ANL/MCS-TM-00

Users Guide for the ANL IBM SP-1 DRAFT

by

William Gropp and Ewing Lusk
Mathematics and Computer Science Division

Steven C. Pieper
Physics Division

ARGONNE NATIONAL LABORATORY • UNIVERSITY OF CHICAGO •

MATHEMATICS AND
COMPUTER SCIENCE
DIVISION

Contents

Chapter 1

Introduction

This guide presents the features of the IBM SP-1 installed in the Mathematics and Computer Science Division at Argonne National Laboratory. This guide describes the available hardware and software, access policies, and hints for using the system productively.

This document is available online as an Emacs "info" document and as a man page (in '`/usr/local/man`'). Many of the tools mentioned in this document also have on-line documentation; in particular, p4, PCN, and Chameleon have info documentation; Chameleon also has an extensive set of man pages.

To read this document on-line without using Emacs, you may use

```
gnu-info -f /home/gropp/papers/sp1/guide.info
```

or

```
/home/lusk/tcl/tkinfo/tkinfo-0.3/tkinfo
```

1.1 Getting Started

This section describes how to get an account and get started on the ANL SP-1.

To get an account, send email to `neumann@mcs.anl.gov` and request an SP-1 account. Fill out the form that is returned to you. You will be notified when your application is approved or denied.

To get started once you have an account, log into `bonnie@mcs.anl.gov`. This is an IBM RS/6000 that serves as a compile server and file system for the IBM SP-1. Assuming that your default shell is `csh` or `tcsh`, do the following:

```
mkdir /u/$LOGNAME
/u/INIT/initcsh
```

This will copy into your '/u/$LOGNAME' directory a default '.login' and '.cshrc' file.

To change your default shell, you will have to send mail to spsupport@mcs.anl.gov. The only shells supported are csh, ksh, sh, and tcsh.

The nodes of the SP-1 are named spnode1 through spnode128. Names of the form spnode001 also work. You may log into them by using rlogin, telnet, or rsh. There are several tools see section 4.5 [Displaying System Information], page 16 that will create xterms for you on any of the nodes.

1.2 Usage Rules

The SP-1 is intended to be used for large computational science projects, particularly those that require massively parallel computing and large and fast disk I/O support. In support of these activities, many groups are developing algorithms and implementations for the SP-1. In both cases, tuning of the programs is an important part of the development and production process. In support of this, usage of the SP-1 is divided into two phases: development during the day and production at night. Further, to simplify daytime development, some nodes (1-32 and 97-128) are reserved for use by the two parallel programming environments (POE/EUI and EUI-H, respectively). You should use nodes spnode33 through spnode96 for any logins. See section 4.6 [Scheduling Use of the SP-1], page 17. In addition, each project should limit itself to one compute process per node.

1.3 SP-1 Etiquette

The following is a short set of guidelines for being a "good citizen." These are reminders about usage rules for the SP-1 that differ from the usual workstation (where there are no rules) or computing facilities (where the rules are enforced).

- Read '/u/schedule/schedule' before starting up jobs to make sure that there is no conflict (see section 4.6 [Scheduling Use of the SP-1], page 17).
- Frequently check space on '/u' and '/u/$LOGNAME' to make sure that other people will have space to work (see section 5 [The File Systems], page 21).
- Run at most one production job on each node.
- Make sure that you haven't left any jobs running, by using pps and pkill (see section 4.3.1 [Summary of Parallel Unix Tools], page 13):

```
pps -all aux | grep $LOGNAME
pkill -all $LOGNAME
```

(You will need '/u/gropp/bin' or '/home/gropp/bin' in your path for these to work.)

1.4 Comments on This Manual

Please send any comments on this manual to gropp@mcs.anl.gov. We are particularly interested in new items for the chapters on known problems and "in Case of Difficulty."

Chapter 2

Machine Configuration

This chapter discusses the hardware and software configuration of the Argonne SP-1, as well as what is not provided on this system.

2.1 Hardware

The Argonne SP-1 consists of 128 nodes and a compile server. Each node is essentially an RS/6000 model 370. This model has a 62.5 MHz clock, a 32-KB data cache, and a 32-KB instruction cache. Key features of this system are:

- 128 MBytes of memory per node
- 1 GByte local disk on each node (400 MBytes available to users, rest for paging)
- Full Unix on each node (IBM AIX 3.2.4)
- Each node accessible by Ethernet from the Internet
- High-performance Omega switch (50 μsec latency, 8.5 MBytes/sec bandwidth when using EUI-H).

In addition, the ANL SP-1 will soon have a large high-performance file system (220 GBytes of RAID disk and a 6-TByte automated tape library).

The peak performance of each node is 125 MFlops (1 64-bit floating-point add and 1 floating-point multiply in each clock cycle). In practice, each node can achieve between 15 and 70 MFlops on Fortran code. Higher performance can be reached by using the BLAS or ESSL routines.

2.2 Software

As each SP-1 node is running a full Unix, most of the usual Unix tools are available. Users may log directly into any SP-1 node using `telnet`, `rlogin`, or `rsh`.

- Multiple parallel programming environments (see section 3 [Programming], page 6).
- IBM's ESSL library
- Performance debugging tools

2.3 What is Not Provided

While the nodes support a full AIX, several services are not supported. These include mail, printing, and access to other file systems (with the exception of '`/u`'; see section 5 [The File Systems], page 21).

We have made no effort to provide PVM or PVM/e on our system.

We also provide no user-support services. We run a "dark" machine room; this means that there is no "operator" to respond to user requests, particularly outside of normal working hours. There are no consultants to help with writing or debugging your programs. We do respond to mail about problems with the machine, and the various research groups often express interest in helping people use their tools and in suggesting new features. See section 11 [Reporting Problems], page 40 for more information.

Please see section 9 [Known Problems], page 28 for a list of known problems. See section 10 [In Case of Difficulty], page 35, for help in identifying problems; please check it before sending mail about a problem with the machine.

Chapter 3

Programming

3.1 Transport Layers

There are a number of *transport layers*, or ways to communicate between nodes, on the SP-1. For most uses, programmers will not use these directly; rather, they will use one of the portable programming libraries. However, as the programming libraries use these transport layers to actually accomplish the communication, it is important to understand them so that the proper transport layer can be chosen.

The available transport layers are Ethernet/IP, Switch/IP, EUI, and EUI-H. Only the first two support multiple parallel jobs on the same node. Both versions of EUI can only run one process per node. In addition, EUI-H is incompatible with EUI and Switch/IP on the same nodes (though the SP-1 can be configured so that EUI-H runs on some nodes and EUI and Switch/IP run on others; this is a common daytime configuration at ANL).

3.1.1 Ethernet/IP

All nodes in the SP-1 are connected by Ethernet, with node names `spnode001` through `spnode128` (leading zeros may be ignored; thus `spnode001` and `spnode1` are the same node). With this choice of transport layer, the SP-1 looks just like a collection of workstations. This method does suffer from the same drawbacks as any Ethernet-connected system: high latency (about 1 msec), low bandwidth (1 MByte/sec), and low scalability (the 1 MByte/sec is shared among all processors).

3.1.2 Switch/IP

It is possible to configure the SP-1 so that the high-performance switch can support IP applications. In essense, any program that runs over Ethernet using the node names `spnode`*xxx* can run over the switch by using the node names `swnode`*xxx*. For example,

```
rcp /tmp/myfile swnode2:/tmp
```

run on any of the nodes copies the file '`/tmp/myfile`' from the local disk on the current node to the local disk on `spnode002` using the switch (thus getting a higher bandwidth). These node names may be used only between nodes on the SP-1; they are not accessible from any other system, including the compile servers. This method suffers from relatively high latency (around 1 msec) but does have good bandwidth and scalability. This mode is available only when EUI is available, and then only on the lower 64 nodes.

