time/Iter.
1	512	512	.050	2.30	.0100
2	512	1024	.050	2.89	.0167
4	512	2048	.060	2.99	.0186
8	512	4096	.060	3.02	.0214
16	512	8196	.070	3.00	.0214

Table 2: Results for BlockSolve on the SP-1 with 3-D partitioning (times are CPU times and do not include communication times)

3-D PARTITIONING

Num. Procs.	Local Grid Sz.	Total Grid Sz.	Time Init.	Time Fact.	Time/Iter.
1	512	512	.050	2.30	.0100
2	512	1024	.040	2.79	.0157
4	512	2048	.040	3.22	.0229
8	512	4096	.080	3.92	.0286
16	512	8192	.080	4.28	.0357

sparsity structure. BlockSolve is a portable package that is compatible with several different message-passing paradigms.

For the results presented here we are using the p4 communication package (through Chameleon) on the IBM SP-1. The local problem is based on the 3-D seven-point stencil on an $8 \times 8 \times 8$ grid and remains fixed as the number of processors increases. For the first set of test problems, the local grids are connected end to end in one dimension. For the second set, the local grids are connected in all three directions for eight and sixteen processors. We expect that for larger local problems the time per iteration will be roughly constant. The results for measuring the CPU time only are shown in Tables 1 and 2. Results for the elapsed time (including communication times and potential interference from other running jobs) are shown in Tables 3 and 4.

Table 3: Results for BlockSolve on the SP-1 with 1-D partitioning (times are elapsed time, averaged over 5 runs)

1-D PARTITIONING					
Num. Procs.	Local Grid Sz.	Total Grid Sz.	Time Init.	Time Fact.	Time/Iter.
1	512	512	.0860	2.41	.0216
2	512	1024	.0704	5.43	.0373
4	512	2048	9.076	14.24	.0660
8	512	4096	9.092	17.53	.1301
16	512	8196	11.167	26.43	.2630

Table 4: Results for BlockSolve on the SP-1 with 3-D partitioning (times are elapsed time, averaged over 5 runs)

3-D PARTITIONING					
Num. Procs.	Local Grid Sz.	Total Grid Sz.	Time Init.	Time Fact.	Time/Iter.
1	512	512	.0860	2.41	.0216
2	512	1024	.0704	5.43	.0373
4	512	2048	.0896	16.21	.4705
8	512	4096	.1508	28.25	.4088
16	512	8192	8.4649	44.58	.6019

2.2 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 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.

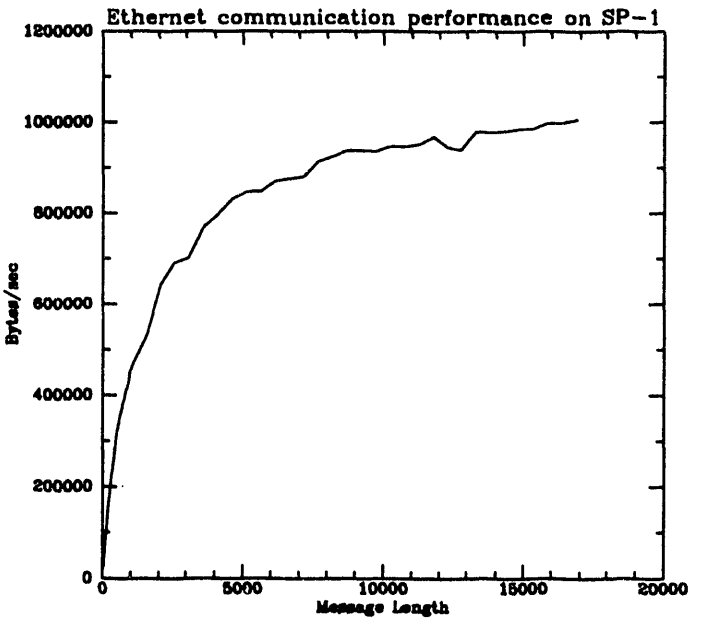

Figure 1: Communication performance for the ethernet links in the SP-1

This feature was used by the global climate model (Section 3.2) to port to the SP-1.

Chameleon ported to the SP-1 with no problems other than the need to statically link Fortran programs. Chameleon includes a set of programs that test the communications performance of the system. Bearing in mind that the tested system does not have the switch (all communications are over ethernet), the performance is quite reasonable (better than our Sun network), as is shown in Figure 1.

