Intel Corporation
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-6497

(503) 696-8080

intel®

April 1996

Dear Paragon™ System Customer:

The Paragon™ XP/S supercomputer can accommodate thousands of advanced microprocessors connected in a two-dimensional rectangular mesh. The Paragon XP/E supercomputer is a distributed-memory multicomputer that can accommodate up to thirty advanced compute nodes connected in a two-dimensional rectangular mesh.

The interconnect network offers high-bandwidth, low-latency communication and frees programmers from having to concern themselves with interconnect topology. The operating system brings the Open Software Foundation's industry standard version of the UNIX operating system to the performance-driven environment of scalable, distributed-memory computing.

Thank you for joining the fast-growing community of scientists and programmers now taking advantage of Paragon systems to tackle problems not before possible at an affordable price.

This package contains the system software for the Paragon system. Please read through the documentation and distribute the materials to those intending to use the system.

Before using your system:

- **Read this letter completely.**
- **Verify the contents of this package.**
- **Read the *Paragon™ System Software Release 1.4 Release Notes.***

Package Contents

The operating system software is provided on three 0.25-inch 12000 BPI cartridge tapes in UNIX **tar** format. These tapes are listed in Table 1.

Table 1. Installation Media

Description	Order Number
Paragon™ System Software Release 1.4	650928-001
Paragon™ SAT Software Release 1.4	650929-001
Paragon™ ParAide Software Release 1.4 Native, Sun4/SunOS-4 and Silicon Graphics Hosted Development Environment Tools	650930-001

Diagnostic and compiler software are packaged separately If you are a brand new customer, receiving Paragon system software for the very first time, Table 2 lists the included documentation.

Table 2. Documentation (1 of 3)

Description	Order Number
Paragon™ System Software Release 1.4 Release Notes	654100-001
Paragon™ System Software Installation Guide	654099-001
OSF/1 Documentation Errata	312857-003
Paragon™ System Technical Documentation Guide	312820-004
User Group Membership Pack	638535-001
Paragon™ System Acceptance Test User's Guide	312648-005
Paragon™ System Administrator's Guide	312544-005
Paragon™ System Application Tools User's Guide	312545-005
Paragon™ XP/S System Cabling Guide (only Paragon XP/S system customers)	312823-002
Paragon™ System Commands Reference Manual	312486-005
Paragon™ System C Compiler Release 5.0 Release Notes	633948-001
Paragon™ System C Compiler User's Guide	312490-003
Paragon™ System C Calls Reference Manual	312487-005
Paragon™ System Fiber Distributed Data Interface Installation and Configuration Guide	312814-003

Table 2. Documentation (2 of 3)

Description	Order Number
Paragon™ System Fortran Compiler Release 5.0 Release Notes (only Paragon XP/S MP system customers)	633953-001
Paragon™ System Fortran Compiler User's Guide (only Paragon XP/S MP system customers)	312491-003
Paragon™ System Fortran Language Reference Manual (only Paragon XP/S MP system customers)	312644-002
Paragon™ System Fortran Calls Reference Manual (only Paragon XP/S MP system customers)	312488-005
Paragon™ System Graphics Libraries User's Guide	312887-001
Paragon™ XP/S System Hardware Installation Manual (only Paragon XP/S system customers)	312543-004
Paragon™ XP/E System Hardware Installation Manual (only Paragon XP/E system customers)	312843-004
Paragon™ XP/S System Hardware Maintenance Manual	312822-003
Paragon™ System High-Performance Parallel Interface Manual	312824-004
Paragon™ System Interactive Parallel Debugger Reference Manual	312547-005
Paragon™ XP/S i860™ 64-Bit Microprocessor Assembler Reference Manual	312546-001
Paragon™ System Multi-User Accounting and Control System Manual	312891-004
Paragon™ System Network Queueing System Manual	312645-005
Paragon™ System Performance Visualization Tool User's Guide	312889-003
i860™ Microprocessor Family Programmer's Reference Manual	240875-002
Paragon™ XP/S System RAID Utilities Manual	312646-003
Paragon™ XP/S System Site Preparation Guide (only Paragon XP/S system customers)	312485-004
Paragon™ XP/E System Site Preparation Guide (only Paragon XP/E system customers)	312842-003
Paragon™ System User's Guide	312489-005

Table 2. Documentation (3 of 3)

Description	Order Number
C: A Reference Manual	313152-001
The C Programming Language	122008-002
Effective Fortran 77 *(only Paragon XP/S MP system customers)*	312201-001
X Protocol Reference Manual	312654-001
Xlib Programming Manual	312655-001
Xlib Reference Manual	312656-001
X Toolkit Intrinsics Reference Manual	312658-001
X Toolkit Intrinsics Programming Manual	312657-001
Open Desktop User's Guide	312954-001
Open Desktop Administrator's Guide	312955-001
OSF/1 Network Application Programmer's Guide	PSCOSF1M[1]
OSF/1 Command Reference	PSCOSF1M[1]
OSF/1 Programmer's Reference	PSCOSF1M[1]
OSF/1 System and Network Administrator's Reference	PSCOSF1M[1]
OSF/1 User's Guide	PSCOSF1M[1]
Using MPI	654094-001
MPI: The Complete Reference	654096-001

[1]This product code designates all five volumes of the OSF/1 documentation set.

If you are an existing customer, receiving an update, Table 3 lists the included documentation. This list includes all manuals and release notes that have changed since your last shipment.

Table 3. Updated Documentation (1 of 2)

Description	Order Number
Paragon™ System Software Release 1.4 Release Notes	654100-001
Paragon™ System Software Installation Guide	654099-001
Paragon™ System Technical Documentation Guide	312820-004
User Group Membership Pack	312805-001

Table 3. Updated Documentation (2 of 2)

Description	Order Number
Paragon™ System Acceptance Test User's Guide	312648-005
Paragon™ System Administrator's Guide	312544-005
Paragon™ System Application Tools User's Guide	312545-005
Paragon™ System Commands Reference Manual	312486-005
Paragon™ System C Calls Reference Manual	312487-005
Paragon™ System Fortran Calls Reference Manual (only Paragon XP/S MP system customers)	312488-003
Paragon™ XP/S System Hardware Installation Manual	312543-005
Paragon™ XP/E System Hardware Installation Manual (only Paragon XP/E system customers)	312843-004
Paragon™ XP/S System Hardware Maintenance Manual	312822-003
Paragon™ System High-Performance Parallel Interface Manual	312824-004
Paragon™ System Interactive Parallel Debugger Reference Manual	312547-005
Paragon™ System Multi-User Accounting and Control System Manual	312891-004
Paragon™ System Network Queueing System Manual	312645-005
Paragon™ System User's Guide	312489-005

If items are missing, or if you have any questions, please contact Intel Scalable Systems Division immediately. Please refer to the section "Comments and Assistance" in this letter for information about how to contact Intel Scalable Systems Division.

What is in Release 1.4?

The Paragon System Release 1.4 software is the latest release of the operating system. It includes the X Window System, online manual pages, and manuals in both hardcopy and PostScript formats. For a list of features in this release, refer to the *Paragon™ System Software Release 1.4 Release Notes.*

Installation

For directions on how to install the Paragon System Release 1.4 software, refer to the *Paragon™ System Software Release 1.4 Release Notes.*

NOTE

Adding or removing any boards or components from your Paragon system can damage the system and may invalidate your warranty. Please contact Intel Scalable Systems Division Customer Support for assistance in answering your questions.

Restrictions and Limitations of Release 1.4

Every effort has been taken to ensure the quality of this release, but at shipment we are aware of some limitations. Please refer to the *Paragon™ System Software Release 1.4 Release Notes* for known limitations and available workarounds.

Comments and Assistance

We are eager to hear of your experiences with the Paragon system. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

United Kingdom Intel Corporation (UK) Ltd.
Scalable Systems Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056
(44) 793 431062
(44) 793 480874
(44) 793 495108

Intel Japan K.K.
Scalable Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Scalable Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 677-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 677-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com (Internet)

Intel Supercomputer Users' Group

The Intel Supercomputer Users Group promotes the exchange of information among users. Intel strongly supports the Users Group and encourages participation in its activities, which include: Special Interest Groups (SIGs), an annual international users conference, an electronic mail task force, and a "freeware" library of user-contributed software, available electronically to all members of the Intel Supercomputer Users' Group. For membership information contact:

JoAnne Wold (503-677-5322)
joanne@ssd.intel.com (Internet)

Again, thank you for acquiring a Paragon Supercomputer.

Sincerely,

Peter Wolochow

Product Marketing Manager
Intel Scalable Systems Division

April 1996
Order Number: 654100-001

Paragon™ System Software
Release 1.4
Release Notes

Intel® Corporation

WARNING

Some of the circuitry inside this system operates at hazardous energy and electric shock voltage levels. To avoid the risk of personal injury due to contact with an energy hazard, or risk of electric shock, do not enter any portion of this system unless it is intended to be accessible without the use of a tool. The areas that are considered accessible are the outer enclosure and the area just inside the front door when all of the front panels are installed, and the front of the diagnostic station. There are no user serviceable areas inside the system. Refer any need for such access only to technical personnel that have been qualified by Intel Corporation.

CAUTION

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

LIMITED RIGHTS

Preface

These release notes contain guidelines and limitations to keep in mind when using the operating system software. The release notes also contain a list of open and fixed bugs.

Organization

Chapter 1	Describes guidelines and limitations that you should know before using the operating system software.
Chapter 2	Contains an open bug list and a fixed bug list. The open bug list lists open bugs against the operating system software. The fixed bug list lists bugs fixed since the last release.
Appendix A	Describes changes made to how the allocator works.

Notational Conventions

This manual uses the following notational conventions:

Bold	Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.
Italic	Identifies variables, filenames, directories, processes, user names, and writer annotations in examples. Italic type style is also occasionally used to emphasize a word or phrase.
`Plain-Monospace`	Identifies computer output (prompts and messages), examples, and values of variables. Some examples contain annotations that describe specific parts of the example. These annotations (which are not part of the example code or session) appear in *italic* type style and flush with the right margin.

Bold-Italic-Monospace
Identifies user input (what you enter in response to some prompt).

Bold-Monospace
Identifies the names of keyboard keys (which are also enclosed in angle brackets). A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. For example:

<Break> **<s>** **<Ctrl-Alt-Del>**

[ ] (Brackets) Surround optional items.

. . . (Ellipses) Indicate that the preceding item may be repeated.

| (Bar) Separates two or more items of which you may select only one.

{ } (Braces) Surround two or more items of which you must select one.

Applicable Documentation

For information about the manuals shipped with the Paragon system, see the *Paragon™ System Technical Documentation Guide*.

Comments and Assistance

Intel Scalable Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

United Kingdom Intel Corporation (UK) Ltd.
Scalable Systems Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056
(44) 793 431062
(44) 793 480874
(44) 793 495108

Intel Japan K.K.
Scalable Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Scalable Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 677-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 677-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com
(Internet)

Table of Contents

Chapter 1
Features, Guidelines, and Limitations

Chapter 2
Bug Lists for System Software

Appendix A
Allocator Changes

List of Tables

Features, Guidelines, and Limitations 1

Introduction

The chapter contains the following information:

- Memory usage information for the operating system.
- New features with this release.
- Guidelines for using the operating system software.

CAUTION

Never shut down the diagnostic station before shutting down the Paragon system. Doing so would cause system errors and incorrectly halt the system.

Operating System Memory Usage

This section provides memory usage information for Release 1.4 of the operating system. Table 1-1 shows the amount of memory after a fresh boot. The numbers are scaled up to the next megabyte.

The amount of memory shown includes that taken up by the microkernel and the compute server, including SPV (which takes about 0.5M byte); SPV is assumed to be idle. The table also assumes that server symbols are not loaded; that is, the bootmagic BOOT_LOAD_SYMBOLS is 0. Server symbols are not loaded by default. This number in the table represents the amount of memory available for your application before paging commences.

Note that the memory used by the operating system increases with the physical memory on the board. This is primarily because larger page tables are required. Also note that these numbers do not include memory taken up by default message passing buffers, which is about 1M byte. Please refer to the application manual page for information about how to lower message buffer requirements for applications that do minimal message passing.

Table 1-1. Operating System Memory Usage

Physical Memory on Compute Node Board (megabytes)	Available Memory Before Paging of the Operating System Begins (megabytes)
16	7.0
32	22.75
64	53.4
128	115.3

New Features

This section highlights features that are new to Release 1.4.

SCSI-16 (SIO) Nodes

The SCSI-16 (also called an SIO node) node is a combination of an MP node base board, and the SCSI-16 daughterboard. The SCSI-16 node provides serial, Ethernet, and SCSI channel interfaces to the Paragon system. These interfaces are used to satisfy Paragon system external communication, diagnostics, and I/O requirements.

The serial channel uses an RJ-11 connector and the hardware supports an RS232 interface at rates up to 288 Kbps. The serial channel is normally only used by a boot node at boot time, or for diagnostic purposes during debug.

The Ethernet port uses an RJ-45 connector and supports the IEEE802.3 Ethernet specification. This interface supports the industry-standard IEEE 10BaseT protocol, which is a 10MHz twisted pair media standard.

The two SCSI channels use high-density 68-pin connectors, and the hardware supports 8-bit or 16-bit SCSI operations at rates up to 20M-bytes/sec. Each SCSI channel normally communicates with a single RAID controller and (optionally) with a DAT tape drive.

The SCSI channels are labeled 0 and 1 on the node front panel. The **makedev** command automatically recognizes devices installed on either channel. To add devices by hand, the minor number calculation includes either a 0 or 1 to select the appropriate channel, as indicated by <channel> in Table 11-1 of the *Paragon™ System Administrator's Guide*.

Specifying SIO channels in RAID Utilities

With the introduction of SCSI-16, a second channel is available on each SIO node. This channel is referred to with an additional directory "ch1" in a logical unit device name specification, as shown in the following example of the **can** command:

```
# can /dev/io1/ch1/scsi0
```

The new channel can also be accessed using "b1" in a hardware address, as in the following example of the interactive array configuration editor:

```
# ace -z m72c0b1p010
```

Using SIO as a Boot Node

In order to use the SIO node as a boot node, you must change the boot node specification in the system configuration files, add new bootmagic variables to *MAGIC.MASTER* and/or *MAGIC.md*, change cables to allow the diagnostic station to communicate directly with the SIO node, and modify the **reset** script.

Configuration Files

If the SIO node is a boot node, the boot disk must be attached to channel 0. Using channel 1 for data is optional. Add lines like the following to the *DEVCONF.TXT* file and then re-run the */u/paragon/diag/config* script to recreate the SYSCONFIG.TXT file:

```
SIO 00D03 H04
RAID 00D03.ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3 CTLR0
```

Bootmagic variables

In order to use the SIO node as a boot node, you must add the following variable to the *MAGIC.MASTER* and/or *MAGIC.md* file(s):

```
ZONE_MAP_SIZE=<bootnode>33554432
```

If the SIO has 128M-bytes of RAM, use this ZONE_MAP_SIZE specification:

```
ZONE_MAP_SIZE=<bootnode>68157440
```

Cabling

When used as a boot node, the Ethernet port of the SIO card is connected to a 10-base-t Ethernet network via an 8-pin RJ45 connector. The serial port is connected to the diagnostic station, using a cable adapter (P/N) that mates the 9-pin D connector on the diagnostic station to the 6-pin RJ11 serial connector on the SIO card. You must use an SIO-specific RAID cable (P/N 340943-001) to connect to the 68-pin channel connectors. (The MIO cable appears identical, but is wired differently.) For installation details, refer to the *Paragon™ System Cabling Guide*.

reset Script

If the SIO node is a boot node, you must also modify the **reset** script. Edit the file *reset* in */usr/paragon/boot* on the diagnostic station and set *BOOT_VM_PAGESIZE*=8192. If your boot node is not an SIO node, do not change the setting of *BOOT_VM_PAGESIZE*. Leave it at 4096.