3.1.3 EUI

EUI is IBM's message-passing interface to the high-performance switch. There are two versions: one that works with the Parallel Operating Environment (POE) and one that does not. EUI refers to the POE version. POE supports a parallel symbolic debugger (`xpdbx`) and a performance visualization tool (`vt`). However, its performance is inferior to EUI-H (latency is about 405 μsec). In addition, at most 64 nodes are available to EUI. EUI is documented in [7].

3.1.4 EUI-H

EUI-H is an experimental, low-overhead implementation of the EUI interface. It does not support either `xpdbx` or `vt`. It is difficult to provide standard input to EUI-H programs (but see section 3.2.1 [Chameleon], page 7). Also, it is not possible to produce `gprof`-style profiling information from EUI-H programs. All 128 nodes may be accessed using EUI-H when the machine is configured for that (usually at night when a request has been made (see section 4.6 [Scheduling Use of the SP-1], page 17). Features specific to EUI-H are documented in [6].

3.2 Parallel Programming Libraries

We strongly encourage all programmers to use one or more of the systems described in this section when developing parallel programs for the SP-1. These have been used on other systems and have made it easy to port significant applications to the SP-1 very quickly. These systems also offer a spectrum of tradeoffs between functionallity and performance; the Chameleon system in particular has no overhead when used in "production" mode.

3.2.1 Chameleon

Chameleon is a lightweight, portable message-passing system. It provides access to a wide range of transport layers, including EUI, EUI-H, PVM (not supported on the ANL SP-1), and p4. Chameleon provides a common startup model that simplifies choosing a transport layer.

Examples: '`/home/gropp/tools.n/comm/examples`' contains C and Fortran examples as well as makefiles.

Documentation: '`/home/gropp/tools.n/docs/tutorial/parallel.tex`'. The script '`/home/gropp/tools.n/bin/toolman`' will start an `xman` for the manual pages for the Chameleon routines (and others). See also [4].

Supported Transport Layers: EUI, EUI-H, p4

Special Comments: The Chameleon makefiles provide portability by using names defined on the `make` command line. You should set `ARCH=rs6000` and `BOPT=g` (for debugging) or `BOPT=O` (for production) and `COMM=p4`, `COMM=eui`, or `COMM=euih` for the transport layers p4, eui, and euih respectively. If you choose not to use the Chameleon makefiles, be sure that you get all of the required options (e.g., C programs require `-D_POSIX_SOURCE`). Chameleon also provides a way to specify that a file provides standard input to EUI-H programs with the command-line switch `-eui stdin=filename`.

Contact: `gropp@mcs.anl.gov` for more information.

3.2.2 Fortran M

Fortran M 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* which 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.

Examples: '`/usr/local/fm/examples`' contains Fortran M examples as well as makefiles.

Documentation: '`/usr/local/fm/docs/fm_prog_v1.0.ps`' contains a Postscript version of the Fortran M manual. Bound hardcopies (the ANL Technical Report) may be obtained from the contact listed below.

Supported Transport Layers: Ethernet/IP and Switch/IP

Special Comments: See the "Network Specifics" section of the manual for details on running Fortran M on the SP-1. The hostnames of the switch interface (i.e., swnode1) should be used. The latest version of the Fortran M compiler is installed in '`/usr/local/fm`'.

Contact: `fortran-m@mcs.anl.gov` for more information.

3.2.3 MPI

MPI (Message-Passing Interface) is a new message-passing system "standard" that has recently been defined by a broadly based group of parallel computing vendors, library writers (including us), and users. The current draft is now in the public-comment stage [3]. It should be finalized in the spring of 1994. The current draft is available in '`/home/gropp/MPI/mpi-report.tex`' and '`mpi-report.ps`'. A version is available on the World-Wide-Web.

There are two preliminary implementations: one being done by us and a group from Mississippi State University, and the other being done by IBM. Both are in early stages, but do run now on the SP-1.

3.2.3.1 The IBM Version

Not yet available at Argonne.

3.2.3.2 The Public Version

Examples: '`/home/lusk/mpich/examples`' contains examples in Fortran and C, and a makefile.

Documentation: No implementation-specific documentation is yet available. Our implementation uses Chameleon, so the usual Chameleon methods of preparing programs and starting jobs apply.

Supported Transport Layers: EUI, EUI-H, Ethernet/IP, Switch/IP.

Contact: `gropp@mcs.anl.gov` or `lusk@mcs.anl.gov` for more information.

3.2.4 p4

p4 [1, 2] is a portable message-passing system that runs on a very wide variety of parallel systems and workstations. It is in use at approximately 200 sites around the world. Existing p4 programs will run unchanged on the SP-1. The current version is version 1.3, but experimental versions (such as 1.3a) and later releases (1.4) may appear from time to time.

Examples: '`/usr/local/p4-1.3/examples`' on `bonnie` contains C and Fortran examples, example makefiles, and a makefile that can be used to construct your makefiles with the appropriate options for the SP-1's various transport layers.

Documentation: '`/home/lusk/ibm/p4-1.3/docs`' contains the latexinfo source for the manual [2], an ASCII version, and the postscript for a reference card. '`/home/lusk/p4.manual`' contains the postscript for the manual itself in ('`p4.ps`'). The manual is also online via `info` and any of the inside-Emacs or stand-alone `info` readers (e.g., `gnu-info`).

Supported Transport Layers: EUI, EUI-H, Ethernet/IP, Switch/IP.

Special Comments: See the section of the p4 manual entitled "Running on Specific Machines" for how to specify each of the transport layers.

Contact: `lusk@mcs.anl.gov` for more information.

3.2.5 PCN

PCN 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.

Examples: '`/usr/local/pcn/examples`' contains PCN examples as well as makefiles.

Documentation: '`/usr/local/pcn/docs/pcn_prog_v2.0.ps`' contains a Postscript version of the PCN manual. Bound hardcopies (the ANL Technical Report) may be obtained from the contact listed below.

Supported Transport Layers: Ethernet/IP and Switch/IP

Special Comments: See the "Network Specifics" section of the manual for details on running PCN on the SP-1. The hostnames of the switch interface (i.e., swnode1) should be used. The latest version of PCN is installed in '`/usr/local/pcn`'.

Contact: `pcn@mcs.anl.gov` for more information.

Chapter 4

Using the System

This section describes how to run parallel programs and how to monitor the state of the system (IBM provides a powerful system monitoring utility, but it is only available to `root`. The monitoring tools described here were locally developed).

4.1 Compiling and Linking Applications

All compilation and linking should be done on the compile server, `bonnie.mcs.anl.gov` (there will soon be a second compile server called `clyde`). The compile server should not be used to run any other programs.

The compilers are named `xlc` (for C) and `xlf` (for Fortran). (For EUI but not for EUI-H programs you must use `mpcc` and `mpxlf` instead.) These support most of the usual options but sometimes with unusual (for Unix systems) names. Useful options include

-c Compile to a `.o` file

-O3 Produce optimized code

-qsource -qxref Produce a source listing with cross-references

-qlist Produce an object listing. This is equivalent to `-S` on other Unix systems

The following arguments are available only to Fortran programmers:

-qautodbl=dblpad Promote `REAL` declarations to `DOUBLE PRECISION`. This can be useful with Cray codes that require more than 32-bit precision (the default precision of `REAL` on RS/6000's). However, any double precision or `REAL*8` statements will be promoted to `REAL*16` (128 bits) which is significantly slower and probably not what is wanted.

-qdpc=e Treat floating-point constants as double precision.

-P -Wp,-ew Preprocess the Fortran. This can produce faster code (these are the options used to get the published LINPACK benchmark numbers). However, they can also slow down code and, in combination with `-qautodbl`, have been observed to produce incorrect results (on the NAS uniprocessor benchmarks). There are other, less aggressive, preprocessing options.