2.3 MPI

Contributed by **William Gropp and Ewing Lusk**

MPI is a 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 point-to-point routines of the current (May 1992) MPI draft standard has been implemented and run on the SP-1. This implementation is designed to give a vendor maximum flexibility in matching the MPI operations to vendor-specific hardware and/or software. This instance of the implementation builds on Chameleon, using the **p4** transport layer. We expect it to be easy to port MPI directly on top of EUI-1 or lower level communication primitives.

2.4 PCN

Contributed by **Steven Tuecke**
The RS/6000 network version of PCN (net-PCN) [3, 4] worked on the SP-1 with no modifications. This version of PCN uses TCP/IP (i.e., sockets) for communication between nodes and uses rsh for node startup.

Initially, PCN programs were compiled on an RS/6000 220 and the executables copied to the /u filesystem which is mounted on all of the SP-1 nodes. These programs worked perfectly. Then, to test the stability of the C compiler and environment on the SP-1 nodes, we rebuilt the PCN compiler from scratch on an SP-1 node in /u with no difficulties.

To work around the problem of needing to statically link the Fortran libraries, the PCN compiler driver on `spgw` (the SP-1 gateway machine) was modified to pass extra arguments to the linker when linking with Fortran subroutines, so as to force static linking of the Fortran libraries.

2.5 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 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.

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) is being developed; the object-oriented nature of PETSc means that users can take advantage of these changes by relinking rather than rewriting their code.

2.6 Porting the p4 Parallel Programming System to the SP-1

Contributed by **Ewing Lusk**
The p4 parallel programming system [2, 1] 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 SP-1. This would

have been true except for the shared library problem for Fortran programs. For this reason the Fortran part of p4 has not yet been ported, although this should happen soon. At the moment, two different installations of p4 are maintained, one for the RS/6000's and one for the SP-1.

The C part of p4 compiled and linked the first time on the SP-1, using all parameters from the RS/6000 version. C programs compiled and linked for the RS/6000 network have run unchanged on the SP-1. The phylogenetic tree application (Section 3.3) is in this category.

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.

3.1 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 (Section 2.4), 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.

3.1.1 SP-1 Difficulties Encountered

MPMM is a hybrid code, composed of top-level PCN code to manage parallelism between core modules of native Fortran. The PCN part of the port was simply a matter of recompiling. A day or two earlier Steven Tuecke had ported PCN to the SP-1. The port of the Fortran code was more difficult, though still manageable.

There are a number of troublesome inconsistencies between IBM's implementation of Fortran and other Unix implementations.

The major problem we encountered was `xlf`'s lack of an -extend option to relax the column 72 restriction on source lines. We have a number of automatic source code transformations built into the Fortran parts of MPMM to facilitate the fine-grained decomposition of the code. These source transformations can generate longer lines, and they do not respect the column 72 restriction. `Xlf` does have a free-format option, but this required some

very radical syntactic changes to the code. Our solution was to write an additional source transformation that, as a last step before entering the compiler, breaks the long lines into continuations, respecting column 72.

3.1.2 SP-1 Benefits

The SP-1 has two principal benefits: an excellent programming environment and a large RAM and virtual memory.

Notwithstanding the relatively minor difficulties we encountered with `xlf` incompatibilities, the RS/6000 programming environment is vastly superior to the environment we have experienced with the Intel machines. There are no cross compilers; a program compiled on any RS/6000 runs on the SP-1. `Xlf` has a rich set of options to assist in debugging, such as array bounds-checking and floating-point traps. A debugger exists, which gives it an immediate advantage over the Intel environment; the fact that it also works well is a welcome bonus.

Each node of the SP-1 has a prodigious amount of physical memory: 128 Mbytes, augmented by an additional amount of virtual memory. This feature makes it possible to run large problems on small numbers of nodes or even a single node if performance is not the main consideration. In particular, this is very useful for debugging. The large amounts of memory also allow programs to set up large buffers that can be used for asynchronous I/O.

3.2 Parallel Community Climate Model

Contributed by **John Michalakes**

PCCM2 is a message-passing implementation of the NCAR Community Climate Model 2 (CCM2). The model is patch decomposed in two horizontal dimensions. Spectral transport of prognostic variables 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.