NQS Features

For detailed information about NQS, refer to the *Paragon™ System Network Queueing System Manual*. The following new features are documented in the *Paragon™ System Network Queueing System Manual*.

Defined Scheduling Priorities

- The **qmgr create batch_queue** subcommand has new (optional) **epl_nprime** and **epl_prime** parameters that explicitly set the queue's EPLs during queue creation (the default is 1 during prime time and 3 during non-prime time).
- Two new **qmgr** subcommands, **set epl_prime** and **set epl_nprime**, which can be used to change an existing queue's EPLs.
- The **qmgr show long queue** subcommand and **qstat -bl** have been modified to display the queue's prime and non-prime EPLs.
- NQS partitions are now always created with the appropriate EPL, and all NQS partitions (not just those that overlap the interactive partition) change their EPLs.
- Two new **sched_param** parameters, **ncur_open_epl** and **cur_open_epl**, set the EPLs for the currently active and inactive interactive partitions.

Dynamic Job Priority

Five new **sched_param** parameters have been added, which are tunable to allow smaller or shorter jobs to tend to run before larger or longer jobs:

node_weight
node_base
time_weight
time_base

Converting the Release 1.3 Queues Database

The NQS databases must be converted from R1.3 format to R1.4 format. NQS should not be running when the queues are updated. If NQS is running, stop it with the following command before updating the database:

```
# /sbin/init.d/nqs stop
```

The utility */usr/lib/nqs/setup/nqs_queue_upgrade* can be used to convert the queues database files to the new format. You must be a root user to user this utility, which takes no arguments. The old database files are saved with the extension ".bak".

The NQS database may also be converted by doing a complete NQS setup with the */usr/lib/nqs/setup/nqs_setup* utility.

Interactive Parallel Debugger (IPD)

Debugging Threaded Applications

IPD allows viewing of information from multiple threads in a process. To do this, use the **threads** command with the **-on** switch. When threads mode is on, commands like **display**, **frame**, and **process** print information for each thread in each process in the current context.

Debugging MPI Applications

An application that successfully executes a call to *MPI_Init()* is recognized as being an MPI application by IPD. This causes IPD to change the manner in which a process is named in context displays from the NX node/ptype pair to the MPI communicator/rank pair. Once *MPI_Finalize()* is executed the context displays revert to the NX style. The **msgstyle** command with the **-nx** switch can be used to force context displays to use the NX style, and the **-mpi** switch returns to MPI mode.

The **commshow** command lists all of the communicator handles available for use in a context specification, or the communicator handle for a particular communicator in the application.

Listing Executable Code Lines

Code listings in IPD (and XIPD) now identify executable lines (those that may be used with breakpoints and tracepoints) with a ">" character.

Redirecting Output

The **load**, **run**, and **rerun** commands now allow you to direct the output of the debugger to a file by specifying the outfile on the command line. Refer to the *Paragon™ System Interactive Parallel Debugger Reference Manual* for details.

ParAide

The shell area of the ParAide main window did not previously provide support for interactive jobs and job control. ParAide now provides a terminal area that runs like an embedded *xterm*, providing support for interactive jobs and job control.

In addition, a status line has been added to the bottom of the ParAide main window. A status message is displayed when any of the following occur.

- A graphical tool is started.
- A load command is sent to the terminal.
- An editor window is brought up.
- A command that does not bring up the Command Viewer is started from the Command menu.

Finally, the ParAide main window now provides a Command menu that you can customize. The items in the Command menu are defined by an application resource, so you can modify the menu to contain any non-interactive commands you wish. The commands can execute directly or prompt you. The output of the commands can be discarded or loaded into a dialog for viewing. For a complete description of the Command menu and how to customize it, refer to the ParAide online help for the Command menu.

The Paragon C++ Compiler and Cfront

If you have a **cfront** source license, you may have received a version of **cfront** from Intel for your Paragon system. Please note that **cfront** may not work as expected after installing Release 1.4 of the operating system. No updates to **cfront** are planned for Paragon systems. The best way to continue using C++ on Paragon systems is to install the Paragon system C++ compiler. This compiler has **cfront** compatibility switches to help you when porting your code.

XIPD

Debugging Threaded Applications

XIPD allows viewing of information from multiple threads in a process. To do this, enable the **threads** item in the display menu. When threads mode is on, the **display**, **frame**, and **process** dialogs show information for each thread in each process in the current context.

Debugging MPI Applications

An application that successfully executes a call to *MPI_Init()* is recognized as being an MPI application. This causes XIPD to change the manner in which a process is named in context displays, from the NX node/ptype pair to the MPI communicator/rank pair. Once *MPI_Finalize()* is executed the context displays revert to the NX style. Refer to the *Paragon™ System Application Tools User's Guide* for complete information about the new MPI-specific parts of XIPD.

The **Show Communicator** dialog lists all of the communicator handles available for use in a context specification, or the communicator handle for a particular communicator in the application.

Listing Executable Code Lines

Code listings in IPD (and XIPD) now identify executable lines (those that may be used with breakpoints and tracepoints) with a ">" character.

prof/gprof

The profiling tools combine the execution times from routines in the *libpm* library into a single entry, title <overhead>. In **gprof**'s index display, routines from the *libpm* library are surrounding by angle brackets "<>".

The underbars added by the linking loaded are removed in the **prof** and **gprof** displays.

A new "%Parallel" field in the output from **prof** shows thread execution information for applications running on nodes with two or more arithmetic CPUs. A new **-u** switch may be used to disable this enhanced data display.

Changes in File Location

The Paragon system files in diagnostic station's */usr/local* directory have been moved in R1.4. System administrators should move the Paragon system files and subdirectories from the */usr/local* directory as follows:

- The files formerly in */usr/local/bin/what_patches* have been moved to */bin/what_patches* in the diagnostic station.

- All the other Paragon system files in */usr/local/bin* have been moved to */usr/paragon/bin*:

  /usr/local/bin/scanio
  /usr/local/bin/fscan.hlp
  /usr/local/bin/fscan
  /usr/local/bin/async
  /usr/local/bin/bootpp
  /usr/local/bin/parsemagic
  /usr/local/bin/what_patches
  /usr/local/bin/perl
  /usr/local/bin/taintperl
  /usr/local/bin/a2p
  /usr/local/bin/cppstdin
  /usr/local/bin/h2ph
  /usr/local/bin/c2ph
  /usr/local/bin/pstruct
  /usr/local/bin/s2p
  /usr/local/bin/find2perl

- All the Paragon system files in */usr/local/lib/perl* have been moved to */usr/paragon/lib/perl*:

  /usr/local/lib/perl
  /usr/local/lib/perl/abbrev.pl
  /usr/local/lib/perl/assert.pl
  /usr/local/lib/perl/bigfloat.pl

/usr/local/lib/perl/bigint.pl
/usr/local/lib/perl/bigrat.pl
/usr/local/lib/perl/cacheout.pl
/usr/local/lib/perl/chat2.pl
/usr/local/lib/perl/complete.pl
/usr/local/lib/perl/ctime.pl
/usr/local/lib/perl/dumpvar.pl
/usr/local/lib/perl/exceptions.pl
/usr/local/lib/perl/fastcwd.pl
/usr/local/lib/perl/find.pl
/usr/local/lib/perl/finddepth.pl
/usr/local/lib/perl/flush.pl
/usr/local/lib/perl/getcwd.pl
/usr/local/lib/perl/getopt.pl
/usr/local/lib/perl/getopts.pl
/usr/local/lib/perl/importenv.pl
/usr/local/lib/perl/look.pl
/usr/local/lib/perl/newgetopt.pl
/usr/local/lib/perl/open2.pl
/usr/local/lib/perl/perldb.pl
/usr/local/lib/perl/pwd.pl
/usr/local/lib/perl/shellwords.pl
/usr/local/lib/perl/stat.pl
/usr/local/lib/perl/syslog.pl
/usr/local/lib/perl/termcap.pl
/usr/local/lib/perl/timelocal.pl
/usr/local/lib/perl/validate.pl

See step 3, Modifying .profile, on page 2-5 of the *Paragon™ System Software Release 1.4 Installation Guide* for an installation change relating to this new feature.

If any user application in the diagnostic station uses *perl* in */usr/local* directory, the application needs to be modified to point to the */usr/paragon/bin* and */usr/paragon/lib* directories.

Guidelines and Limitations

This section contains guidelines for using the Paragon system. These guidelines are pertinent for both a system administrator and a programmer. The section also lists some the most important limitations. A complete list of bugs in a reference format is in Chapter 2, "Bug Lists for System Software."

Booting the Paragon™ System

We strongly recommend booting your Paragon system with kernel assertions off.

Please contact SSD Customer Support for information about which operating system kernel is best for the applications running on your system. See the diagnostic station online **reset** reference page for information about using the **reset** command to boot a Paragon system. To boot with assertions, one must specify **reset debug**.

Recompile and Relink Application Code

When a new release of the system software is installed on your system, recompile and relink your application code using the compilers and system software libraries provided with the new release. You need to do this because executables from different releases are not compatible.

General Programming Guidelines

This software release does not require workarounds for most code that runs on fewer than 128 nodes. If you are running on a larger number of nodes or if you experience difficulty with your program, the following guidelines may be helpful.

- Dynamically allocate memory at run time, especially if the application requires more than 10M bytes of memory. Use **malloc**() for C and **ALLOCATE** for Fortran. Typically, dynamic allocation gives a shorter startup execution time.
- By keeping the runtime size of an application within the available memory as shown in Table 1-1 (under 22M bytes for 32M-byte nodes), you can prevent excessive paging in an application.
- When setting up a paging tree, maintain a maximum ratio of 32 compute nodes per disk subsystem.
- Never use an **nx_initve**() or **nx_initve_rect**() system call in a program running in the compute partition.

Do Not Use Mach System Calls

Do not use the Mach system call interface. Only use calls described in the Paragon system documentation. The Mach interface is not documented in Paragon system manuals, but you may read about it elsewhere. Using Mach system calls bypasses the resource limits and programming model of the Paragon system. Intel will neither diagnose nor correct problems that may develop with your application if you use the Mach interface. Mach system calls performing memory allocation are the most likely to cause problems.

Please note that this restriction is not intended to include the *pthread* library calls. The *pthread* interface is supported and documented.

Avoid Using Large Statically-Allocated Data Structures

Pages for statically-allocated data structures are touched at load time. This can substantially increase the load time for applications with large statically-allocated data structures. Use dynamically-allocated data structures when possible (for example, use **malloc**() for C or **ALLOCATE** for Fortran).

SMP Programming

Symmetric Multiprocessing(SMP) programs are multi-pthreaded programs that run on one or more processors. A single image of the operating system runs on these processors. Also, none of the pthreads has any specialized access to the hardware.

Namelist Groups

Namelist groups are used in Fortran programs. Local variables that appear in a namelist group are not placed on the stack, even if the module is compiled with **-Mreentrant**. They are statically allocated. If you compile with **-Mncall** (which allows loops with procedures to be parallelized), then you should check that the variables in the namelist group do not introduce unwanted thread dependences. Treat the namelist variables as if they are declared in a **SAVE** statement.

-Msave and -Mreentrant

-Msave has the opposite effect of **-Mreentrant**. Programs compiled with **-Msave** have their local variables static, not stack-based. When compiling procedures intended to be thread-safe, do not use **-Msave** with **-Mreentrant**.

Pthread-Specific Data

Pthread-specific data are bound to a pthread key with **pthread_setspecific**(). The key is created with **pthread_keycreate**(). Note that all pthread-specific data for a process reside in one memory page, which on Paragon systems is 4K bytes. Hence, you may have **pthread_keycreate**() fail with an ENOMEM when your system still has memory available.

setjmp() and longjmp() Functions

If automatic and register variables are changed after the **setjmp**() (or **sigsetjmp**()), their values are indeterminate after the **longjmp**() (or **siglongjmp**()). You cannot assume that these variables are restored to what they were when the **setjmp**() (or **sigsetjmp**()) was called. Nor can you assume that they are set to whatever value they had when the function sequence containing the **longjmp**() (or **siglongjmp**()) was started.

If you want them to retain the values they had when the function sequence containing the **longjmp**() (or **siglongjmp**()) is started, declare them as **volatile**.

Global Operations

The global operations (**g...**() calls) are not tuned for maximum performance in the Release 1.4 system software. Only minor performance improvements can be expected over the Release 1.2 system software. A side effect is that the global operations may scale in unexpected ways. For example, the **gdsum**() call may exhibit a substantial linear component when purely logarithmic scaling might be expected.

SIGUSR2 Signal

The **SIGUSR2** signal value is not available to parallel applications because the system uses it to implement gang scheduling. This is true whether or not gang scheduling is used.

Loading the Mach Library

When using the **ld** link editor, the **-lmach** command line switch must be used to load the *libmach.a* library. If you use the **cc** driver to compile and link your application the library is loaded automatically, and it is not necessary to specify it manually.

Message Passing

This section provides release information for operating system message passing.

Global Sends

Global sends use a -1 value for the *node* parameter in an operating system send system call. Global sends are now implemented using a tree-structured store-and-forward message passing strategy. This requires each node in an application to completely receive the message, otherwise, the next node in the tree will not receive the message. Use any of the operating system receive calls to do the receive.

Synchronous Sends on Large Paragon™ Systems

An application that uses the **csend**() call for synchronous message passing can block or hang if you do not allocate enough memory for the application to handle messages. The memory allocated to a given node for message passing is based on the number of nodes in the application and the total memory available to the application for message passing. Therefore, as the number of nodes for an application increases, the memory available to each node for message passing decreases. With a large application running on a large Paragon system, this decrease can happen faster than expected, causing the application to block. For example, an application that uses the **csend**() call for message passing may run fine on 64 nodes, but have problems running on 128 nodes, because there is not enough memory for message passing. The following strategy can prevent this problem from happening:

- Post a receive before doing the synchronous send.
- Use a non-blocking send, for example, the **isend**() call.
- Decrease the message size for the application.
- Increase the memory buffer size for the application using the **-mbf** switch.
- Use the **-noc** switch to control the number of correspondents.

Setting the Process Type of a Controlling Process

Do not call **setptype**() in a controlling process that does not do message passing, because this call assigns memory for message buffering that will be unused in this process. This memory is wired-in; which means it cannot be paged.

Message Handlers

You must use **masktrap**() around any code in the main program that could interfere with calls in a handler established with one or more **h...**() calls.

Because it is often not obvious which calls could interfere with each other, use **masktrap**() to protect *all* library calls in the rest of the program that could call the same subsystems as the calls in the handler while the handler is active. For more information, see the discussion of **masktrap**() in the *Paragon™ System User's Guide.*

Forced Message Types

If you use a forced message type and don't post the receive in advance then the message is dropped. The message is also dropped if the receiving buffer is paged out. If the application is run with the **-plk** switch, the receiving buffer is guaranteed to be in memory. Forced messages have no performance advantage on Paragon systems.