WE ARE CURRENTLY DISCOURAGING THE USE OF THE PREPROCESSOR.

A number of correct Fortran codes have been shown be preprocessed into *incorrect code* by the preprocessor. See section 10 [In Case of Difficulty], page 35 for more details.

-qextname Suffix an underscore to all external names. This makes Fortran routines available to C programs that expect Fortran to add an underscore suffix; this is common (but not standard) behavior of many Unix Fortran compilers. Note that some software packages do not expect Fortran users to use this option; their Fortran interfaces adjust to the default behavior of the Fortran compiler. Using `-qextname` with these libraries can lead to unresolved references.

There is no preprocessor for C programs. Unlike many other C compilers for Unix systems, the RS/6000 C compiler does not appear to do any loop unrolling.

When linking, this option can be useful:

-qloadmap:filename Produce a load map. This is handy for identifying from which library an object file was loaded, or for seeing the total memory requirements of your module.

-o name Provide a name for the module. The usual Unix default of `a.out` is used if no name is specified.

4.2 Running Applications

Each parallel programming system has its own requirements for running an application. In this section, we describe some issues that are common to all systems.

To execute a program, choose a node that you will use as the master node; you can use `xsp1load` to identify lightly loaded nodes. You may also want to identify the specific nodes to run on or leave that choice to the parallel programming package. Login to the chosen master node by using `telnet` or `rlogin`, or pressing the left mouse button over the node in an `xsp1info` or `xsp1load` display. You can then run any uniprocessor program in the same way you would on any Unix system.

If you are running a parallel job, make sure that when it exits, all of the cooperating processes actually exit. This is particularly important if you kill the job with a `control-C`. Because the SP-1 is a relatively stable machine, jobs can persist for days or even weeks. See section 4.3 [Parallel Unix Tools], page 13, which describes tools that can be used to check for and kill these processes.

You can run jobs in the background; if you are using `ksh` as your shell, be sure to start the job with `nohup`:

```
nohup ./myprog.x ... < input.file  > output.file &
```

4.3 Parallel Unix Tools

We have developed parallel analogs of the common Unix tools `cp`, `ls`, `ps`, and others. These are scalable and relatively fast. They simplify the task of executing commands on all or part of the SP-1. This section discusses only a few of the available tools. A paper that describes the tools and their design and implementation is in preparation; a draft is available in '`/home/gropp/papers/ptools/paper.tex`' and in '`/home/gropp/papers/ptools/paper.ps`'. You will need '`/u/gropp/bin`' or '`/home/gropp/bin`' in your path for these to work.

4.3.1 Summary of Parallel Unix Tools

This section gives a brief description and example of some of the available tools. These tools are shell scripts and can be found in '`/var/bin`' and '`/u/gropp/bin`' on the SP-1 nodes and in '`/home/gropp/bin/sp1`' in the MCS file systems. Since they are shell scripts, they may be used from all of the workstations; it is not necessary to log into a spnode to use them.

The first arguments specify the nodes to run the command on. This is given as a range of nodes (two numbers separated by blanks) or one of the following shortcuts:

-all All 128 nodes

-low Nodes 1 through 64

-high Nodes 65 through 128

-frame n The 16 nodes in frame n, with frames numbered from one to eight

These shell scripts produce a number of messages about starting and stopping background processes; these may be eliminated by piping the output through `egrep`:

```
pls -all ... | egrep -v "^ \["
```

For each command that generates output (such as `pls`), each line contains the name of the node that generated the output at the beginning of the line. By piping the output of these commands through `sort`, the output can be listed in spnode order.

```
pls -all ... | egrep -v "^ \[" | sort
```

pls list files on a range of nodes

```
pls 1 5 /tmp
```

lists the contents of /tmp on spnode001 through spnode005.

pps list the processes on a range of nodes

```
pps 6 11 aux
```

is similar to `ps aux` on spnode006 through spnode011.

pcp Copies a file to a range of nodes

```
pcp 12 16 a.out /tmp/$LOGNAME/a.out
```

copies '`a.out`' to '`/tmp/$LOGNAME/a.out`' on spnode012 through spnode016. Note that you must have created '`/tmp/$LOGNAME`' first (see prun).

pdisp Display the output from these commands in an X Window. The input is taken from standard input; `pdisp` is usually used as the right end of a pipe:

```
pls -all /tmp/foobar | grep -v found | pdisp
```

prm Removes a file from a range of nodes

```
prm 12 16 /tmp/$LOGNAME/a.out
```

removes the '`/tmp/$LOGNAME/a.out`' on spnode012 through spnode016.

ppred Run a "predicate" on a range of nodes:

```
ppred 17 21 '! -s /tmp/a.out' 'echo $nodename'
```

writes the names of the nodes (between spnode017 and spnode021) that have a file '`/tmp/a.out`'.

pkill Kill jobs with a given name on a range of nodes

```
pkill 21 25 infiniteloop
```

kills (-9) `infiniteloop` on spnode022 through spnode026.

prun Runs a command on a range of nodes:

```
prun 12 16 "mkdir /tmp/$LOGNAME"
```

makes the directory '`/tmp/$LOGNAME`' on nodes spnode012 through spnode016.

In addition to these tools, there are two more that are not yet integrated with these. These are located in '`/u/spieper/bin`'. They are

pst Shows jobs using the nodes. This script uses `prun` to do a filtered `ps` command on the specified nodes:

```
/u/spieper/bin/pst 1 12
```

shows user tasks that have used more than 1 CPU second on spnode001 through spnode012.

spstat Show status (essentially the same report as `pst`) or can be used to run a command on a bunch of nodes. This command does not use the parallel tree structure of the above commands and hence is much slower if a large number of nodes are to be addressed. However, it is significantly more robust against failure if some nodes are broken.

```
/u/spieper/bin/spstat 1-12
/u/spieper/bin/spstat "1-12 22 33-37"
```

produces a status report for the specified nodes.

```
/u/spieper/bin/spstat -cmd "kill -9 -1"  "1-12 22 33-37"
```

will kill (-9) ALL tasks belonging to you on the specified nodes.

4.3.2 Examples of Parallel Unix Tools

In running a parallel program, it is important to distribute it to the local disks. The following code may be used:

```
prun -all "mkdir /tmp/$LOGNAME"
pcp  -all /u/$LOGNAME/myprog /tmp/$LOGNAME/myprog
```

(Of course, the first command to make the directories need only be used once.)

The following code will display on which nodes a file is present; this may be used to check that a file has been properly distributed:

```
pls -all /tmp/$LOGNAME/myprog | grep -v 'found' | pdisp
```

This requires `wish` for `tk-3.3` (see section 4.5 [Displaying System Information], page 16).

4.4 Linking and Distributing Parallel Programs

When building your program, you should statically link it. Add the options

```
-bnso -bI:/lib/syscalls.exp
```

to your link line. This *may* help reduce time-out problems that we have been seeing with jobs that use large numbers of processors. The Chameleon and p4 makefiles do this for you.

Once the program is built, it should normally be distributed to the local disks and run from there. This avoids having a large number of nodes make network requests for the file to be executed.

First, make sure that you have built a directory in '`/tmp`' for your executables. This can be done with

```
prun -all "mkdir /tmp/$LOGNAME"
```

Then, copy the executable to each node. Assuming that the program is called `myprog` and is in your '`/u/$LOGNAME`' area, use

```
pcp -all /u/$LOGNAME/myprog /tmp/$LOGNAME/myprog
```

These assume that `prun` and `pcp` are in your `PATH`. If you have already distributed a version of `myprog`, you should consider doing

```
prm -all /tmp/$LOGNAME/myprog
```

before copying the file (this should not be necessary but it sometimes seems to be required).