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.

PCCM2 is implemented using a message-passing library, PICL. Prior to the IBM SP-1, PCCM2 has run on the Intel Touchstone Delta and Paragon computers.

3.2.1 SP-1 Difficulties Encountered

We encountered two difficulties: lack of a message-passing library and shortcomings with the the IBM Engineering and Scientific Subroutine Library (ESSL).

Since the machine is new, there is no implementation of PICL for the SP-1. Bill Gropp kindly generated a PICL/Fortran compatibility library for his Chameleon package (Section 2.2), and we have run PCCM2 on the SP-1 using that library.

ESSL is available on the SP-1, and we have begun converting the FFTs in CCM to use the library routines in the hope that processor performance will be further improved. In general, the availability of this library is a plus, though there have been some difficulties. One feature of the ESSL FFT routine that could be a problem is the requirement that it be reinitialized for even minor changes in the data being transformed. CCM calls forward and inverse FFTs a number of times each time step, with different numbers of vectors to be transformed at each call. There is no other change than the number of vectors, yet a separate initialization is required. This may produce a hit against the potential performance gain from using the library. The other shortcoming of the library is that there are no parallel distributed memory implementations of the FFT or other routines. Therefore, the ESSL FFT can be used on PCCM2 only if the zonal dimension of the model is undecomposed.

3.2.2 SP-1 Benefits

The biggest advantage of the SP-1 for the climate model is processor performance. Running on a single node of the SP-1, PCCM2 achieved a sustained performance of 21 Mflops at T21 resolution, as compared with 3-5 Mflops per node on i860 nodes of the Delta.

3.3 Phylogenetic Trees

Contributed by **Ross Overbeek**

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.

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).

Gary, Rusty Lusk, and I decided to put the program on the SP-1 and to attempt to use it to investigate a specific scientific issue: Where do the mitochondria fit within the alpha purple bacteria? This is a serious issue; the approximate answer produced by a fast insertion

algorithm that we developed for use on workstations conflicted with the opinion of Woese (who is generally acknowledged as a world authority on phylogeny of microorganisms).

The basic goal of our program is to construct a tree with a maximum likelihood of generating the observable data. The set of trees to be searched is huge. Hence, one uses heuristic optimization algorithms to search for a "best" tree. Our version is sensitive to the order in which individual sequences are placed into the tree. We compensate for this by using many random orders, hoping that the true global optimum will be revealed from multiple attempts. Thus, the degree of confidence that one can feel about the computed tree is directly related to the number of attempts made to locate an optimum value.

An overnight run involved to construct a tree from 47 aligned rRNA sequences produced 72 different outputs based on random orderings of the sequences. The best value occurred 3 distinct times, and the placement of the mitochondrial sequence within the alpha purple was substantially different from the approximate position computed earlier.

My early reactions to the machine are as follows:

1. It is fast. Using 16 nodes, each run took roughly seventeen minutes each; these would take substantially longer (on the order of days) on single Suns.

2. The software environment is relatively good (infinitely better than the Delta). Unix works. Unlike the Delta, basic tools such as "head" work, Emacs is available, and long lines do not produce erratic behavior.

I look forward to making a large effort to resolve a number of critical phylogenetic questions during the next 3–4 months using the machine under the guidance of Olsen and Woese.

3.4 Protein Folding by the Study of Hydrophilic and Hydrophobic Loops

Contributed by **Richard Feldmann (NIH), and Ewing Lusk**

The program `Lfold` grows a protein in steps of one amino acid at a time. There are 1000 cycles between each amino acid addition. The protein TIM (Triose Phosphate Isomerase, the most prevalent protein architecture) is 248 amino acids long, so synthesis is complete at about 247,000 cycles. We believe that if the program can fold TIM, it will be capable of folding any protein. A cycle is one random examination of each hydrophobic loop in the protein to see whether this hydrophobic loop can move. The moves are a sort of Feynman exchange: that is, the existence of a hydrophobic loop at the head of a pair of hydrophilic loops catalyzes the exchange of the hydrophilic loop pair.

Folding then is the interaction of the backbone hydrophilic water loops (which are common to every amino acid except PRO) with the hydrophilic water loops of certain amino acids (i.e., about 10 of the 20 amino acids) with the hydrophobic capacity of each amino

acid. Essentially there are two grammars, the hydrophilic grammar and the hydrophobic grammar. The folding is driven by the interaction of these grammars through the topological exchanges and the event-driven extraction of water.