Forced messages are useful for backward compatibility. Because earlier systems (iPSC/860 systems) did not provide paging, any program trying to be backward compatible should either use **-plk** or not use forced messages at all.

Using the Allocator

With Release 1.4, you now specify how the allocator controls partitions and applications using the allocator configuration file */etc/nx/allocator.config*, instead of using command line switches. This functionality is described in the **allocator** and **allocator.config** manual pages in the *Paragon™ System Commands Reference Manual.* You can also find information on this in the *Paragon™ System Administrator's Guide.*

Two major changes to the way that the allocator works are described in Appendix A.

I/O System

This section provides release information for the I/O system.

Maximum Compute Nodes Per I/O node

Applications requiring high performance should not attempt simultaneous I/O from greater than 32 compute nodes to any one I/O node (MIO) at one time (the compute node/I/O node ratio). This is true for UFS, NFS, and PFS (striped on one I/O node). The default configuration accepts compute node/I/O node ratios less than or equal to 32. Compute node/I/O node ratios greater than the configured amount can cause system hangs when multiple nodes are accessing that same I/O node.

To allow for larger compute node/I/O node ratios, set the bootmagic variable *NORMA_RDMA_GROUP_XMIT_ALLOC* to the desired compute node/I/O node ratio plus 16. For example, to allow for a compute node/I/O ratio of 48, set *NORMA_RDMA_GROUP_XMIT_ALLOC* to 64. The larger *NORMA_RDMA_GROUP_XMIT_ALLOC* is, the more kernel memory is used. This is memory that you cannot use for an application and that cannot be paged. Hence, it is desirable not to use more than you need.

The system administrator must be careful to configure the PFS file systems so that the compute node/I/O node ratio is enforced across multiple simultaneous applications.

For more information about configuring PFS file systems, see the *Paragon™ System Administrator's Guide*. You must understand PFS I/O modes and stripe attributes in order to code your applications within this restriction. For more information about programming PFS applications, see the *Paragon™ System User's Guide*.

Logical Volume Manager

Although documented in the OSF/1 manuals, the logical volume manager (LVM) is inappropriate for a Paragon system. The LVM software and online manual pages have been removed from the system. The LVM software is a set of resources that OSF/1 provides for managing disk storage in a system. The following LVM commands are not installed on the Paragon OSF/1 system:

lvchange	**lvremove**	**pvmove**	**vgreduce**
lvcreate	**lvsync**	**vgchange**	**vgremove**
lvdisplay	**pvchange**	**vgcreate**	**vgsync**
lvextend	**pvcreate**	**vgdisplay**	
lvreduce	**pvdisplay**	**vgextend**	

File System Checking

When you boot the system, the script */sbin/bcheckrc* runs the **fsck** utility on each file system defined in */etc/fstab* (such as */home*, */pfs* and the PFS stripe directories typically mounted on */home/.sdirs/vol**). The script */sbin/bcheckrc* assumes that the following is true in the */etc/fstab* file:

- The root file system (*/*) is the only file system at pass 1.
- The user file system (*/usr*) is the only file system at pass 2. Also, any other file system needed for booting should be added to pass 2.
- Only boot-node RAID file systems (excluding */*, */usr*) are at pass 3.
- Pass 4 is for non-boot-node I/O nodes. Because of the way *checkrc* is written, file systems with passes greater than pass 4 are checked the same time as pass 4.

In the case of a severely damaged file system, the **fsck** will fail and a message will be displayed to the screen. Because the message scrolls off the screen before the reboot is complete, you may not notice it. The system will continue to boot to a multiuser state with no other indication that anything is wrong.

If the **fsck** step fails, the specified directory will not be mounted on the affected file system. Any files written to that directory will be written into the file system that contains the parent directory. Later, if you notice that the file system did not mount and you **fsck** and mount it, the files that had been written into the directory become hidden. To avoid this problem, after each boot you can manually compare the mounted file systems with the contents of the *fstab* file.

For example, if the PFS stripe directory */home/.sdirs/vol1* does not mount because the **fsck** step failed, then any data written to that stripe directory will actually be written to the file system that contains the */home* directory.

HIPPI Limitations

Use the bootmagic string *TCP_SPACE_SIZE* with caution. A value larger than 64K (65536) can cause the system to panic. Use this bootmagic string at your own risk.

(262144 ?)

IPI-3 Limitations

Under certain conditions, PFS striped across IPI-3 Unix partitions may cause the Paragon system to panic. For example, if PFS is striped across all the Unix partitions in an IPI-3 facility partition and if an attempt is made to create a PFS file greater than the available disk space with **lsize**, a panic occurs. The error message is as follows:

```
panic: getblk: size too big when IPI-3 PFS filesystem is full
```

The only response at this point is to reboot the Paragon system. If the system were operating properly, **lsize** would return with an error, allowing you or a system administrator to delete the file and free up disk space. For more information about IPI-3 refer to the *Paragon™ System High-Performance Parallel Interface Manual.*

Interactive Parallel Debugger (IPD)

The msgqueue Command and Global Sends

The **msgqueue** command may appear to be incorrectly reporting messages that have been sent but not received if the program being debugged is using global sends. The implementation of a global send is such that only a subset of nodes in the partition are initially sent the message. This subset then sends the message on to another subset and so on (spanning tree is used). The problem is that the send is not forwarded to the next subset until a receive of that message has completed on the node expected to do the forwarding send.

If you stop execution immediately after executing a global send and execute a **msgqueue** command for all nodes, you would expect to see an unreceived message waiting on every node. But what you see instead is just a few nodes with messages waiting. These are the subset of nodes targeted by the initial send. These nodes must now receive the message they were sent and then they will send that same message on to the next subset. The **msgqueue** command does not show the sends to the second level of the tree, as they have not occurred yet.

Performance Monitoring

IPD supports performance monitoring on 66 or fewer nodes. Performance monitoring data collection can be done on more nodes, but will occasionally cause the system to hang. To do performance monitoring, use the IPD **instrument** command with the **-prof**, **-gprof**, or **-paragraph** switches. For example:

```
(all:0)> instrument -prof
```

Performance monitoring of applications running on a large number of nodes can take a long time.

Scaling IPD

IPD does not currently scale well above 512 nodes.

Debugging Code Compiled with -Mconcur

Code complied with the **-Mconcur** switch may be difficult to debug, due to the limited number of executable lines available for breakpoints, tracepoints, or watchpoints. The compiler disables low levels of optimization when the **-Mconcur** switch is used.

Actionpoints in Multi-Threaded Applications

Processes may fail to hit actionpoints (breakpoints, tracepoints, or watchpoints) when debugging multi-threaded applications using IPD.

Multi-threaded applications include those that call any of the *hrecv()* family of functions and those compiled with **-Mconcur**, as well as those that access the *pthreads* library directly. The problems occur when continuing execution from an actionpoint. The first problem happens if two threads trigger an actionpoint at the same moment OR in rapid fire succession (one immediately after the other)—the debugger may get confused and interpret the second actionpoint incorrectly. This results in the loss of all actionpoint information for the process. The symptom in this case is the sudden disappearance of actionpoints for that process. This means that the process runs to unexpected points in the code, such as to the exit. The you can confirm that you have encountered this debugger error by confirming that the break, trace, and watch commands list no actionpoints after having just continued from an actionpoint.

The other problem is that during the process of resuming execution of one thread, other threads execute and possibly miss hitting an actionpoint. There is no indication when this occurs.

There is no workaround for either problem.

Graphical Tools

The graphical tools (SPV, ParaGraph, ParAide, XIPD, XProf, and XGprof) use Motif 1.2 for their interface, and Motif 1.2 display commands can crash non-current versions of Sun's XNeWS window server. If you are not running the current version of Sun's XNeWS window server, you must upgrade the XNeWS server in order to run these tools.

Due to the size of their binary executables, running multiple X applications slows down the system.

To improve performance when using the graphical tools, change the bootmagic variable *TCP_SPACE_SIZE* on the Paragon system from the default size of 4K to 64K. 64K is a better size when using X applications that transmit data through an Ethernet connection.

CAUTION

64K is the suggested upper limit for the *TCP_SPACE_SIZE* variable. Using a larger value can result in undesirable system side effects.

The ParAide Toolset

The ParAide toolset does not support SMP programming. If you perform IPD **prof** or **gprof** instrumentation, performance data will be collected. However, the percentage of time spent in the procedures (first three columns in **prof** output data) will only reflect the initial or main thread, while the call counts (fourth column in **prof** output data) will reflect performance data for all threads.

Paragraph and NX Handler-Invoking Routines

Currently, there is no support for identifying ParaGraph trace event information at the thread level. Thus, if both a main thread and a handler thread are generating trace information, both events will be interleaved in the resulting trace file with no means of sorting during post-processing. ParaGraph will report an error whenever it finds a sequence of events that is not properly sorted.

XIPD

XIPD should be run on 32M-byte service nodes. This is especially important when debugging applications that run on a large number of nodes, because the more nodes an application runs on the more memory XIPD uses.

XIPD, unlike IPD, does not support debugging single, non-parallel host-node processes. Because of this, XIPD only supports the analysis of core directories, not single host-node core files. IPD should be used to analyze single host-node core files.

XIPD may hang if an MPI application is loaded on an NX ptype other than 0.

The limitations described for the Interactive Parallel Debugger (IPD) elsewhere in this section also apply to XIPD.

SPV X Resources for Pre-X11R5 Servers

You may need to set the fonts in the resource file(s) to execute applications on a pre-X11R5 server on the workstation. For a description of the SPV X resources, refer to the *Paragon™ System Performance Visualization Tool User's Guide.*

SPV and Memory Usage

SPV computes memory usage as the total number of pages minus the current number of free pages. This does not give a complete picture of memory usage, because the operating system has a concept of an "inactive" pool separate from the free pool. This allows memory to become active quickly without the overhead associated with moving memory from the free pool to the active pool.

The operating system tries to keep pages in the inactive pool, so when the number of free pages gets small, a lot of paging occurs to reclaim pages from the inactive pool. This tends to increase the usage value over time, but should not be confused with a memory leak, where the amount of wired memory increases over time. For example, if someone logs in and then immediately logs out, a number of pages would be left in the inactive pool and SPV would show an increase in the usage percentage.

SPV Not Started If Configuration Includes SUNMOS

The */sbin/init.d/spv* script has been modified to not start the *spvdaemon* if an alternate kernel is specified to execute on the compute nodes. The script prints the following error message if BOOT_ALT_NODE_LIST is not null:

```
Spv data collection NOT started: no support for alternate OSs.
```

System Administration

This section provides release information for Paragon system administration.

No Disk Quota Support

The Paragon system does not provide disk quota support. This means that the commands **edquota**, **quotacheck**, **quataon**, **quotaoff**, and **repquota** are not available. The command **quot** is available; note, however, that *file system* must designate the raw device. Also, the **-f** and **-n** options on **quot** do not work.

PFS/SFS Filesystem Mounting Order

The mounting of a PFS filesystem and its stripe filesystems (SFS) now have an order of dependency. Prior to Release 1.4, the mounting and unmounting of SFS had no effect on the mounted PFS filesystem. Because of changes to PFS pathname caching, the SFS must now be mounted before mounting a PFS. The PFS must also be unmounted before unmounting the SFS. The SFS will not unmount (returning an FS busy error) until the PFS is unmounted.

Shutting Down the Paragon™ System

Use the following command sequence to reboot the system. In a console window:

```
# cd /
# shutdown now
# umount -A
# sync;sync;sync
# halt
```

This writes the system buffers, kills all running processes, brings the system to single-user mode, unmounts the file systems, and halts the system. You can safely power down the system after this sequence. To get back to the diagnostic station prompt, enter a **~q**.

To reboot the system, use the following command after the **halt** command completes.

```
DS# cd /usr/paragon/boot
DS# reset
```

For more information about system shutdown, see the *Paragon™ System Administrator's Guide*.

Backing Up a File System to the DAT Tape Drive

The following example allows you to efficiently use the **dump** command to back up a file system to the DAT tape drive on the Paragon system:

```
/usr/sbin/dump -f /dev/io0/rmt6 -c /dev/io0/rrz0a
```

The **-f** switch specifies using the */dev/io0/rmt6* storage device for the dump. The **-c** switch specifies that the dump medium is a cartridge tape. The command dumps the file system on */dev/io0/rrz0a* to the DAT drive.

If you specify the file system by its mount point name (for example, */home*) rather than by its raw disk name (for example, */dev/io0/rrz0f*), the specified file system must be listed in */etc/fstab*. If a specified mount point name is not in */etc/fstab*, the dump is aborted. The resulting error message does not readily indicate what the problem is. The error message is as follows:

```
.ioctl DIOMRINFO: Not a typewriter
dump: The ENTIRE dump is aborted
```

Preventing Core Dumps

User programs should not attempt to write to page 0. With Release 1.3, referencing a pointer to a location in page 0 caused a core dump; earlier versions of the operating system allowed this and returned 0.

A typical programming error (which is now longer allowed) is to dereference the NULL pointer that some calls (such as **malloc**()) return as an error condition. You can allow dereferences to page 0 by setting the bootmagic variable BOOT_PAGE0_ACCESS to 1 in */usr/paragon/boot/MAGIC.MASTER* and resetting the Paragon system, if dereferencing a pointer to page 0 does not cause your program to fail and you want to avoid the core dump. It is better programming practice, however, to write code that does not attempt to dereference page 0.

Shared Virtual Memory

The OSF/1 operating system supports a feature called *shared virtual memory* that can be used to share data between processes. Both the System V shared-memory calls and the mapped-memory interface are supported (see **shmget**(**2**) and **mmap**(**2**) in the *OSF/1 Programmer's Reference* for information about these calls). However, these calls can only be used to share memory between processes running on the *same node*. Any attempt to share memory between processes running on different nodes will fail.

NQS For Workstations

The Paragon System NQS Network Queueing System (NQS) supports a networked environment. However, the NQS executable files shipped with the operating system system software are *only* for Paragon systems. If you want to use NQS from your workstation, you have to separately obtain the NQS software for your workstation.

NFS For Workstations

The diagnostic station OS installation instructions do not currently point out that SCO UNIX requires that a "key" be entered during software installation. This is an activation key. If you use the same source media (floppies/tape) to load the SCO UNIX on another system, and then try to use NFS, the system detects that another system is on the network with the same activation key, and then NFS is "disabled" and refuses to work.

Bootmagic Variables

This section lists currently defined bootmagic variables. There are other bootmagic variables, but they are defined for internal use and should not be changed. Details about the *bootmagic* file and the bootmagic variables can be obtained from the *bootmagic* manual page in the *Paragon™ System Commands Reference Manual*. This manual page is available for online viewing with the **man** command on a Paragon system.

When a Paragon system is reset, the **bootpp** command reads the *BADNODES.TXT*, *MAGIC.MASTER*, and *SYSCONFIG.TXT* files and creates the *bootmagic* file in the directory */usr/paragon/boot* on the diagnostic station.

CAUTION

Do not use any bootmagic variable that are not listed in the following sections. Other variables may cause unpredictable system behavior.

Minimum Bootmagic List

Below is the minimum bootmagic requirements for booting a Paragon system.