4.5 Displaying System Information

Currently, there are four X Windows tools for displaying the state of the system. These are `wish` scripts that require `tk-3.3`. This code is available in the MCS Division. As `tk` is in the public domain, other sites should have no problem acquiring it. In addition, there is a program provided by IBM that gives some information on POE.

The five tools are

xsp1info Shows the EUI-H usage of the machine. Each partition is displayed in a different color (up to six colors), and the owner and amount of time that each partition has been running are displayed.

xsp1load Shows the load (short-term value from `rup` on each processor. The load is displayed by using cool to warm colors.

xsp1df Shows the amount of '`/tmp`' in use on each node as a pie chart. Clicking on a node with the right mouse button pops up a pie chart showing the usage of that node's '`/tmp`' disk.

xsponu Shows the usage of the '`/u`' disk as a pie chart. The left mouse button brings up a window with the directory of '`/u`' and the amount of space; the middle mouse button labels the directory on the pie chart.

vt The "performance monitor" entry under "file" in the `vt` window will give a display of the nodes that are available to POE/EUI and the nodes that are in use. It is currently impossible to determine what users is using POE/EUI.

Currently, all except `vt` reside in '`/home/gropp/bin`'. You must have read-access to the MCS file-systems to use them. These may not be run from the SP-1; rather they should be run from a Sun or RS/6000 workstation. Currently, they require a color display to work properly.

Each of these has a large and small version. The left mouse button in every case will start an xterm on the indicated node. The meaning of the other mouse buttons depends on the tool; each tool indicates the mouse-button bindings along the bottom of the tool's display.

Sample output from these tools are shown in Figures 4.1, 4.2, and 4.3.

4.6 Scheduling Use of the SP-1

From 8:00 am to 5:00 pm Central Time, the machine is divided into four parts (note the overlap):
Nodes 1-32 reserved for EUI jobs
Nodes 97-128 reserved for EUI-H jobs
Nodes 33-64 available for EUI, Switch/IP, and other jobs
Nodes 65-96 available for EUI-H and other jobs

During the day, program development has priority for the EUI-H part of the machine. You should restrict yourself to *one* partition and, if using 32 or more nodes, try to not let it run for more than 10 minutes.

The schedule of stand-alone users for evening and night runs is kept in the file '`/u/schedule/schedule`'. To be added to the schedule, send mail to `gropp@mcs.anl.gov` and to `lusk@mcs.anl.gov` giving

- Number of nodes
- Kind of transport (EUI-H, any, none, EUI)
- Time (e.g., from 6:00 pm to 10:00 pm)
- Nodes, if less than 128 nodes requested

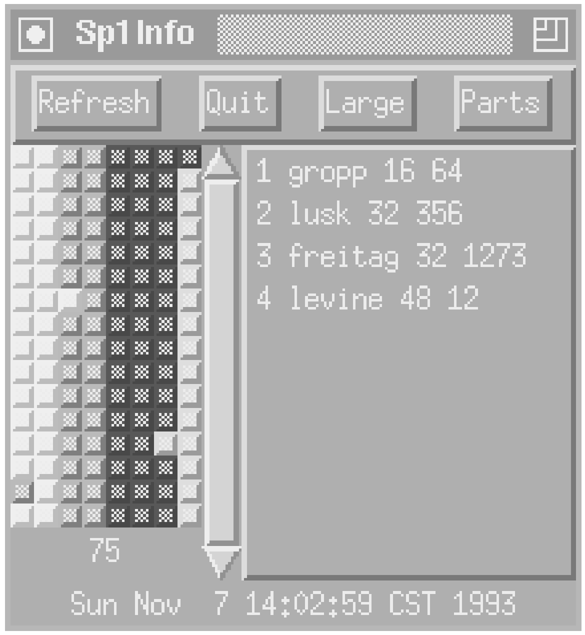

Figure 4.1: Sample `xsp1info -small` display

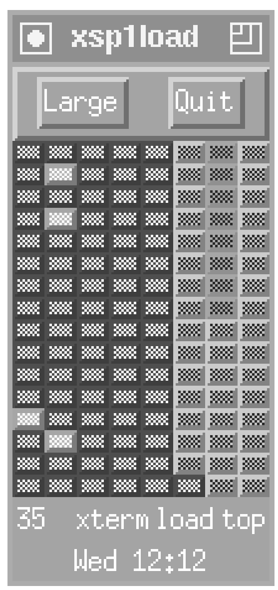

Figure 4.2: Sample `xsp1load -small` display

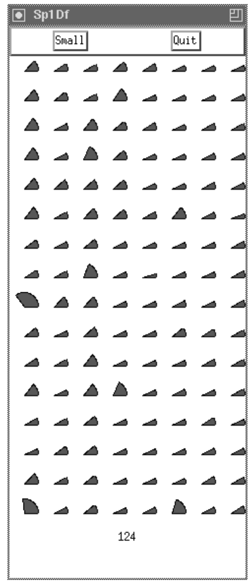

Figure 4.3: Sample `xsp1df` display

- Jobs you wish to continue running into the next day, if any (only available for jobs that do not use the switch).

4.7 Additional Information

There are three IBM manuals on the Fortran (and C) compilers:

The Language Reference In addition to describing the Fortran language it explains a large number of IBM-provided system-interface type of subroutines.

The User's Guide describes compiling, linking, use of the preprocessors, interaction of Fortran with files, etc.

The Optimization and Tuning Guide for the XL Fortran and XL C Compilers a very useful (and readable) description of optimization considerations for the RS/6000 architecture.

If you are using an X-windows system to access the compile server, you may view these manuals on line using IBM's `info` system. Before invoking `info`, define your display to the compile server:

```
setenv DISPLAY myid.mcs.anl.gov:0.0    (csh)
```

```
export DISPLAY=myid.mcs.anl.gov:0.0    (ksh)
info
```

where `myid.mcs.anl.gov` is the TCP/IP address of your computer or X-station. When `info` finally comes up (it can take several minutes), select the "List of Books" panel, find the desired manual, and double-click on it.

When connecting to Sun workstations, `info` (and most IBM X Windows tools) will generate several warning messages. These may be ignored.

Chapter 5

The File Systems

There are currently two file systems accessible to the nodes: '`/u`' (shared) and '`/tmp`' (local). The third file system (the 220 GByte RAID arrays) is not yet available (as of 1/4/94) and will not be discussed in this document.

The file system '`/u`' is shared between the nodes and the compile servers. This file system is NFS mounted.

Warning:

There is a problem where files that are copied over an existing file do not always seem to be noticed. You should always `rm` a file before you copy over it.

When you first get an SP-1 account, you may need to create a directory in '`/u`' for your use:

```
mkdir /u/$LOGNAME
```

Into this directory you should copy your environment (the files '`.login`', '`.cshrc`', etc.). Make sure that they work by trying to log into an spnode. Typical problems include a reliance on files in '`/mcs`' and Sun-specific commands.

Be frugal with your use of '`/u`'. There is only 1 GByte of space in '`/u`', and this must be shared by all users, including those with large input and output files. Try not to keep any large files in '`/u`' except when you need them for a run (don't move files back and forth during the day but also don't leave a large output file on '`/u`' once it is generated). The program **`xsponu`** may be used to display the usage of '`/u`'.

Each local disk holds 400 KBytes and may also be used for temporary data files and for executables, but again, you should not leave large files on them. The program **`xsp1df`** (see section 4.3 [Parallel Unix Tools], page 13) can be used to display the amount of space available on each node's '`/tmp`' partition. (The rest of the local disk holds the local '`/var`' partition and the swap area.)

The SP-1 nodes do *not* mount the MCS file systems. Thus, the only way to move files is either by first moving them to '`/u`' or by using `rcp`.

For additional information on other file systems being considered or developed for the ANL SP-1, see '`/home/nickless/Documents/io.ps`'. Contact `nickless@mcs.anl.gov` for more information.