The parallelism is easy to explain. There are three types of blind parallelism that we examine: different random seeds, different parameters, and different proteins. Right now we are focusing just on TIM, but in earlier experiments we looked at proteins in the other structural classes to assure ourselves that the program would work for all proteins. We vary the parameters from run to run. We change the program whenever a new rule or a variation on an existing rule is indicated by the results from the computational experiments or just intuition.

3.5 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 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.

The parallel approach used with this code is based on the task farming model. Since each realization is both time consuming and independent of the other realizations, we run a number of sequential jobs in parallel. Our "production" machines have primarily been the BBN TC2000 and a Sun Sparc network. Porting this code to the SP-1 was fairly easy. We compiled, linked, and tested the code on an RS/6000 workstation and ran the *same* binary without change on the SP-1. It is worth noting that no problems were encountered using the Unix system calls `gettimeofday()` and `times()`.

3.6 Superconductivity—Time-Dependent Ginzburg-Landau Equation

Contributed by **David Levine and Gary Leaf**

We have developed a parallel, three-dimensional code to study the formation of vortices in the mixed state of type-II superconductors. The code is based on the time-dependent Ginzburg-Landau (TDGL) equation, which provides a phenomenological description of the macroscopic properties of high-temperature superconductors. Effects of external currents, material defects, and thermal fluctuations can be incorporated into this equation. We are particularly interested in the formation and subsequent evolution of flux vortices and the influence of random impurities on vortex pinning.

Table 5: Results for TDGL on IBM SP-1

No. Processors	Time (sec.) (52 ×52 ×12)	Time (sec.) (150 ×150 ×12)
1	133	1092
2	87	725
4	80	675
8	103	444
16	97	327

We use the single-program multiple-data (SPMD) distributed-memory programming model. The arrays associated with the superconductor are decomposed among the memories of the individual processors. The communication of data between processors is handled using the BlockComm software of Bill Gropp (Section 2.5). Before using the SP-1 we had run this program on several parallel machines, including Sun Sparc and IBM RS/6000 workstation networks and the Intel Gamma and Delta machines.

Porting this code to the SP-1 was fairly easy. We compiled and linked the code with the BlockComm and p4 libraries on an RS/6000 workstation. (We also ran it on several RS/6000 workstations.) We then moved the executable and data files to the SP-1. We were able to run the code, on two different problems, without any changes on 1–16 processors. Solution times for 100 iterations are given in Table 5.

3.7 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 SP-1 with no problems. dac was developed on the Symmetry and a Sun network using p4. Single-node performance was excellent, despite the lack of floating-point operations in dac. To test speedup, we need to do a more carefully controlled set of experiments, in which other users are not on the nodes we are using. The benchmark problem we are using is to prove that a ring where $x^3 = x$ for all x is commutative.

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. Fur-

thermore, 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 SP-1 in the context of significant applications.

We note that Fortran programs must be linked with `-bnso -bI:/lib/syscalls.exp`. This is a serious drawback; it is the one factor that kept people from feeling entirely good about the IBM SP-1. There are a number of places where `xlf` behaves differently from many other Unix Fortrans; these also caused problems. They include failure to provide the C preprocessor to Fortran programs (typically applied automatically to files with extension `.F`) and the choice of the compiler to produce symbols from Fortran that are indistinguishable from symbols generated by C (the lack of a trailing underscore). (We note, however, that we do not have a full software environment up yet; tools such as `loadleveler` and the parallel operating environment could be very helpful. Better documentation would also help; the `loadleveler` "User's Guide" neglects to mention how to start the `loadleveler` GUI and does not mention the command line routines until Chapter 3.

With the delivery of the fast interconnect, we believe that we can immediately begin to use the SP-1 for large applications as well as continued development of portable parallel programming tools. Since all of the applications are built on top of these tools, they should all run with the fast interconnect without any changes other than relinking. Based on early documentation about the EUI-1 programming interface, Chameleon already has an (untested) implementation for the fast interconnect.

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. Technical Report P362-0493, Argonne National Laboratory, 1993.

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

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

[5] 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.

END

DATE
FILMED
9/29/93