To boot you will need a *SYSCONFIG.TXT* file in the */usr/paragon/boot* directory. This can be generated automatically if *DEVCONF.TXT* is correct by issuing this command:

```
> reset autocfg
```

To boot with ramdisk you need the following bootmagics in your *MAGIC.md file* in the */usr/paragon/boot* directory:

REAL_FIRST_NODE=[n]	[n] = the bootnode. This is usually c * 4 - 1, where c is the number of cabinets.
BOOT_ROOT_DEV=md0a	Without this, the install command is not found.

You will also need a *MAGIC.MASTER* file with at this symbol:

BOOT_FIRST_NODE=[n]	[n] = the bootnode. This is usually c * 4 - 1, where c is the number of cabinets.

To boot without ramdisk the file *MAGIC.MASTER* must exist, even if you specify a different magic file through the **-f** command line switch or by redefining the environment variable *MAGIC_MASTER*.

The *bootmagic* file must contain:

BOOT_FIRST_NODE=[n]	[n] = the bootnode. This is usually c * 4 - 1, where c is the number of cabinets.

Frequently Used Bootmagic Variables

These bootmagic variables are generated automatically by the boot process (if not already defined in *MAGIC.MASTER* or *$MAGIC_MASTER*):

BOOT_FIRST_NODE	Defines the boot node.
BOOT_CONSOLE	Console device selection. Determines the interface to scan. Default: *cm.*
BOOT_STARTUP_NAME	Pathname of server executable to launch on the nodes. Default: */mach_servers/startup.*
BOOT_COMPUTE_STARTUP_NAME	Pathname to server binary to be loaded on the compute nodes. Default: */mach_servers/startup.compute.*
BOOT_EMULATOR_NAME	Pathname of emulator binary to start on the nodes. Default: */mach_servers/emulator.*
BOOT_KERNEL_NAME	Pathname to kernel binary that is broadcast to the non-boot nodes. Default: */mach_servers/mach_kernela* or: */mach_servers/mach_kernelb.*
BOOT_REMOTE_KERNEL	Diagnostic station pathname to the kernel executable loaded to the bootnode. Default: <*address*>*:/usr/paragon/boot/mach_kernel[ab]* where <*address*> is the internet address of the diagnostic station.
BOOT_COMPUTE_KERNEL_NAME	Paragon system pathname to the kernel image loaded on the compute nodes. Default: */mach_servers/mach_kernel[ab]*

BOOT_ALT_KERNEL_NAME	Paragon system pathname to the alternate compute node kernel. Default: */mach_servers/sunmos*.
BOOT_ALT_NODE_LIST	The nodes to be booted with kernel specified in BOOT_ALT_KERNEL_NAME. Default: *<null>*
BOOT_ROOT_DEV	Device name in */dev* where the root file system is located. Default: *rz0a*
BOOT_HOWTO	Hex number used for turning on debugging features. Reserved for internal use. Default: *0x0*
BOOT_ARCH	Not used. Default: *paragon*
BOOT_MY_NODE	Bootmesh sets this value for each node. The value is the node number to which it is being sent. It cannot be changed in any magic files.
ROOT_FS_NODE	The node that provides the root filesystem services. Default: *<bootnode>*.
ROOT_DEVICE_NODE	The node to which the root filesystem device is attached. Default: *<bootnode>*.
NETSERVER	Node which provides network services. Default: *<bootnode>*.
ALLOCATOR_NODE	The node on which the allocator runs. Default: *<bootnode>*.
BOOT_NUM_NODES	Number of nodes in the system. Determined by boot process, from the *SYSCONFIG.TXT* file.
BOOT_MESH_X	Number of columns in the mesh. Determined from the *SYSCONFIG.TXT* file.
BOOT_MESH_Y=*n*	Number of rows in the mesh. Determined from the *SYSCONFIG.TXT* file.

BOOT_CAB_ROWS	The number of cabinets high the mesh is. Possible values are 1 or 2. This is almost never 2 (only when the mesh has 32 rows).
BOOT_IO_NODE_LIST	List of non-compute nodes. These nodes provide disk, ether, or other I/O services, or they are service nodes.
BOOT_COMPUTE_NODE_LIST	This is a list of the compute nodes. These nodes are booted using the executable specified in BOOT_COMPUTE_KERNEL_NAME.
BOOT_NODE_LIST	The complete list of nodes to boot. Determined from SYSCONFIG.TXT
EXPORT_PAGING	This is a list of nodes that provide paging services. Default: *<bootnode>*.
PAGER_NODE	This list shows which nodes are assigned to what pager nodes. Default: *<all nodes page to the bootnode>*
BOOT_TIME	Wall clock time of day according to the diagnostic station. This is used to set the Paragon system time of day clock at boot. Determined by the boot process.
BOOT_DISK_NODE_LIST	List of nodes that have disks. Determined from the *SYSCONFIG.TXT* file.
BOOT_ENET_NODE_LIST	List of nodes that have physical ethernet connections. Determined from the *SYSCONFIG.TXT* file.

Other Useful Bootmagic Variables

RB_MULTIUSER	Determines whether or not to continue booting through single-user mode or create a shell in single-user mode before booting to multiuser. 0 = create a single-user-mode shell. 1 = boot directly to multiuser mode. Default: 0

TCP_SPACE_SIZE	Determines the size of TCP socket buffer size. Reasonable values include "65536" for systems with just ethernet connections, "131072" for systems with HIPPI/FIDDI connections, and "262144" for systems with HIPPI.
BOOT_PAGE0_ACCESS	Determines whether an attempt to access an address in virtual page 0 causes a bus error. 0 = No access allowed. 1 = Access allowed. Default value: 0
BOOT_LOAD_SYMBOLS	Determines whether or not emulator and server symbols are loaded at boot time. 0 = Symbols are not loaded. 1 = Symbols are loaded. Default: 1
BOOT_RED_LED	Intel recommends that machines use the default value for this symbol. This describes the behavior of the red LEDs. The reasonable values are strings from the set "**DcsiglXRme**". Default value: **Dcew** D - Intended for kernel debugging (set *BOOT_RED_LED=D* and call *led_red_[on, off]_debug()* so that debug events can be observed on the LEDs without interference. c - If set, green on at start of *i860_init()* and green off at end of *machine_startup()*. s - Used for scan driver debugging. i - ON during Interrupt service, off otherwise. g - ON if KDB is doing input. Redundant uses in the various console drivers should be deleted. l - Called only by *roll_led_out()*, which is only called by the mesh "multicomputer" console driver. X - Called by *mcmsg_nic_check()* (called by *master_cpu* every clock tick) if NIC *transmit-in-progress* flag is set. R - Called by *mcmsg_nic_check()* (called by *master_cpu* every clock tick) if NIC *receive-in-progress* flag is set.

m - Various uses in the MCP code.

e - ON upon ECC Correctable Errors. Also can be used for other handled error conditions in conjunction with additional bootmagic.

BOOT_GREEN_LED

Intel recommends that machines use the default value for this symbol. This describes the behavior of the green leds. The reasonable values are strings from the set "Dckhibwa".
Default value: Dciw

D - Intended for kernel debugging (set *BOOT_GREEN_LED=D* and *call led_green_[on, off]_debug()* so that debug events can be observed on the LEDs without interference.

c - GREEN ON at start of *i860_init()* and GREEN OFF at end of *machine_startup()*.

k - GREEN shows KDB input focus. (OFF on entry into KDB, ON if cpu acquires the kdb lock, and OFF on exit from KDB.)

h - (GREEN_HIT)
i - (GREEN_INTERVAL)
b - (GREEN_BAR)

w - If none of the *pretty_lights()* options are enabled this will cause a "heart-beat" on the middle GREEN.

a - 1 GREEN LED for each CPU ON if the CPU is not in the IDLE loop upon clock tick, OFF otherwise. Thus it is ON for both user and kernel activity.

DISABLE_BOOTMESH

0 Boot normally.
1 Don't allow bootmesh.
Default: 0

The NORMA_RDMA_GROUP_* symbols determine the number of various Norma transport resources. These values represent the maximum number of incomplete RDMA transmissions that can exist before blocking. Each controls the quantity of RDMA handles in one of the handle groups. These values may need to be increased from their defaults when the system has a large number of nodes, and a large number of PFS stripes.

NORMA_RDMA_GROUP_XMIT_ALLOC
Transmit group
Default: 48

NORMA_RDMA_GROUP_RECV_ALLOC
Receive group
Default: 8

NORMA_RDMA_GROUP_EMMI_PRIV_ALLOC
External memory management group.
Default: 8

NORMA_RDMA_GROUP_PAGEOUT_ALLOC
Pageout group
Default: 9

NORMA_RDMA_GROUP_XMIT_PRIV_ALLOC
Privileged transmit group
Default: 8

NORMA_RDMA_GROUP_PAGEIN_ALLOC
Pagein group
Default: 8

NORMA_RDMA_GROUP_VNODE_PAGER_RX_ALLOC
Vnode pager receive group
Default: 8

NORMA_RDMA_GROUP_VNODE_PAGER_TX_ALLOC
Vnode pager transmit group
Default: 8

Bug Lists for System Software 2

Introduction

This chapter contains a list of open bugs and a list of fixed bugs. These lists are updated just before shipment and are also available online in the files */usr/share/release_notes/ss_buglist* and */usr/share/release_notes/ss_fixed* on the Paragon system.

The open bug list shows the open bugs since Release 1.3 of the Paragon system software. The open bug list is organized in alphabetical order by subsystem name. The bug list includes the following:

- Bug number
- Subsystem name
- Bug synopsis
- Bug description

The fixed bug list lists the bugs fixed since Release 1.3 of the Paragon system software and included in Release 1.4. The open bug list is organized in alphabetical order by subsystem name. The bug listing includes the following:

- Bug number
- Subsystem name
- Bug synopsis

These bug lists were generated on 29 March 96.

Open Bug List

The following lists the open bugs for Release 1.4 of the Paragon system software:

```
CR ID  Release Notes
-------------------------------------------------------------------------
007954 BOOT PROCESS

    Synopsis: When booting, problems detected with fsck are sometimes
    handled improperly.

    If fsck encounters unrepairable file system problems during the
    boot process, the system is supposed to leave you in single-user
    mode so you can repair the file system before you allow users on
    the system.  If the file system is in bad shape (i.e., its
    superblock is corrupted), the boot process may continue to
    multiuser mode (with the errors detected by fsck scrolling off the
    screen) instead of going to single-user mode.

    Workaround: Bring the system up in single-user mode and run fsck
    manually. If fsck is not able to clean up the file system, you may
    need to rebuild the partition using newfs.

009139 BOOT PROCESS

    Synopsis: Double colon incorrectly placed in PAGER_NODE list for
    auto paging tree.

    When attempting to generate an automatic paging tree using the -P1
    flag for bootpp in the "reset" script, an error may occur on
    reset.  This error may be the result of a double colon (::) that is
    incorrectly generated in the PAGER_NODE line of bootmagic.

    Workaround: Enter the PAGER_NODE line in MAGIC.MASTER manually and
    avoid using "-P1" to generate an automatic paging tree.

011895 BOOT PROCESS

    Synopsis: Fscan causes "node not responding" messages.

    Sometimes when fscan says a node is not responding, it is actually
    bogged down and quickly recovers. No reboot is necessary.

    Workaround: To eliminate incorrect messages like "Node not
    responding", you can turn off notify, as follows:

      FSCAN> set notify off
```

012765 BOOT PROCESS

Synopsis: fsck -y in /sbin/bcheckrc can destroy data.

In the course of bringing up a customer system after a disk error, fsck -y produced the following message:

```
root inode unallocated                allocate ?
```

Because the -y switch was specified, fsck did allocate the root inode, thus wiping out data on the file system.

Workaround: The problem is probably due to an extremely rare condition in the customer's file system. However, if this possibility bothers you, you may want to consider replacing fsck -y in /sbin/bcheckrc with fsck -p.

011588 DEVICE DRIVERS

Synopsis: SCSI driver does not handle disconnect phase properly with STK4280 tape.

When another device hangs in the same SCSI bus with the STK4280, the driver does not handle the data properly after reconnecting with the device.

030794 FSDB

Synopsis: fsdb doesn't work with large partitions.

006495 HIPPI

Synopsis: No way to report ULA on HIPPI controller from user level.

There is currently no easy way to determine the ULA of a HIPPI controller. Using the "arp -a" command fails with the error message "arp: could not allocate 0 arptab entries."

Workaround: You can determine the ULA by looking on the controller card for its ULA label. Assuming that the system was booted with fscan, you can type ~# from the console and then enter the node number of the HIPPI board.

008133 IPD

Synopsis: Beyond 64 nodes event tracing is too slow.

Collecting an event trace for an nx application running on more than 64 nodes causes abnormal pm behavior. This behavior is characterized by high perturbation by the performance monitor. Moreover, the amount of time required to output the event data maybe misinterpreted as an application hang.

009100 IPD

Synopsis: system() call not supported by IPD.

Other calls that use system()(such as getenv()) are also not supported. The error message is *** ERROR: Executable not found: unable to access file /home/<user>/sh.

013389 IPD

Synopsis: IPD hangs in single step.

IPD sometimes hangs when single stepping. When this happened, one node appeared to be in a waiting state and never got the signal to step.

030015 IPD

Synopsis: IPD/XIPD cannot display dynamically allocated FORTRAN variables.

Without additional compiler support, the only way to view the contents of array A is to display a pointer to the array, which gives you address and then to display that address, which gives you a dump of the memory contents. If you want the values translated, you then need to use one of the -real, -double, ... switches on display when dumping the memory.

030038 IPD

Synopsis: IPD fails at watch and break points within multi-threaded programs.

When attempting to continue from a multi-threaded portion of an -Mconcur compiled application, IPD fails to break at the action point. When the application finally stops, the action point is gone.

030203 IPD

Synopsis: Option for compiling -Mconcur at opt level 0 is needed to debug code under IPD.

Low levels of optimization are disabled by the -Mconcur switch in the compiler, which creates code that is difficult to debug using IPD or XIPD.

030541 IPD

Synopsis: Cannot display contents of a C++ reference variable.

030672 IPD

Synopsis: IPD exits from app prematurely & hangs on quit when nx app makes nx_load() calls.

030827 IPD

Synopsis:Very poor performance when executing instrumented MPI application under IPD.

Attempts to execute an MPI application that has been instrumented with -prof or -gprof under IPD appear to hang, but are just very slow.

031209 IPD

Synopsis: IPD internal err & loss of contect if context's comm is freed by freed intercommunicator.

If the current context is set to an intracommunicators that has already been freed, and an attempt is made to step or continue through a portion of the application that frees that same intracommunicator, IPD returns an internal error message and loses its context.

012998 IPI-3

Synopsis: panic: getblk: size too big when IPI-3 PFS filesystem is full.

With a system configured using the half_a_gen4 disklabel and all 6 IPI-3 Unix partitions used as PFS stripe directories, an attempt was made to fill the PFS filesystem using the lsize command, which caused the HIPPI node to panic. This panic is reproducible every time when all six IPI-3 Unix partitions are used as PFS stripe directories.

005029 LIBNX

Synopsis: The led() system call has no effect on the LED displays.

011651 LIBNX

Synopsis: exec in compute partition causes memory corruption.

Performing an execl() in the compute partition causes memory corruption on subsequent programs. The memory corruption does not occur with the standard Unix fork and exec nor if only a fork is performed in the compute partition. The failure appears to only occur when an exec is performed in the compute partition and occurs for programs linked with either -nx or -lnx.