Important Note:

Neither '`/u`' nor '`/tmp`' are backed up. In addition, the '`/tmp`' areas will be cleaned of old files on a regular basis; large files may be removed at any time.

Chapter 6

Correctness and Performance Debugging

Several tools may be used to do correctness and performance debugging of parallel programs. This section describes these tools and how to use them.

6.1 pdbx and xpdbx

IBM's POE provides parallel versions of `dbx` and `xdbx` called `pdbx` and `xpdbx` respectively. These work with POE/EUI programs, and allow you to run and debug parallel applications. To debug an EUI program `myprog` on, for example, four processors, use

```
setenv MP_PROCS 4
xpdbx myprog
```

6.2 dbx

It is also possible to use `dbx` and `xpdbx` with parallel programs by loging into each node and using the command

`dbx` *process_id*

after the parallel programs have started. You can get the *process_id* with

```
ps -ef | grep $LOGNAME
```

For EUI-H jobs, use

```
ps -ef | grep xeui
```

Note that using `dbx` to debug EUI-H jobs is difficult; you can, however, get a traceback (showing in what routines the program is executing) by using the `where` command in `dbx`.

6.3 Chameleon Options

Chameleon provides a number of features that support both correctness and performance debugging. These apply only to programs compiled with `BOPT=g` or `BOPT=Opg`. These featuers may be selected by command-line switch when the Chameleon application is started:

-event Generate an event log that may be viewed with upshot (see section 6.4 [upshot], page 24).

-summary Generate a summary of message passing activity, giving the amount of time spent in each process sending and receiving messages

-trace Produce a line of output for each message-passing operation (including the begining and ending of receives). This is useful for finding the cause of deadlock in parallel applications.

6.4 upshot

Upshot [5] is a portable X Windows program for visualizing the behavior of a parallel program. Both Chameleon and p4 can generate the event logs that `upshot` reads. To generate an event log from Chameleon, use the `-event` flag

```
a.out -np 4 -event
```

This generates a file '`bl`'. To view this file with `upshot`, use

```
upshot -l bl
```

In order for a Chameleon program to be able to generate this event file, the `BOPT=g` or `BOPT=Opg` version must be used.

Currently, you should have '`/home/gropp/bin/sun4`' in your path to use `upshot` on the Sun workstations. The version in '`/usr/local/bin`' will be updated in the near future.

Sample output from `upshot` is shown in Figure 6.1.

Chameleon can generate event logs with any transport layer, including both EUI and EUI-H.

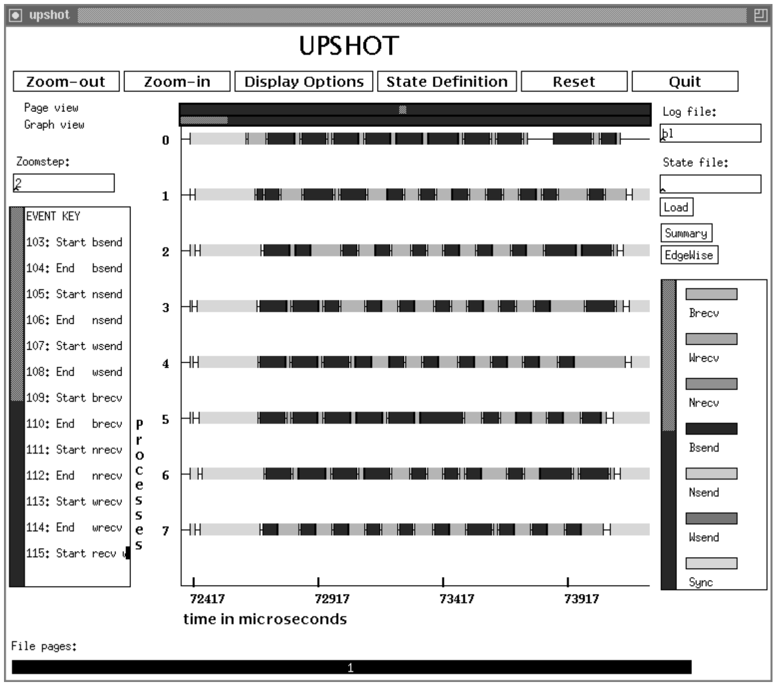

Figure 6.1: Sample upshot output

6.5 vt

The POE (Parallel Operating Environment) provided by IBM includes a performance visualization tool called vt. This tool, like upshot, can analyze a tracefile that is generated by a parallel program. Currently, the only way to generate the tracefiles that vt reads is with EUI (not EUI-H). To generate the trace file, run your EUI application with the option -tlevel 9:

```
a.out -procs 4 -tlevel 9
```

To run vt (which is an X Windows tool), make sure that your DISPLAY variable is correctly set and that your server allows connections from the node on which you are running (see the xhost command), and do

```
vt -tfile a.out.trc
```

You may run vt from bonnie or any of the spnodes (but use nodes 33–96 only during the day).

Chapter 7

Nonportable Programming

It is possible to use EUI and EUI-H directly without using the portability tools. This section describes how to run these programs.

7.1 Running EUI-H

First, when building your program, you should statically link it. Add the options

```
-bnso -bI:/lib/syscalls.exp
```

to your link line. This *may* help reduce time-out problems that we have been seeing with jobs that use large numbers of processors.

Second, distribute the executable to the local disks. See section 4 [Using the System], page 11, for more information.

You should use the script '`/usr/lpp/euih/eui/cotb0`' (the EUI-H documentation describes `tb0eui`; this script is obsolete). A typical example is

```
/usr/lpp/euih/eui/cotb0 -b /tmp/$LOGNAME/a.out 10 ...
/usr/lpp/euih/eui/cotb0 /tmp/$LOGNAME/a.out 10 ...
```

where the first runs `a.out` as a batch program and the second opens an `xterm` window for each processor.

7.2 Running EUI

First, when building your program, you should statically link it. Add the options

```
-bnso -bI:/lib/syscalls.exp
```

to your link line. This *may* help reduce time-out problems that we have been seeing with jobs that use large numbers of processors.

Second, distribute the executable to the local disks. See section 4 [Using the System], page 11, for more information.

Chapter 8

Benchmarking

Benchmarking on the SP-1 is more difficult than on most other MPPs because the SP-1 provides a friendly multiprocessing environment on each node. Avoiding the effects of contention with other users requires that the machine be scheduled as single user. To schedule single-user time, see section 4.6 [Scheduling Use of the SP-1], page 17.

It is important to distinguish between elapsed (also called wall-clock) and CPU time when running a parallel program. Depending on the transport layer that is used, CPU time may not include the time in which processes were idle while waiting for messages to arrive. It is best when benchmarking to compute both the elapsed and CPU time; for most purposes, elapsed time is the better indicator of true performance.

The Chameleon package (see section 3.2.1 [Chameleon], page 7) provides portable routines for measuring both elapsed and CPU time.

```
double start_cpu, cpu_time, start_elapsed, elapsed_time;
...
start_cpu     = SYGetCPUTime();
start_elapsed = SYGetElapsedTime();
... work to time ...
elapsed_time  = SYGetElapsedTime() - start_elapsed;
cpu_time      = SYGetCPUTime() - start_cpu;
```

Fortran versions of these are also available.

Chapter 9

Known Problems

1. NFS file updates. We are seeing a problem where changing a file on '`/u`' does not seem to actually change the file. Until this is fixed, the best work-around is to always `rm` the file rather than `cp`'ing over it. If you change a program, rebuild it, and then run it and don't see the effect of your change, you are probably seeing this problem.

2. '`/tmp`' fills up on `spnode1`. Beause of a bug in the "resource manager" for POE/EUI, the file '`/tmp/resd_err`' can consume all available space on the '`/tmp`' area of `spnode1`. Removing this file does not fix the problem, because the file is still open (the resource manager is still running and has a link to the file). If you remove the file manually, it becomes very difficult to recover the space. Instead, use the following:

   ```
   cat /dev/null > /tmp/resd_err
   ```

3. Linking takes *forever*. This is a feature. However, there are some things that you can do to (sometimes) speed things.