The system must be rebooted to use the nodes.

010928 LOAD LEVELING

Synopsis: Load leveler takes too large (14%+) performance hit.

003000 MESH UTILS

Synopsis: nx_nfork() and nx_loadve() are not returning -1 when they should.

nx_fork() is not returning -1 in some circumstances, such as:

* When the num_nodes parameter is greater than the size of the partition.

* When the node_list contains invalid node numbers (such as -1 or node numbers greater than the size of the partition).

* When the node_list contains duplicate node numbers. (In this case nx_nfork() succeeds, but you get a "setptype: Ptype already in use or No active process" error from one of the duplicate children.)

* When the process type is invalid (such as -1).(In this case nx_nfork() succeeds, but you get a "setptype: Invalid ptype" error from the child.)

In other circumstances, nx_nfork() returns -1 as expected, but the errno is wrong. For example, when the num_nodes parameter is invalid (for example, -2 or 0), you get a "nx_nfork(): Not enough space" error instead of the expected EPBADNODE error.

nx_load() and nx_loadve() also sometimes fail to correctly indicate an error condition. For example, calling nx_load() with the pathname of a nonexistent file does not return -1.

Also, whenever nx_loadve() is given a text file to load (such as /etc/motd):

* Patch R1.2.5.1 and earlier: nx_loadve() returns 1 (success).

* Patch R1.2.6/7: nx_loadve() hangs until the user hits CTRL-C. (no error is reported)

010011 MESH UTILS

Synopsis: nx applications that exit(!0) return 0 exit code.

The exit code returned by a parallel application is the exit code set by the proxy process. If you compile with -nx, then you will get an exit code of 1 if any of the internal calls fail (e.g., nx_waitall(), nx_loadve()), otherwise you get an exit status of 0.

Workaround: Compile via -lnx so that you can handle the exit code as you desire.

010230 MESH UTILS

Synopsis: nx_node_attr() does not indicate that MCP is on (used by SPV)

The nx_node_attr() call returns an array of strings that describes each node.If the node's message coprocessor is being used then the token "MCP" is supposed to appear in the node string. Currently, nx_node_attr() does not return the "MCP" token when the MCP is on. This information is used by SPV to determine if performance data are to be reported on the MCP.

012724 MESH UTILS

Synopsis: NQS does not enforce per_req cpu_lim correctly for multi-app jobs

NQS does not correctly enforce the per_req cpu_lim. If an NQS job spawns multiple applications, then the CPU limit on the queue is multiplied by the number of jobs that have been spawned. For example, if you have a 4-node queue with a 60-second per_req cpu_lim, if the NQS job simultaneously starts up two different 2-node jobs, the jobs will both run for 120 seconds, thereby

defeating the per_req cpu_lim. A workaround is to set wallclock_limit=1. Refer to the NQS manual for more information.

004886 MESSAGE PASSING

Synopsis: msgcancel doesn't cancel a message id.

Calling msgcancel() does not currently release the specified message ID. After a call to msgcancel(), the number of available asynchronous message IDs is not increased.

006168 MISCELLANEOUS

Synopsis: BADNODES.TXT is not updated with failed nodes nor accessed by bootpp.

When a node is detected as bad, and has been the cause of 'n' successive reboots, the watchdog should add this node to a file called BADNODES.TXT, and "bootpp" should read this file and remove any node from the list of valid nodes. This would prevent the watchdog from booting a machine over and over again. However, this file is currently not created by the watchdog and "bootpp" does not use it.

012695 NFS

Synopsis: NFS disk access through second ethernet locks up.

This has only been seen when the user's home directories are on a Sun file server that is on the subnet that contains the Paragon(TM) system's second ethernet. Most of the Sun directories are NFS mounted onto the Paragon system through this second ethernet, but a few are mounted through the HIPPI/FDDI node. Occasionally, the system goes into a mode where any attempt to access the NFS mounted directories (an ls, for example) that are mounted through the second ethernet causes a hang and lock. Even ^C will not work. Access to the directories that come through the HIPPI/FDDI are fine, however. This may not be a Paragon(TM) system problem, but a local network problem. Unfortunately, it makes the Paragon system unusable.

030237 NQS

Synopsis: Job moved from pipe queue to batch queue sticks in ROUTING state.

This fault happens about once or twice a month at some sites. After using the qmgr move command to move jobs from a user's batch queue to the pipe queue, a job remains in the ROUTING state and cannot be deleted. This hanging job blocks all new incoming requests, because these new jobs were waiting to be piped to their final batch queues.

The problem is to get NQS working again without doing a complete new installation and losing all the NQS jobs. One possible solution that may work at your site is to delete an internal control file:

```
rm -f /var/spool/nqs/private/root/control/+++++0
```

After a new start NQS should work, but the hung job will be lost.

030497 NQS

Synopsis: NQS can fail to make partitions if partition with EPL=5 exists in NQS space

When the compute partition is gang-scheduled, NQS should be able to create partitions even if there is a non-NQS partition in the node set dedicated to NQS. However, if the allocator is configured with MAX_OVERLAP_PER_PRI=1 and there is a partition with EPL 5 in NQS node set, NQS will fail to create a partition that overlaps it (even though that partition is supposed to have EPL 3 or 1). This failure causes NQS to shut itself down.

007563 OSF COMMANDS

Synopsis: Transitions between run levels are not clean.

Using the init command to go from multiuser mode to single-user mode and back again to multiuser mode does not work cleanly.

Workaround: To go back to multiuser mode reliably, you will need to reboot the system.

008778 OSF COMMANDS

Synopsis: fsck hangs during reset if reset occurred while devices were being formatted.

If the system is reset while one or more RAID devices are being formatted, fsck will hang during the reboot process.

Workaround: Go into single-user mode and edit /etc/fstab to remove references to the devices that were being formatted when the system was reset.Then, reboot the system and reformat the devices. After reformatting the devices, you can partition the devices and add entries to /etc/fstab again.

009607 OSF COMMANDS

Synopsis: rwhod does not work.

The rwhod command does not work properly. It sends a packet to other hosts only once on startup, instead of every 30 seconds. Also, the initial packet usually contains the wrong load average. The command also does not receive packets from other hosts properly.

There is no workaround for this problem.

010995 OSF COMMANDS

Synopsis: ps shows incorrect information about server processes on nonlocal nodes.

ps shows all server processes as having essentially the same attributes as the server process on the local node. For all but the local node, ps gives wrong process information.

011678 OSF COMMANDS

Synopsis: newfs after format should fail, but doesn't.

After a low level format, disklabel says it is damaged (as expected), but newfs does not complain. In addition, for every sized partition, the superblock numbers displayed are the same. This is incorrect as they are different sized partitions. An fsck on any of the partitions says everything is clean and a previously mounted filesystem is displayed. This is definitely wrong.

012158 OSF COMMANDS

Synopsis: cron stops intermittently.

013371 OSF COMMANDS

Synopsis: sed s/xxx/yyy/p doesn't work correctly

The p (print if changed) option of the sed s command does not work if there is a later d (delete) command affecting the line. It does work if the -n option to suppress output is enabled.

030010 OSF COMMANDS

Synopsis: showfs and df may be overflowing, causing a negative number to be reported.

030142 OSF COMMANDS

Synopsis: /sbin/init S command behavior unexpected.

/sbin/init S brings the Paragon system up to multiuser and then down to single user.

006409 OSF LIBS

Synopsis: National Language Support locale categories LC_COLLATE & LC_CTYPE no longer work.

If you use setlocale() to change the value of LC_COLLATE or LC_CTYPE to a language other than the default, the calls tolower(), toupper(), strcoll() and strxfrm() return incorrect values.

008561 OSF LIBS

Synopsis: Floating-point accuracy of printf() and other functions is not IEEE compliant.

The floating point accuracy of atof(), _dsto2fp(), fcvt(), and ecvt() are not up to IEEE standards when converting double-precision numbers. These functions in turn affect printf() and scanf(). For example, when converting double-precision numbers using printf() or scanf(), the last couple of digits (least-significant) may not be correct.

There is no workaround for this problem.

012165 OSF LIBS

Synopsis: floor and ceil don't work right on denorm and NAN.

007552 OSF MICROKERNEL

Synopsis: Long parallel SAT runs temporarily stall.

010816 OSF MICROKERNEL

Synopsis: Scripts hang with rsh and qsub in infinite loop.

010999 OSF MICROKERNEL

Synopsis: With MCP on, LEDs are 100% busy when blocked at crecv().

The CPU is in a spinning loop during crecv() when the MCP is on. The green LED will then indicate that the CPU is 100% usage.

012272 OSF MICROKERNEL

Synopsis: Encountered assert_wait during tape operation.

030213 OSF MICROKERNEL

Synopsis: a node ran out of free memory and server threads stuck with vm_page_wait, vm_fault_pag

030462 OSF MICROKERNEL

Synopsis: Exception at mach_msg(478000,2,0,2000,6300,0,0)+0x180

030570 OSF MICROKERNEL

Synopsis: kernel panic at dipc_receive_port()+0x58

The kernel paniced at dipc_receive_port()+0x58, but for some reason, the autoddb script couldn't collect all the useful information in the first shot. It therefore continued out from the debugger trying to examine the node some more. The node however hit another panic (this time in the server at rpvpsop_get_vproc_port()).

007934 OSF SERVERS

Synopsis: hostname or settime commands hang system during single-user mode.

In single-user mode, the hostname and settime commands hang. These commands only work after bootmesh has run.

008720 OSF SERVERS

Synopsis: acct is not usable with Paragon systems.

The standard Unix accounting functions under /usr/lib/acct do not function well on Paragon systems. When using these functions, some of the data that is collected is incomplete, due to incompatibility with the multinode environment.

There is no workaround for this problem.

009323 OSF SERVERS

Synopsis: Shutdown commands don't always bring down the system cleanly.

The commands "shutdown," "init 0," "fastboot," "halt," "fasthalt," "reboot," and other shutdown commands do not always shut the system down cleanly.

009741 OSF SERVERS

Synopsis: "ls -l" of a circular symbolic link may panic the system.

This bug occurs when there are circular symbolic links between files on the boot node disk and a remote I/O node disk (non-boot node) or when there are circular symbolic links between files on two or more remote I/O node disks. Performing an "ls -l" in this situation will cause the server to generate thousands of threads, which will eventually panic the system.

Workaround: Do not create circular symbolic links between files on remote I/O nodes. This practice is considered an improper operation on the system.

011743 OSF SERVERS

Synopsis: Code executing kill(0,9) in IPD will wire down 4 pages of memory.

Running the following code under IPD, kills the TAM and wires down four pages of memory. Running the same executable without IPD leaves the wired page count unchanged.

```
#include <nx.h>
#include <stdio.h>
 main()
 {
 if(mynode() == 0) {
 printf("Now killing application...\n");
 kill(0,9);
 } else {
 sleep(10);
 }
 }
```

012261 OSF SERVERS

Synopsis: *part commands hang after rmpart -f tried to kill hung application.

010318 OSF SYSTEM CALLS

Synopsis: plock(2) with data segment greater than physical memory hangs machine.

If you call plock(2) to lock your data segment (DATLOCK) and your data segment is greater than the physical memory on the node will hang the Paragon system. The plock() call uses the vm_wire() mach call which cannot bail out in trouble. It will wire up pages until it is wedged.

010898 OSF SYSTEM CALLS

Synopsis: Processes reading stdin in iomode M_UNIX don't have their own file pointers.

A test program reads stdin and redirects it to a file. Each node should read from the beginning of the file, but they read in sequence. The test program allows for using the default iomode (M_UNIX), or setting the iomode through a command argument. In either case, the nodes share a file pointer rather than having their own. That is to say, the nodes read from the file in turn, and they read sequentially through the file instead of each process reading from the start of the file. In at least one case, just calling setiomode caused an emulator error.

011414 PARAIDE

Synopsis: Having multiple SPV sessions from Paraide freezes all windows.

011848 PMAKE

Synopsis: pmake does not return MACS accounting msgs from nx_initve().

007468 PTHREADS

Synopsis: pthread_setcancel sometimes fails.

When general cancellation of pthreads is blocked by pthread_setcancel(), pthreads are sometimes cancelled anyway, when functions that set up "cancellation points" are used. The

pthread_cond_wait(), pthread_join(), and pthread_cond_timedwait() functions (which are "cancellation points") incorrectly carry out cancellation requests even though general cancellation of threads is blocked.

Workaround: Add additional synchronization to insure that no cancels are pending prior to the call of pthread_cond_wait(), pthread_join(), and pthread_cond_timedwait() and that cancels do not occur during the wait.

008136 PTHREADS

Synopsis: SIGCONT does not always restart all pthreads.

The SIGCONT signal does not always restart the pthreads of a process that have been suspended with the SIGSTOP signal.

There is no workaround for this problem.

008180 PTHREADS

Synopsis: Pthread internal error occurs during a file I/O operation.

You cannot cancel a pthread during a file I/O operation such as cread(). Doing so will result in a "pthread internal error in thread_suspend" message and the thread will be left in an indeterminant state.

009086 PTHREADS

Synopsis: sigwait() causes hang on an MP system.

The sigwait() function should return the expected asynchronous signal number when the signal is received.Instead, when run on an MP system, the function causes a hang. CTRL-C will relieve the hang.

There is no workaround for this problem.

011554 PTHREADS

Synopsis: Can only create 17 pthreads on GP system with 64 M bytes stacksize.

A test program is unable to create 21 pthreads. The test fails on pthread_create Error: Not enough space. The test, hello, is a unix process running on a service node. The test passes on MP systems and fails on GP system.

011378 SAT

Synopsis: Output report not updated if sat exits on first error or on interrupt.

If you run sat -x -o <report> the report will not be updated prior to exit. The report is currently updated after each iteration of the test list but should always be updated prior to exit unless system resources are unavailable. This problem also exists for exiting because of an interrupting signal other than the timer alarm (-m option).

030174 SCSI DRIVER

Synopsis: The Panic on unrecoverable SCSI errors doesn't allow retries before panicking

030194 SCSI-16 H/W

Synopsis: SCSI-16 returns the wrong data on read from PFS

A SCSI-16 with either RAIDs or disks attached to both channels and both channels configured into PFS will periodically return the wrong data on a read operation.

012886 SPV

Synopsis: SPV daemon mysteriously dies on HIPPI node requiring reboot to clear.

With the SPV daemon running and HIPPI raw transfers occurring, the SPV daemon (apparently) on the HIPPI node dies causing SPV to display its warning message.

030284 SPV

Synopsis: launching SPV sometimes leads to core dump

012684 SUNMOS

Synopsis: Sunmos doesn't access MDC memory.

004708 UFS

Synopsis: Mount of two partitions on same directory permitted.

It is currently possible to mount more than one disk partition on the same directory. The second partition mounted "masks" the first (that is, the contents of the mount point appear to be the contents of the second partition mounted). Once this has occurred, attempting to unmount one of these partitions may actually unmount the other.

012124 XIPD

Synopsis: XIPD can't display class methods of same-name classes defined in functions.

XIPD does not have enough information to specify a method in order to list the method's source code in the following situation:

In a single file, file.C, two routines, RoutineOne() and RoutineTwo(), declare and define a class call LocalClass.This class is local only to the routine that it's defined in.

LocalClass has a method called Print() in both classes.

Trying to display either of the Print() methods for LocalClass, IPD emits the following error:

```
*** ERROR: search failed
***                    ambiguous name: file.C{}LocalClass::Print(void)
```

Since there's more than one LocalClass::Print(void) in file.C{}. XIPD actually needs to have some way of getting the start and end line number of the routine.

There is no workaround.

030990 XIPD

Synopsis: XIPD core dumps loading coredir if faulted app wasn't run on all partition nodes.

031311 XIPD

Synopsis: XIPD viewpoint & execution problems is MPI application is loaded on ptype > 0.

031414 XIPD

Synposis: XIPD core dumps when reloading an MPI application.

031285 XIPD

Synopsis: XIPD internal error & core dump using intercommunicator context within core file.

If an attempt is made to show an intercommunicator from a loaded MPI-type core file, an internal error occurs and a core dump is generated.

Fixed Bug List

The following lists the bugs fixed since Release 1.3 of the Paragon system software:

CR ID Release Notes

013660 DOC

Synopsis: crecvx() does not behave as advertised in manual pages

030461 DOC

Synopsis: No documentation for ntpd

The only documentation available on ntpd is in the description of postboot in the Installation Guide.

012941 IPD

Synopsis: Interrupted IPD skips reading symbol table for subsequent loads.

If IPD's program loading is interrupted during its symbol table reading, then IPD will skip reading the symbol table for subsequent program loads, thinking that the symbol table already loaded successfully. This causes IPD to fail or segfault later on.

006313 LIBTGI

If you select ALL nodes with XIPD, select an executable to load, select APPLY, and then press OK, XIPD should load the nodes, once. Instead, it attempts to perform a load once for each time the APPLY button was pressed and then once more for pressing the OK button. XIPD does not try to perform any of these loads until the OK button

is pressed. The net result of this operation is that XIPD tries to load the same program on the same set of nodes with the same ptype. Load error messages are then generated, stating that the ptype is already in use. There is no workaround for this problem.

010024 LIBTGI

In the Load application dialog, selecting 'Load into all nodes of the partition' after selecting some nodes via 'Load into specific node(s) or process type' causes an error. Apparently, selected nodes are not automatically deselected when 'Load into specific node(s) or process type' is deselected.

There are three known workarounds:

1. Before leaving 'Load into all nodes of the partition', deselect any nodes that you may have selected. Then, you can select 'Load into all nodes of the partition' and there will be no conflict.

2. Instead of leaving 'Load into specific node(s) or process type' (and then selecting 'Load into all nodes of the partition'), explicitly select all nodes from within 'Load into specific node(s) or process type'.

3. Cancel the load application dialog and restart it, selecting 'Load into all nodes of the partition' first.

006959 MACS

Synopsis: MACS does not track underused node time in batch jobs.

009752 MACS

Synopsis: Multiple applications in a batch job may not be charged correctly.

This problem occurs when a batch job is run with qsub and the script for the batch job runs applications for accounts other than the account in effect when qsub is invoked. In this situation, MACS views all the applications run from the batch job as running under the account in effect when the qsub command is issued and bills the run time for all the applications to that account. From the point of view of acctrep reports, time is not lost, only mis-assigned.

008284 MESH UTILS

Synopsis: Higher priority does NOT roll out lower priority job in active layer.

Jobs are assigned to layers based on whether the job will fit in the layer.Priority is not a consideration. Here is an example of the problem in a 16 node partition:

```
Job #1: priority 10 size 8
Job #2: priority 5 size 8
Job #3: priority 10 size 8
```

Jobs #1 and #2 are issued first, followed by job #3. Since job #3 is high priority than job #2, job #3 should preempt job #2, but it is not allowed to. Job #3 must wait until job #1 or #2 finishes.

Workaround: Manually schedule jobs in the order you wish them worked on.

008432 MESH UTILS

Synopsis: Using nx_pri() to lower job priority does NOT roll job out for other ovlp jobs.

When two jobs of different priorities are run in an overlapping partition and nx_pri() is used to lower the priority of the higher priority job, the job is not rolled out when its priority becomes lower than the other job.

003144 MESSAGE PASSING

A test that sends two asynchronous messages hangs when run via "node -sz 2 -on 0 \; node -on 1" but runs fine via "node -sz 2". These two methods of running the test should be equivalent.

003231 MESSAGE PASSING

If a process changes its ptype via setptype(), the process can still probe for messages sent to the old ptype, and cannot probe for messages sent to the new ptype. The process should only be able to probe and receive messages for its current ptype.

006082 MESSAGE PASSING

If a node sends messages to another node which does not receive them, the sending node will eventually block because the system buffer space on the receiving node is exhausted (this is expected behavior). However, the number of bytes sent before blocking is expected to decrease when the application is run on a larger number of nodes, but this is not the case--the number of bytes sent before blocking is always the same.

003021 NFS

If root creates a file in an NFS-mounted filesystem, then the owner and group are very strange For example, 42949672944294967294.

009184 NQS

Synopsis: NQS dies when wrong path for a pipe queue client is used.

If you create a pipe queue using a wrong pathname to the client, the following three things happen:

1. The NQS qmgr does not detect the problem.

2. If a job is submitted to the (not installed) pipe queue, NQS dies without the user being informed.

3. You must set up NQS again from scratch to get a working configuration again.

006803 OSF COMMANDS

The command "echo 'string' >> file" sometimes fails with the error "file: cannot create" if the specified file is in an NFS file system.

009632 OSF COMMANDS

Synopsis: ls -l on directory with extremely large files can hang session.

In directories with extremely large files (for example, a file that fills the entire partition), the command ls -l sometimes hangs. You can kill the process to unhang the session, but processes are left upon exiting that do not go away until the system is rebooted.

002796 OSF DOC

The dd manual page indicates that fskip=number, conv=iblock, v=oblock and conv=notrunc are valid options. All four fail.

002800 OSF DOC

The du manual page lists unsupported -k and -x options.

002804 OSF DOC

The -i option for the env command is not supported.

002807 OSF DOC

The -f option for the at command is not supported.

002808 OSF DOC

The "-q queuename" option for atq is not supported.

002908 OSF DOC

In the printed manual page, the -o option that is listed in the Flags section for the at command, is not found in the Synopsis section.

002966 OSF DOC

The macros are missing. Current text incorrectly indicates they follow immediately,

002668 OSF MICROKERNEL

Repeated malloc()s for large amounts of memory can exhaust the system's default paging space (used for virtual memory), causing the system to crash.

004715 OSF SERVERS

Under some circumstances, repeatedly using the "grep" command to search a large (1.5M-byte) file will eventually hang (after 10 to 40 repetitions). You can log in from another window and kill the process, but the process is left in an "<exiting>" state. The system will eventually crash after that.

008398 OSF SERVERS

Synopsis: System crashes when FORTRAN open() is called from hundreds of nodes.

Opening the same file from hundreds of nodes in a FORTRAN program using an open() call can cause a performance bottleneck, which can result in a system crash.

008732 OSF SERVERS

Synopsis: Server does not print all critical warnings to console.

Some O/S errors are only printed to the local node, not to the console.

009993 OSF SERVERS

Synopsis: Dump dies with SIGBUS when reading from raw devices.

012229 OSF SERVERS

Synopsis: Code causes EMULATOR EXCEPTION Signal=10, Code=14 under IPD -sz 1.

The following FORTRAN code fails on gopen when run on the debugger using only a single node:

```
      program main
      include 'fnx.h'

       call gsync()
       call gopen(1,'/pfs/test_file',M_RECORD)
       call gsync()
       write(6,*)'File open on node ',mynode()
       call gsync()
       close(1)

       end
```

012593 OSF SERVERS

Synopsis: A user code can trigger a deadlock in get_data_token().

A user parallel application that performs a large number of open(), close(), fstat(), chmod(), and write() calls to the same file can cause a deadlock situation in the get_data_token () routine of the server.

Basically, some thread wants the token and asks the owner to release it. But the owner does not think it has the token and replies with a "fsvr_token_not_found". This request/reply cycle seems to go on forever once it gets started.

012534 RAID UTILITIES

Synopsis: Deleting or adding LUN forces low level format.

Using the add LUN or Create LUN in ACE forces a low level format on 4G-byte drives when the drive geometry values for spare sectors per track do not match the RAID controller default values.

Allocator Changes A

Introduction

This appendix describes two major changes that were made to the way the allocator works in Release 1.4. These changes were instituted to prevent the following occurrences:

1. Under some circumstances, a lower priority application ran before a higher priority application.

2. Under other circumstances, a very high priority application was unable to preempt lower priority applications and could not execute at all.

Overview of the Allocator

Parallel Applications and Priority

A parallel application is a program running on one or more compute nodes. These compute nodes belong to the *.compute* partition or to some subpartition, created under *.compute*.

A parallel application has a priority associated with it. The priority is an integer, with larger values indicating higher priority.

A partition also has priority. A partition's priority is the lower of:

- The partition's *effective priority limit* (set when the partition was created).

- The priority of the highest-priority application or subpartition in the partition.

Gang Scheduling and Scheduling Layers

When parallel applications have the same priority, they may be *gang-scheduled*. Gang scheduling means that a parallel application becomes active for a certain length of time (called the *roll-in quantum*) before it relinquishes its compute nodes to another parallel application. When a parallel application becomes active it has control over all the compute nodes on which it runs.

Parallel applications that use the same compute nodes are said to overlap. When parallel applications do not overlap, they can be run at the same time. Parallel applications that do not overlap belong to the same *scheduling layer*. Gang scheduling consists of rolling scheduling layers in and out.

The allocator fills up a scheduling layer in the most efficient way it can with *active objects*. Active objects are applications or partitions that have applications running in them.

Gang Scheduling and Scheduling Layers: Example

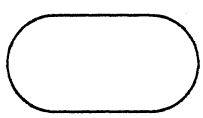
Consider the example shown in Figure A-1. Assume that all applications have the same priority. Figure A-1 shows how one scheduling layer is filled up, causing another one to be constructed. In the example, four applications are loaded into a partition. The first three applications (A1, A2, and A3) use up all the compute nodes in the partition. The fourth application (A4) must share compute nodes with applications A1 and A2. So the allocator must construct two scheduling layers.

Figure A-2 illustrates the scheduling for the layers built up as in Figure A-1. As expected, the *.compute* partition has two scheduling layers, L1 and L2. L1 rolls in, and stays in for a certain length of time, called L1's roll-in quantum. Then, L2 rolls in and stays in for a certain length of time called L2's roll-in quantum. The process repeats as needed. This is called gang-scheduling.

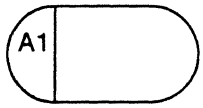
When L1 is rolled in, applications A1, A2, and A3 run. These applications run on different compute nodes and do not interfere with each other. When L2 is rolled in, A4 runs. Note, however, that when A4 runs the compute nodes assigned to A3 are not used. So when L2 is rolled in, A3 migrates to L2 and runs also. An application can migrate to a different scheduling layer, but it cannot migrate to different compute nodes. The result is that A1 and A2 swap with A4, and that A3 runs all the time.

In summary, the execution time profile looks as follows:

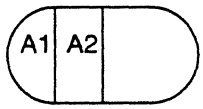
L1: A1, A2, A3
L2: A4, A3
L1: A1, A2, A3
L2: A4, A3
•
•
•



Represents a group of compute nodes organized as a partition.

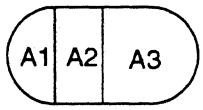


Load application 1 called A1.

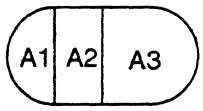


Load application 2 called A2. A1 runs on different compute nodes than A2 does, and so both applications can run at the same time.

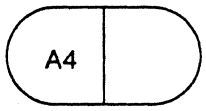
These applications (A1, A2, A3, and A4) are assumed to have the same priority.



Load application 3 called A3. A1 and A2 run on different compute nodes than A3 does, and so all three applications can run at the same time.



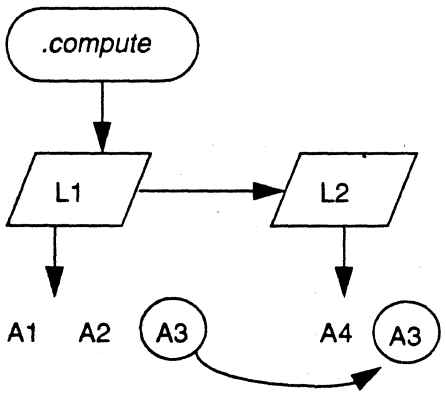
Scheduling Layer 1



Scheduling Layer 2

Load application 4 called A4. There are no available compute nodes in the partition. A4 must share compute nodes with other applications (in this example, A1 and A2). A new scheduling layer is created to contain A4.

Figure A-1. Loading Applications and Making a Scheduling Layer



In this example, A1, A2, A3, and A4 all have the same priority. The *.compute* partition has two scheduling layers, L1 and L2. First the applications in one layer run; then the applications in the other run.

Assume that each layer has a roll-in quantum of one minute. Then, L1 rolls in, and applications A1, A2, and A3 run. They run on different compute nodes and do not interfere with each other.

When the minute is over, L2 rolls in and A4 runs for one minute. The allocator notices that A3 can fit into L2; so A3 migrates to L2, and A3 and A4 run together.

The result is that A1 and A2 swap with A4 and that A3 runs all the time.

Figure A-2. How Applications Run in Two Scheduling Layers

Scheduling with Priority

So far the discussion has assumed that all the applications have the same priority. Now, define the priority of a scheduling layer as the priority of its highest-priority active object. So far the only active objects discussed have been applications, but partitions can be active objects too.