   (a) If you are relinking an application, use your old executable as *input* to the linker, along with *only* the modules that you have changed. For example, if the executable is *a.out* and the file '`changed.f`' contains the *only* changes, then

   ```
   xlf changed.f a.out
   ```

   will generate a new '`a.out`' executable. You will also get an error message about "xlf: 1501-218 file a.out contains an incorrect file suffix": you can ignore this message. If you use an executable name with no suffix, you will not get this message.

   This also works with `xlc` (including the error message).

   (b) If you are using one or more object libraries, consider moving them to a local disk. On `bonnie`, this means to '`/u`'.

(c) Consider prebinding any libraries. For example,

```
ld -r foo.a bar.a -o foobar.o
```

will produce a file, '`foobar.o`', that is "prebound"; that is, all of the references between functions and symbols within these files are resolved. This can speed up the time to link, particularly if you make sure that this file is local (on `bonnie`, on '`/u`').

(d) **(Experts only):** Consider building a *shared* library. Information on this may be found by using `info` and looking for "How to Create a Shared Library."

(e) Check that only four `biod` processes are running on the compile server. In experiments, using 16 `biod` processes caused a significant increase in linking time compared to 4 `biod` processes.

4. `info` generates "X Error of failed request: BadValue" when it starts. This is a feature. You can ignore it (it has to do with the fonts used by the X11 server).

5. Fortran code sometimes fails when optimized. There are a number of potential problems here.

   (a) If the preprocessor is selected (`-O3 -P -Wp,-ew`) and the `-C` option is selected to enable subscript checking, then the Fortran compiler will generate incorrect code for the subscript checks. This is caused by the preprocessor inserting a temporary array with dimension one and equivalencing this to a large common block (also inserted by the preprocessor). Apparently, the `-C` option does not recognize this case, and generates an error when the temporary array is used. The fix is to not compile with `-C`.

9.1 IBM Release Notes

This section contains the known problems from the IBM Release notes for POE. These affect *only* the use of POE (EUI and program development tools `vt`, `pdbx`, etc.). Note that there is currently no similar list for EUI-H.

9.1.1 Use of AIX System Calls

1. If the fork() call is used to create additional tasks, only the parent may issue message-passing library (MPL) calls. The child is unknown to the parallel environment.

2. The `usrinfo()` function does not return valid user information.

3. The Parallel Operating Environment uses many of the system libraries, particularly the socket libraries. Application writers should ensure that their function and subroutine names do not duplicate AIX system function names.

9.1.2 Use of Standard I/O in the User Application

1. STDOUT may be closed by the application. However, communication with the Program Marker Array will be lost.

2. STDERR should not be closed by the application. Communication with the Partition Manager will be lost. Specifically, `MP_STOPALL` will not be able to stop the job.

3. STDIN does not recognize the normal end of file character (EOF = -1). End of file on STDIN is signaled by the character X'FF' (255).

9.1.3 Message-Passing Subroutine Libraries

1. If `MP_STOPALL` is called, VT trace files may be incomplete and are not integrated on the initiating node.

2. If `MP_STOPALL` is called, parallel profiling files are not written.

3. The aggregate length for messages initiated at any one time is limited to 40 MB. There are also individual message length restrictions.

9.1.4 Exit Status of User Application

The exit status of the parallel program is determined as follows:

1. If `MP_STOPALL(nn > 0)` is called, the exit status is nn.

2. If all tasks terminate via exit(mm > 0) or STOP(mm), the exit status is the largest value of mm returned.

3. If any task is terminated via a signal (e.g. segment violation), the exit status is 128+(signal).

4. The exit status of a parallel program terminated by Control-C (Interrupt) is undefined.

9.1.5 Operation and Use Notes

The following information amends AIX Parallel Environment Operation and Use (SH26-7230-0).

9.1.5.1 Visualization Tool

1. Under Motif 1.2 in AIX 3.2.4, some of the Visualization Tool popup panels may be garbled or display distorted geometries. This does not affect the overall behavior. The solution is to use the Motif library (`libXm.a`) from AIX 3.2.3 + Extensions. This can be done by setting the environment variable LIBPATH, to `<motif_path>:/usr/lib:/lib` where `<motif_path>` is the path to the `libXm.a` file from AIX3.2.3 + Extensions.

2. When tracing AIX statistics, the Visualization Tool statistics daemon version (TCP/IP or High Performance Switch) must match the application mode in order to get valid traces. Running a parallel application using TCP/IP message passing while the High Performance Switch version of dig is running will not affect application behavior but may generate invalid trace files that cannot be processed by the Visualization Tool.

3. Trace records are not generated for CCL subroutine calls and are not recognized by VT.

4. A message about an unknown resource, hScale, may be generated when certain Bar Chart displays are used. This message can be ignored.

5. Moving the time slider end range marker to a point *before* the latest point in the trace file read may cause the Visualization tool to terminate abnormally.

6. When using the VT source code widget, the source and executable must be in the directory from which VT was started.

7. If a message-passing call encounters an error that terminates the program, the VT trace files may be incomplete and are not integrated on the initiating node.

8. The `vtsample` executable supplied for the Visualization Tool should be recompiled and executed. This produces a trace file to be used with the Operation and Use Guide. Refer to the `makefile` and `README` files in the `/usr/lpp/vt/tracefiles` directory.

9.1.5.2 Partition Manager

1. Partition Manager automatically terminates jobs when a switch fault occurs.

2. The `mpload` program does not support Visualization Tool trace collection nor the Program Marker Array.

3. The userid (number) and user name must be the same on all parallel nodes and the control node.

4. Standard output streams not terminated with a new-line character are buffered even if a subsequent read to standard input is made. This may cause prompt messages to appear only after the input has been read.

9.1.5.3 Parallel Profiler

1. The Parallel Profiler does not collect profiling statistics for the message passing library.

9.1.5.4 Parallel Debugger

1. pdbx and xpdbx do not recognize command-line flags for the parallel environment. Set these via the `MP_` environment variables.

2. xpdbx does not display many c data structures correctly. Use the command window to view these structures.

3. pdbx and xpdbx do not run properly with csh. Use ksh if possible. Otherwise, make sure that your `DISPLAY` and `MP_PROCS` are set correctly. Also, you may need to run from a subdirectory (not your login directory).

4. pdbx and xpdbx do not support STDIN redirection from a file.

9.1.5.5 On-line Documentation Notes

Consult the file '`/usr/lpp/pedocs/README`' for information about initializing InfoExplorer for the Parallel Environment. `info -l pe` provides access to the pertinent Parallel Environment book articles. `man` pages are also described in file '`/usr/lpp/pedocs/README`'.

9.2 IBM Documentation

This note lists various SP-related documents. Those documents followed by "[== some date, 1993]" are orderable, bound, IBM documents. There is at least one copy of these documents in the lab and Tim Lehmann has unbound equivalent copies (with the noted date). All documents not followed by "[== some date, 1993]" are available unbound (from Tim Lehmann; some are not orderable and some came with the software). There's a day or two turn around to have copies made for the big documents.

The following EUIH document exists for Release 1.0
- EUIH: An Experimental EUI Implementation, 9/27/93 (preliminary), Version 1.06.3

The following LoadLeveler documents exist for LL Release 1.1
- IBM LoadLeveler Quick Reference Summary, Release 1.1, SX23-1048-01, no date [== 1993]

- IBM LoadLeveler User's Guide, Release 1.1, SH26-7226-01, September, 1993 [== August 17, 1993]
- IBM LoadLeveler Administration and Installation Planning Guide, Release 1.1 SH26-7220-01, September, 1993 [== August 17, 1993]
- IBM LoadLeveler Release Notes, AIX V3, Release 01, Mod level 01, no date

The following Parallel Environment documents exist for Release 1.0

- IBM AIX Parallel Environment Parallel Programming Reference, Release 1.0, SH26-7228-0, September, 1993 [== August 17, 1993]
- IBM AIX Parallel Environment Operation and Use, Release 1.0, SH26-7230-0, September, 1993 [== September 8, 1993]
- IBM AIX Parallel Environment Installation and Diagnosis, Release 1.0, SH26-7231-0, September, 1993 [== September, 1993]
- IBM AIX Parallel Environment, AIX 3.2.4, Release Notes, Release 01, Mod level 00, no date

The following Parallel Environment document no longer exists for Release 1.0

- IBM AIX Parallel Environment Application Development, Release 1.0, SH26-7223-00, July 23, 1993 (and earlier). [The release notes refer the user to SH26-7228-0 for this information.]

The following SP documents exist for Release 1.0

- IBM 9076 Scalable POWERparallel Systems: Administration Guide, SH26-7221-00, September, 1993 [== _, 1993]
- IBM 9076 Scalable POWERparallel Systems: Maintenance Information, SY66-0299-00, September, 1993 [== _, 1993]
- IBM 9076 Scalable POWERparallel Systems: Planning and Installation Guide, SA23-2481-0, September, 1993 [== _, 1993]
- IBM 9076 System Support Programs, AIX 3.2.4, Release Notes, Release 01, Mod level 00, no date

The following PFS document exists

- Parallel File System External User Interface, Proposal, Version 1.0, September 17, 1993

The following PVMe document exists

- IBM AIX PVMe User's Guide and Subroutine Reference, Release 1.0, SH23-0019-00, August 18, 1993

The following Vesta documents exist

- Overview of the Vesta Parallel File System, no date
- Satisfying the I/O Requirements of Massively Parallel Supercomputers, no date

- Interfacing Vesta to an External File System, Version 0.3, August 23, 1993
- Vesta File System Programmer's Reference, Version 0.82, August 10, 1993

Chapter 10

In Case of Difficulty

1. Q: You run your program and it seems to be running the "wrong" program (e.g., print statements that you added do not seem to work).

   A: You may be running afoul of the NFS bug (see section 9 [Known Problems], page 28). Remove the executable and object files and rebuild the program.

2. Q: You run your POE/EUI program with `your-program-name -procs 4` and get this output:

   ```
   RM_CLIENT : The RM server is running on spnode001.mcs.anl.gov
   RM_CLIENT : Error connecting stream socket
   ERROR: PEPM117: Unable to contact Resource Manager
   ```

   A: This means that POE/Lightspeed is unavailable. Check the schedule, and if POE is supposed to be available, send mail to `spsupport`.

3. Q: I tried to use IP over the switch (node names `swnode1` etc) and I got an error message. Can the SP-1 be configured so that I can use IP over the switch?

   A: The real problem is this:

   - EUI-H is an order of magnitude faster than POE/Lightspeed/EUI
   - Only POE/Lightspeed/EUI supports IP over the switch
   - While EUI-H can be loaded and unloaded relatively easily, loading POE/Lightspeed/EUI requires rebooting the machine.

   Thus, when the switch is up, we tend to run just EUI-H. Our portability tools hide this from the users, so they get more portability than PVM as well as significantly better performance.

4. Q: When running your EUI-H program, you get messages of the form

   ```
   ...Checksum Error 0x00000AA
   ```

   (the specific number is unimportant).

   A: This means that the switch has reset while messages were in transit to your application. Your application is now dead. Please send the messages and the time when they occured to `spsupport`. This can help us track down hardware problems in the system.

   This can also happen if the switch is reconfigured while your application is running. It may *not* be a hardware error.

5. Q: When running an EUI-H program, you get messages of the form

   ```
   *** blah blah
   ```

   or

   ```
   *** Network is up
   ```

   A: This message comes from the high-performance switch. It usually means that your appication is dead. Send the message and the time when they occurred to `spsupport`.

6. Q: When running your EUI-H program, the application never seems to start.

   A: This usually means that the EUI-H startup code has timed-out. There is nothing to do except try to restart your application. Make sure that your application is statically linked and that you have distributed it to the local disks ('`/tmp`') and that you are starting the version on the local disks. Trying to run a parallel application whose executable must be fetched over the Ethernet from `/u` is a common source of time-outs.

7. Q: My program is in '`/u/$LOGNAME/myname`', but when I try to execute it, I get

   ```
   myprog: Command not found.
   ```

   A: This means that your `PATH` does not contain the current directory (usually a single "`.`" in the PATH. Either modify your `PATH` or prefix the program name with `./`:

   ```
   ./myprog ...
   ```

   Also see section 1.1 [Getting Started], page 1.

8. Q: My program runs correctly when I compile it `-g` but not when I compile it with optimizations (`-O`).

   A: The compiler may be generating incorrect code. Try using bisection on your source files: compile half with `-g` and half with `-O` and test again. Continue replacing files compiled with `-O` with ones compiled with `-g` until the code works. You may be able to simply use a few modules compiled with `-g` in your application.

   If you can produce a simple example (a few dozen lines of code) that you are sure is incorrectly compiled, send it to `spsupport`.

9. Q: My program runs correctly when I compile it without the preprocessor but incorrectly when I use the preprocessor (`-P`).

   A: The following code is known to cause the preprocessor to generate incorrect code:

```
      subroutine foo( nr, nz )
      integer nr, nz
      common /bar/aw(1025,1025), dn(1025,1025)
      integer i, j, k, mij, mk, nrm1

      j = 1
      nrm1 = nr - 1
      do 10 i=2,nrm1
          mij = i + (j-1)*nr
          do 10 k=1, nz
              mk = i + (k-1)*nr
              aw(mij,mk) = dn(j,k)
10    continue
      j = nz
      do 20 i=2,nrm1
          mij = i + (j-1)*nr
          do 20 k=1, nz
              mk = i + (k-1)*nr
              aw(mij,mk) = dn(j,k)
20    continue
return
end
```

   Looking at the output of the preprocessor (use `-P -Wp,-ofoo.f,-lfoo.lst`), it is clear that the first double-loop (to statement `10`) is processed correctly but that the second double-loop does not properly compute the second dimension of `aw`. Here are the generated code fragments. For loop `10`, the body is converted into

```
do i = j1 + 1, nrm1 - 1, 4
   k2 = 0
   do k = 1, nz
      r1 = dn(1,k)
      aw(1+1026*i,1+k2) = r1
      aw(1027+1026*i,1+k2) = r1
      aw(2053+1026*i,1+k2) = r1
      aw(3079+1026*i,1+k2) = r1
      k2 = k2 + nr
   end do
end do
```

For loop 20, the body is converted into

```
do i = j4 + 1, nrm1 - 1, 4
   do k = 1, nz
      r2 = dn(j,k)
      aw(1+(j-1)*nr+1026*i,k) = r2
      aw(1027+(j-1)*nr+1026*i,k) = r2
      aw(2053+(j-1)*nr+1026*i,k) = r2
      aw(3079+(j-1)*nr+1026*i,k) = r2
   end do
end do
```

For some reason, in the second loop, the preprocessor fails to use `(k-1)*nr` for the second argument.

Because of this, use of the preprocessor is not recommended. Unfortunately, this will often reduce performance.

10. Q: I tried to use `xsp1load`, `xsp1info`, or `xsp1df` but I got the error message

    ```
    bonnie % Xlib:  connection to "neptoon:0.0" refused by server
    Xlib:  Client is not authorized to connect to Server
    couldn't connect to display "neptoon:0"
    ```

    A: You need to do `xhost + bonnie` on the X11 server you are using (in this example, `neptoon`). Some other tools will require that you have allowed connections from the individual nodes; you may want to add `xhost + spnode$i` for `i` from 1 to 128.