Scheduling with Priority: Example

Consider Figure A-3. It is like Figure A-1, except that the applications have different priorities. Both A1 and A2 have priority 3; A3 has priority 5; and A4 has priority 4. Then, L1 has priority 5, and L2 has priority 4. The result is that A1, A2, and A3 run to completion (because the scheduling layer to

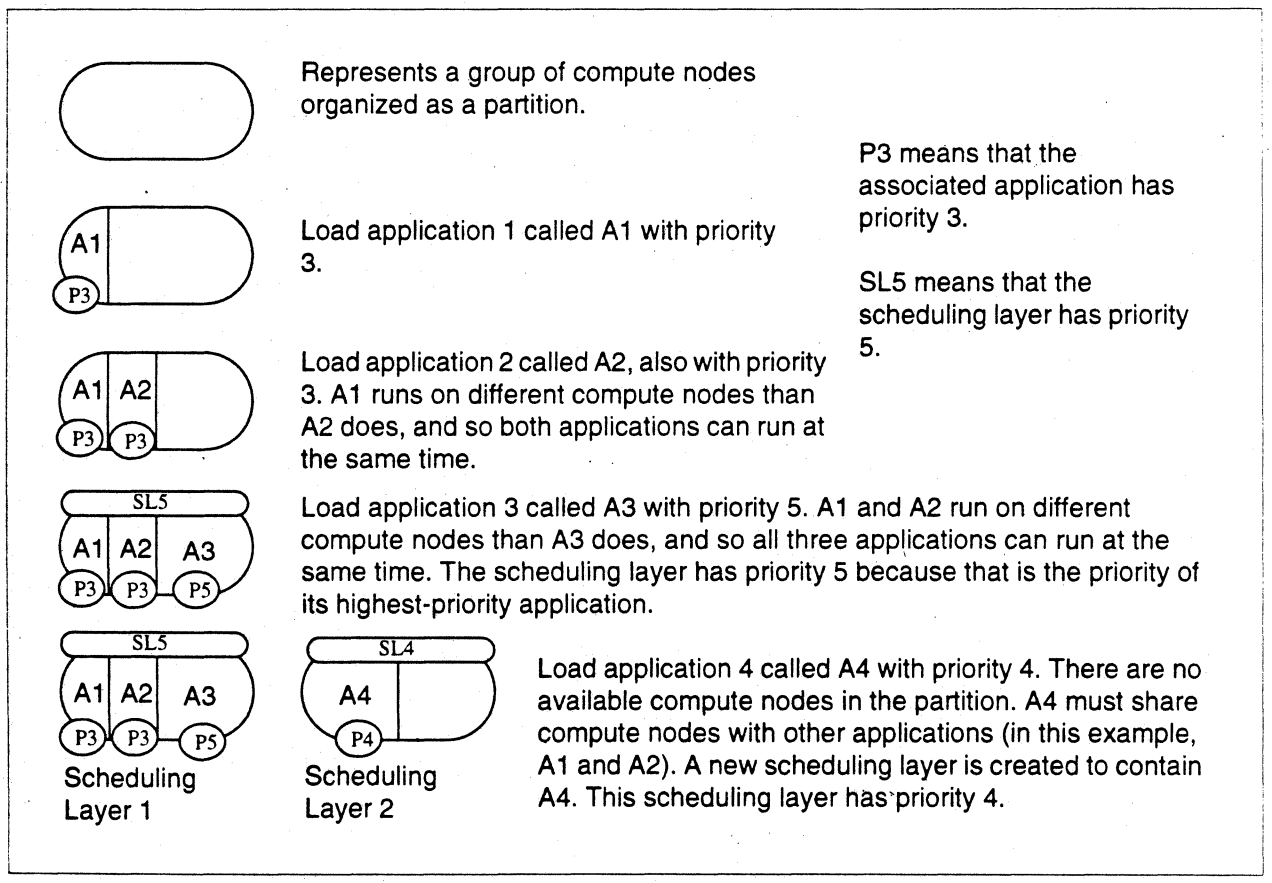
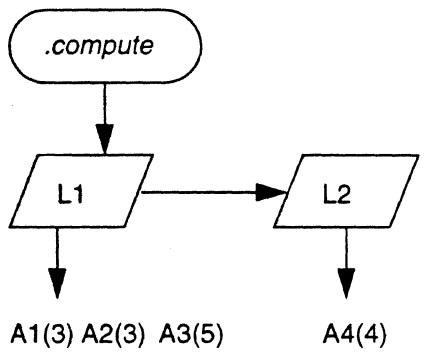


Figure A-3. Loading Applications and Making a Scheduling Layer (with priority)

which they belong has the highest priority) before A4 even executes. Figure A-4 illustrates the scheduling for the layers built up in Figure A-3.



The *.compute* partition has two scheduling layers, L1 and L2. L1 contains applications A1, A2, and A3. The number in parentheses after the application number indicates its priority.

L1 has a higher priority than L2. Hence, applications in L1 run to completion before L2 runs as all. This is true even though an application in L2 has a higher priority than an application in L1.

Figure A-4. How Applications Run in Two Scheduling Layers (with priority)

The example now illustrates a problem that is addressed by subsequent allocator changes. Note that A4 must gang schedule with A1 and A2, even though A4 has higher priority that A1 and A2. Many people do not expect that behavior.

When A1 and A2 complete, A4 will move into scheduling layer 1, and no gang scheduling will occur. If A3 completes before A1 and A2, scheduling layer 1 changes its priority to 3. The result is that scheduling layer 2 has the higher priority and A4 runs while A1 and A2 are suspended,

Consider the following variation on the example. If A4 now has priority 5, gang scheduling still occurs because the two layers have the same priority. But now A4 (with priority 5) shares compute nodes with A1 and A2 (both with priority 3). Again, this is not what many people would expect.

Partitions as Active Objects

This section presents a similar but slightly more complicated example than that already shown. The purpose is to show how scheduling layers can contain partitions as active objects and how scheduling layers exist in a hierarchical structure.

Partitions as Active Objects: Example

Figure A-5 shows a scheduling flow diagram for the applications loaded and the partitions created in the following sequence. Ignore the portion of the figure within the dotted lines for now.

1. The application A1 is loaded onto a subset of compute nodes in the *.compute* partition.

2. Another application A2 is loaded onto the same subset of nodes as A1.

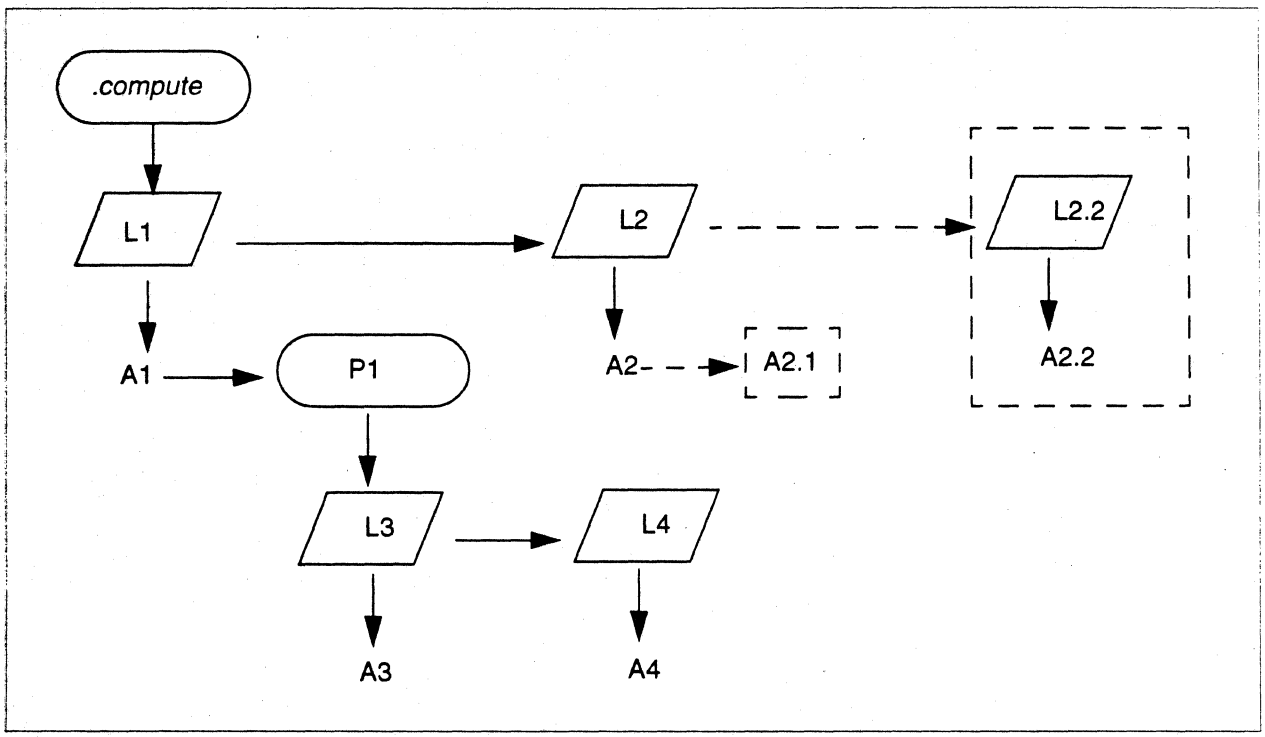


Figure A-5. Active Partitions as Part of Scheduling Layers

3. Applications A3 and A4 are loaded into P1. P1 is a subpartition of *.compute* and does not contain compute nodes that are assigned to A1. It may have been created previously, or you may create just before loading A3 and A4. The point is that at this time (step 3) the partition is made active. An active partition is one that has running applications

Assume that all partitions and applications have the same priority. Assume that all partitions have the same roll-in quantum, one minute. The scheduling flow is as follows:

Minute 1	A1 and A3 run. They run on different compute nodes and do not interfere with each other. A4 does not get to run, because when the minute is up, it is L2's time.
Minute 2	A2 runs because the roll-in quantum for L1 is up, and it is L2's time.
Minute 3	A1 and A4 run. By the time A3 is ready, it is again L2's time.
Minute 4	A2 runs.
Minute 5	A1 and A3 run.
Minute 6	A2 runs.
Minute 7	A1 and A2 run.

•
•
•

Some More Examples of Scheduling Layers

This section presents two more examples that further illustrate the use of scheduling layers.

The 2.1 example. Let us say that after loading A3 and A4 into the partition P1, you load another application A2.1.

A2.1 is loaded into *.compute*. It does not share compute nodes with A2, but it does share compute nodes with P1. Hence, the application A2.1 becomes part of scheduling layer L2. This new application is shown in a dotted box in Figure A-5. Now when A2 runs (minutes 2, 4, 6, etc.), A2.1 runs also.

Consider a variation of this example. What if you loaded A2.1 after loading A2 but before making P1 active? If A2.1 did not share compute nodes with A1, A2.1 would become part of L1. Now after loading A2.1, make P1 active. Because P1 shares compute nodes with A2.1, P1 becomes part of L2.

The 2.2 example. Do not load A2.1. Instead load application A2.2, which shares compute nodes with both A1 and A2. A new scheduling layer L3 is created, and A2.2 belongs to L2.2. This example illustrates a *degree of overlap* of 3 because it has three scheduling layers (L1, L2, and L2.2) at the same level.

You can set the maximum degree of overlap with an allocator configuration variable called *DEGREE_OF_OVERLAP*. If this variable were 2, you would be unable to load A2.2 and create L2.2. The system stays up and running, but your request is rejected.

If A2.2 only overlapped one of A1 or A2, it would join the scheduling layer that contained the application it did not overlap. The degree of overlap would remain at 2.

Allocation Layers

Allocation layers are distinct from scheduling layers. Allocation layers keep track of which compute nodes are assigned to which partitions. When you create a partition, the **mkpart** command consults the allocation layers to determine if it can honor your request.

An allocation layer does not contain any overlapping partitions. If partitions overlap, they must belong to different layers.

Allocation Layers: Example

Figure A-6 shows three allocation layers. They are created as follows:

1. Create partition P1.
2. Create partition P2, which does not overlap P1. Then, P1 and P2 belong to the same allocation layer.
3. Create partition P3. It overlaps with either P1 or P2. Hence, it belongs to a different allocation layer, L2.
4. Create partition P4. It overlaps with either P1 or P2, but not P3. Hence, it belongs to the same allocation layer as P3.
5. Create partition P5 as a subpartition of P2.

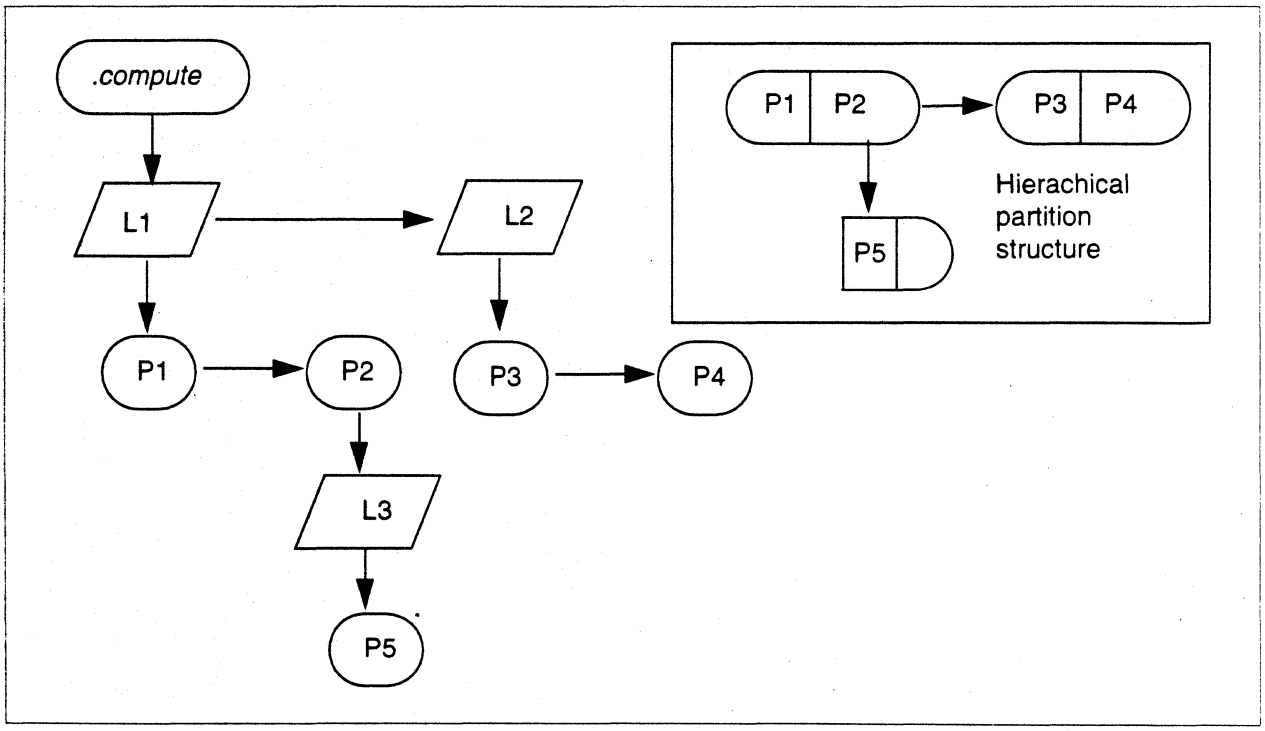


Figure A-6. Allocation Layers

Degree of Overlap

The degree of overlap has meaning for both scheduling layers and allocation layers. The same allocator configuration variable *DEGREE-OF_OVERLAP* determines the maximum overlap for both allocation and scheduling layers.

Degree of Overlap: Example

Figure A-5 (with the dotted lines) shows scheduling layers with a degree of overlap of 3. This means that there are three scheduling layers at the same level—L1, L2, and L2.2. Figure A-5 also shows a sublevel with a degree of overlap of 2. There are two scheduling layers that overlap in a sublevel. These are L3 and L4.

Figure A-6 shows allocation layers with a degree of overlap equal to 2. This means that there are two allocation layers at the same level. These are L1 and L2. If *DEGREE-OF_OVERLAP* were set to 1, this would not be allowed. You would get an error message when you tried to create P3 or P4.

Although, in Figure A-6, P5 shares compute nodes with P2, but this is *not* an example of overlap. P5 is a subpartition of P2. P5 belongs to the allocation layer L3. When the *DEGREE-OF_OVERLAP* is 1, you can still have more than one allocation layer, but they cannot be at the same level. In Figure A-6, L1 and L2 are at the same level; L3 is on a sublevel.

Space Sharing

In a *space-shared* configuration, gang scheduling does not occur. That is, neither scheduling layers nor allocation layers overlap. If Figure A-6 represented a spaced-shared configuration, L2 and L4 could not exist.

Restricting the Amount of Gang Scheduling

The allocation configuration variable *NUM_GANG_PARTS* determines the maximum number of gang-scheduled partitions in a system.

When you create a partition, you specify whether it is space-shared or gang-scheduled. For example, if *NUM_GANG_PARTS* is 1, you can have only one gang-scheduled partition.