11. Q: My parallel program runs on other parallel machines but seems to deadlock on the SP-1 when using EUI, EUI-H, or Chameleon.

    A: The following parallel program can deadlock on *any* system when the size of the message being sent is large enough:

```
send( to=partner, data, len, tag )
recv( from=partner, data, maxlen, tag )
```

where these are blocking send's and receives (`mp_bsend` in EUI/EUI-H and `PIbsend` in Chameleon). For many systems, deadlock does not occur until the message is very long (often 128 KBytes or more). For EUI, the size is (roughly) 128 bytes (*not* KBytes) and for EUI-H, the size if (again roughly) 4 KBytes. The limit for Chameleon is the same as the underlying transport layer (i.e., the EUI or EUI-H limits).

To fix this you have several choices:

- Reorder your send and receive calls so that they are pair up. For example, if there are always an even number of processors, you could use

  ```
  if (myid is even) {
     send( to=partner, data, len, tag )
     recv( from=partner, data, maxlen, tag )
     }
  else {
     recv( from=partner, data, maxlen, tag )
     send( to=partner, data, len, tag )
     }
  ```

- Use non-blocking sends and receives instead
- If you are using Chameleon, you can use the command line argument `-eui nsend=0` to cause Chameleon to provide buffering from user-space for all blocking sends. This works with both the EUI and EUI-H transport layers.

Chapter 11

Reporting Problems

If you believe that the SP-1 has a hardware or software problem, please send mail to `spsupport`. Please be as specific as you can; if relevant, include samples of code or output.

The tools `xsp1load` shows down nodes as red; this means that the node is not responding to `rup`. Since a node may not respond to `rup` but still be available, you can check to see if nodes are down by using `rsh`. For example, if you believe `spnode47` is down, try

```
rsh spnode47 date
```

If after a minute there is no response, or you get a message other than the date, then you should report the node as down to spsupport. *Exception* if you get a *permission denied* message, check the schedule (in '`/u/schedule/schedule`') to see if the machine is in single user mode.

The mail alias `spusers` should be used for sending mail to fellow users of the SP-1. This list is appropriate for mail about new tools, user-group meetings, pleas for reducing disk space usage, and the like.

If the '`/u`' or a '`/tmp`' file system fills up, you should do the following

1. Use the command `df /u`, `xspony`, `df /tmp`, or `xsp1df` to verify that the file system really is full.

2. Remove whatever you can. The command

   ```
   du -k /u/$LOGNAME
   ```

   will show (in kilobytes) the disk usage of all of your subdirectories.

3. Send a message to the other SP1 users alerting them to the fact that '`/u`' is full. This can be done by sending a message to `spusers@mcs.anl.gov`. You might want to include (some) of the output of

```
du  -k -s /u/* | sort -r -n | head
```

which will show the disk usage of each user (note that /u/pass is a mounted file system and hence does not take space away from the /u file system).

4. For a given '`/tmp`', it is probably better to inform only the few other users using that '`/tmp`'.

Chapter 12

Miscellaneous

12.1 Selecting EUI-H Nodes

Programs may select the nodes to run on under EUI-H by setting the environment variable `EUIHOSTS` to a *space-separated* list of node names. For example,

```
setenv EUIHOSTS "spnode65 spnode66 spnode67"
```

12.2 Selecting Interrupt-Driven EUI-H

By default, EUI-H uses a polling method to check for incoming messages. This has the advantage of reducing the number of interrupts that must be handled, but can in some cases introduce long delays as processes wait for messages to be delivered. Programs containing large amounts of nonblocking communications can benefit from setting the environment variable `EUICODE` to `xeui.intr` before running. This works with Chameleon and p4 as well as for EUI-H jobs started with `cotb0`. The polling version is `xeui`; to restore it after choosing the interrupt version, either `unsetenv EUICODE` or set `EUICODE` to `xeui`.

12.3 Interrupt-driven receives in EUI-H

As an *experimental* feature, our version of EUI-H includes a way to specify that a function should be called when a message of a particular type is received.

12.3.1 Interrupt-driven receive routines

Here are dscriptions of the routines. These descriptions were provided by Peter Hochschild and have been edited slightly.

```
rcvncall(ptr, blen, src, type, id, handler)
void *ptr;
int  *blen, *src, *type, *id;
void (*handler)(int *id);
```

Posts a receive buffer `ptr, blen, src, type, id` exactly as for `MP_RECV` and attaches the named handler. When a matching message is delivered, the handler is called (and passed as its sole argument the message id of the arrived message).

```
void handler(id)
int *id;
```

`id` contains the message id returned by the `rcvncall`. The handler must execute an `MP_WAIT` for this message id. The `MP_WAIT` (which returns the actual message length) is guaranteed to return more or less immediately since the message has arrived.

The handler is permitted to make any EUI-H calls.

Since the handler is potentially called by a signal handler, it must live by all the usual restrictions that apply to signal handlers (eg regarding buffered I/O, etc). In particular, it should do *no* I/O.

```
lockrnc(new, old)
int *new, *old;
```

`lockrnc` permits disabling and enabling of dispatching of `rcvncall` handlers. If `new=1`, handler dispatching is suspended. If `new=0`, handler dispatching is resumed. The previous lock state is returned via `old`.

Upon entry to a RECVNCALL handler, dispatching of handlers is suspended. It is resumed upon return from the handler.

12.3.2 Notes on interrupt-driven receives

1. Handler dispatching is suspended while executing EUI-H calls. In particular, handlers are not dispatched while executing blocking message passing calls. Of course the user is free to avoid blocking calls (by using non-blocking message passing calls and `MP_STATUS`). Note that the CCL is implemented with blocking message passing.

2. The underlying transport mechanisms make no distinction between messages that result in handler execution and those that don't. All the usual rules about in-order delivery apply collectively to both kinds of messages.

12.3.3 Interrupt-driven receive example

This section includes a complete example program that uses the interrupt-driven receives.

```
#include <stdio.h>

/* Program to test interrupt-driven receive in EUI-H */

static int done = 0;
static int len2 = 0;

void handler( id )
int *id;
{
int np, myid;

mp_environ( &np, &myid );
mp_wait( id, &len2 );
done = 1;
}

main(argc, argv)
int argc;
char **argv;
{
int len, src, type, id, to, np, myid, ptr[10];

mp_environ( &np, &myid );
len  = 10 * sizeof(int);
type = 0;
src  = -1;

rcvncall( ptr, &len, &src, &type, &id, handler );

if (myid == 0) {
    to = np - 1;
    mp_bsend( ptr, &len, &to, &type );
    while (!done) ;
    }
else if (myid == np - 1) {
    to = 0;
    mp_bsend( ptr, &len, &to, &type );
    while (!done) ;
    }
printf( "[%d] received %d bytes\n", myid, len2 );
}
```

Acknowledgments

The work described in this report has benefited from conversations with and use by a large number of people. Steven Tuecke provided the descriptions for Fortran M and PCN. Thanks to the many people who provided comments on the early draft of this document. In particular, Bill Nickless and Paul Plassmann made a number of useful suggestions.

Bibliography

[1] 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).

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

[3] Message Passing Interface Forum. Document for a standard message-passing interface. Technical Report CS-93-214, University of Tennessee, November 1993.

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

[5] Virginia Herrarte and Ewing Lusk. Studying parallel program behavior with Upshot. Technical Report ANL-91/15, Argonne National Laboratory, August 1991.

[6] Peter Hochschild. *Using EUIH: An Experimental EUI Implementation.* IBM, 1993.

[7] IBM. *IBM AIX Parallel Environment Parallel Programming Subroutine Reference Release 1.0*, 1993.