Figure A-7 shows a partition (the *.compute* partition) that is gang scheduled. Figure A-8 shows a partition P1 (a subpartition of *.compute*) that is gang-scheduled. In Figure A-8 the *.compute* partition is *not* gang-scheduled. Both Figure A-7 and Figure A-8 require *NUM_GANG_PARTS* equal to 1..

If in Figure A-8, the *.compute* partition were gang-scheduled (which is usually the case), then the gang scheduling in Figure A-8 would not be allowed.

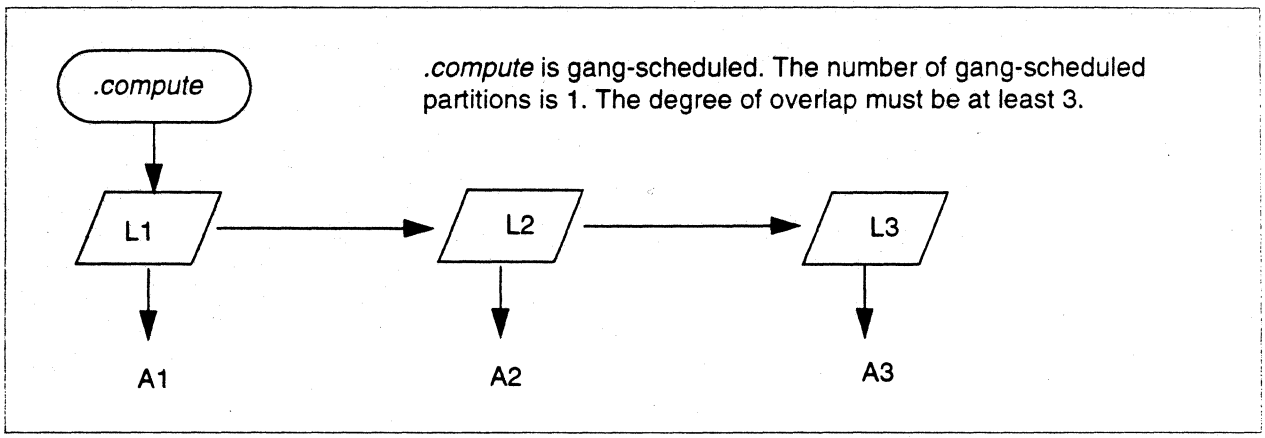


Figure A-7. A Partition (*.compute*) with Gang Scheduling

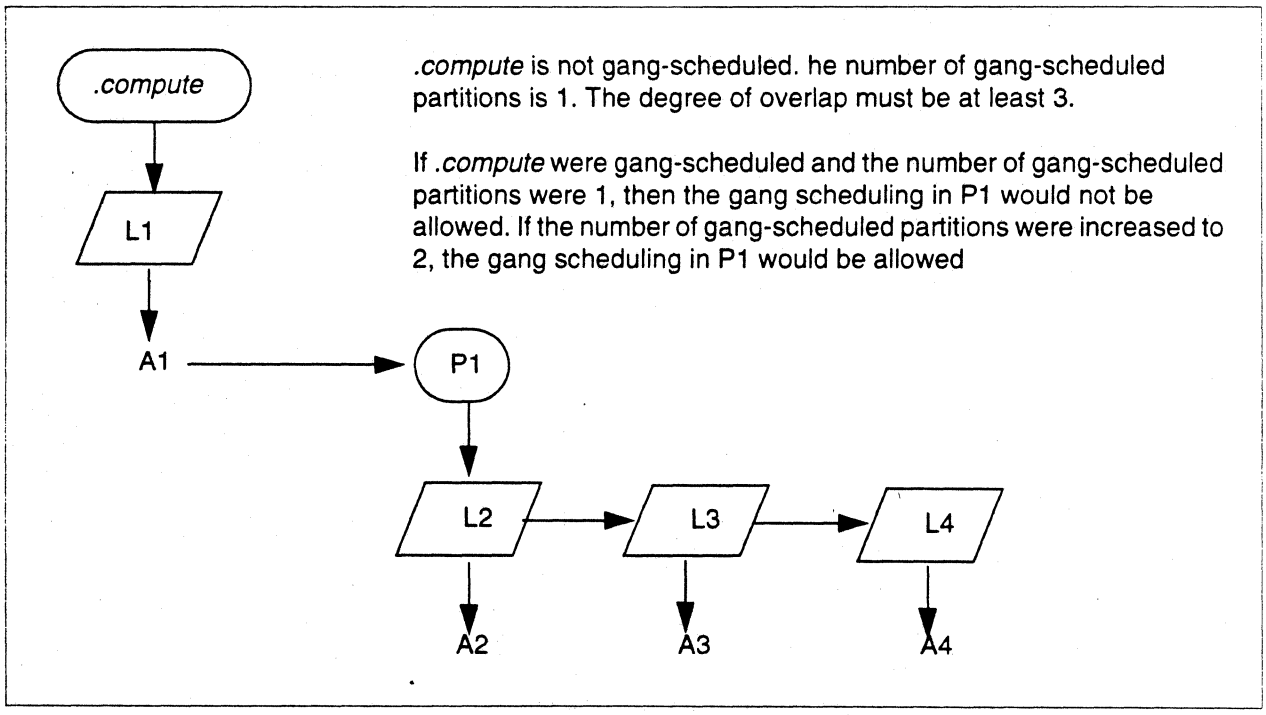


Figure A-8. A Subpartition with Gang Scheduling

Allocator Change 1: Priorities Work

Job scheduling now works the way most users expect. That is, when applications overlap compute nodes, the highest priority application always takes precedence.

The allocator will move an application into a different scheduling layer if by doing so, the higher priority application takes precedence over a lower priority application. However, the allocator still cannot change the compute nodes to which the application is assigned.

Allocator Change 1: Priorities Work: Example

Figure A-9 is a modification of Figure A-1 that shows how higher priority applications take precedence over lower priority applications.

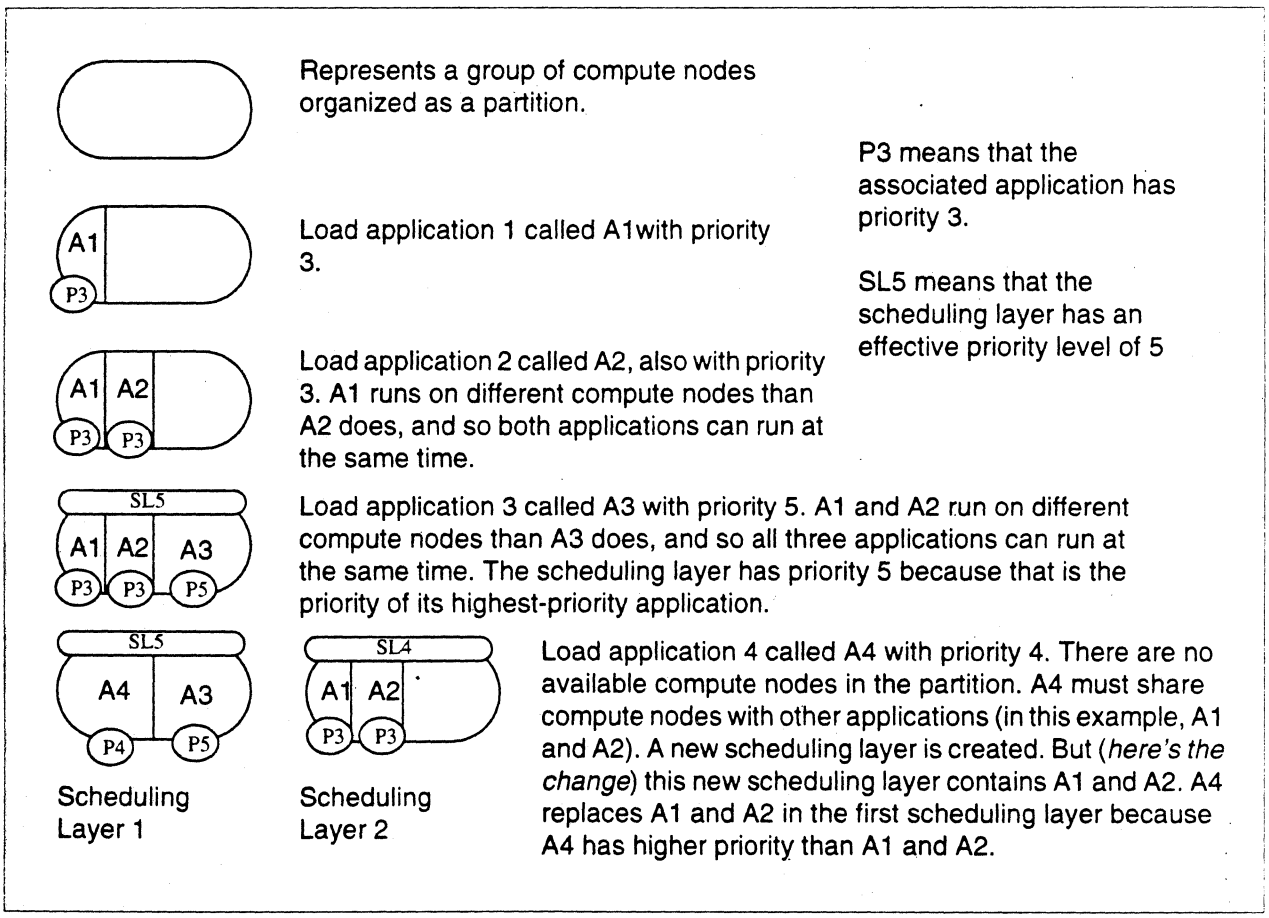


Figure A-9. Loading Applications and Making a Scheduling Layer

The important point to note is that A4 goes into the higher priority scheduling layer. A4 and A3 (the two highest priority applications) run to completion before A1 and A2 begin execution. If, however, A4 did not fit into the space occupied by A1 and A3, A4 would join L2; and the situation would be as it was before the change.

A Standard Configuration

Consider a particular standard configuration. The system runs NQS and an interactive utility. NQS is the Network Queueing System; the interactive utility creates a partition, runs a specified job in that partition, and then removes the partition. At least one site calls this interactive utility **pexec**. Call this standard configuration the NQS/interactive configuration.

Assume that during the day (referred to as prime time), NQS has priority 1 and the interactive utility has priority 2. This means that interactive jobs have priority. If the space is available for an NQS job, it runs.

However, if a subsequent interactive job then requires the compute nodes used by the NQS job, it takes them. The interactive job preempts the NQS job. This can happen because of the first allocation change already described. When this interactive job completes, the NQS job resumes.

The situation is reversed during the night (referred to as non-prime time). NQS has priority 2, and the interactive utility has priority 1.

A Standard Configuration: Example

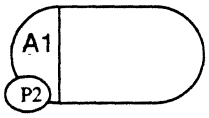
Figure A-10 shows how the allocator with Change 1 operates in the standard configuration just described. Figure A-11 shows how it would operate if the change had not been made.

Note that both figures require a degree of overlap of 2.

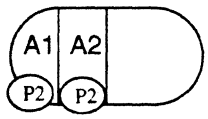
This figure illustrates how the allocator operates **with** the change in one of the standard configurations.

Represents a group of compute nodes organized as a partition.

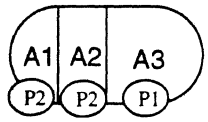
P2 means that the associated application has priority 2.



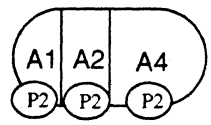
The interactive utility creates partition and loads application 1 called A1with priority 2.



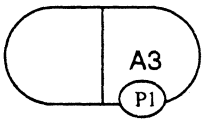
The interactive utility creates partition and loads application 2 called A2with priority 2. A1 runs on different compute nodes than A2 does, and so both applications can run at the same time.



NQS loads application A3 with priority 1. The compute nodes are available; so A3 runs.



Scheduling Layer 1, L1



Scheduling Layer 2, L2

Now the interactive utility loads A4. Because this is prime-time and The interactive jobs have higher priority than NQS jobs, A4 preempts A3. A3 is suspended.

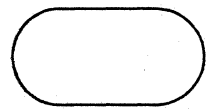
When A4 finishes, A3 is loaded back into L1.

Note the efficient compute node utilization.

DEGREE_OF_OVERLAP must be at least 2, because L1 and L2 overlap.

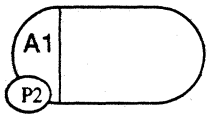
Figure A-10. Standard NQS/Interactve Configuration with Allocator Change.

This figure illustrates how the allocator operates **without** the change, assuming one of the standard configurations.

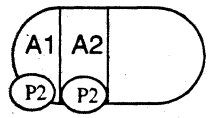


Represents a group of compute nodes organized as a partition.

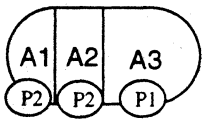
P2 means that the associated application has priority 2.



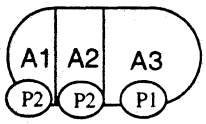
The interactive utility creates partition and loads application 1 called A1with priority 2.

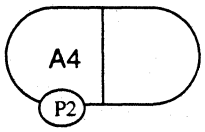


The interactive utility creates partition and loads application 2 called A2with priority 2. A1 runs on different compute nodes than A2 does, and so both applications can run at the same time.



NQS loads application A3 with priority 1. The compute nodes are available; so A3 runs.





Now the interactive utility loads A4. But L1 is filled up A4 goes into L2.

Gang scheduling results. A4 swaps with A1 and A2; A3 runs all the time. Result is a low priority application gets precedence over a high priority application, not a desired situation.

Scheduling Layer 1

Scheduling Layer 2

DEGREE_OF_OVERLAP must be at least 2, because L1 and L2 overlap.

Figure A-11. Standard NQS/Interactive Configuration without Allocator Change.

Allocator Change 2: Overlap per Priority

The second allocator change involves the definition of a new allocator configuration variable called *DEGREE_OF_OVERLAP_PER_PRIORITY*.

The purpose is to allow a high priority job to preempt other jobs, when the configuration like the NQS/interactive configuration is in place.

Assume the NQS/interactive configuration described earlier. During prime time, interactive jobs are running; NQS jobs are waiting. If compute nodes become available for an NQS job, that NQS job runs, but it is suspended as soon as an interactive job requires those compute nodes. During non-prime time, the priorities are reversed.

Because interactive jobs and NQS jobs have different priorities, an interactive job will not gang schedule with an NQS job.

Also assume that you do not want interactive utility jobs gang scheduling among themselves, or NQS jobs gang scheduling among themselves. Set the *DEGREE_OF_OVERLAP* to 2. One overlap is the interactive job; the other is NQS. If you try to load another job (either interactive or NQS) that overlaps with an existing interactive or NQS job, it is rejected because it exceeds the overlap limit.

Now assume that you now have a very high priority job that you want to run. By "very high priority" is meant an application whose priority is higher than both the interactive job and the NQS job. For example, in the NQS/interactive configuration, the priority might be 3. It is a very important job and hence is sometimes referred to as an *admiral* job.

If you load the admiral job and it uses compute nodes assigned to both an interactive job and an NQS job, the allocator attempts to create another scheduling layer. But it cannot because the *DEGREE_OF_OVERLAP* is 2.

You can load the admiral job, if you set the *DEGREE_OF_OVERLAP* to 3. But that causes other problems. With a *DEGREE_OF_OVERLAP* of 3, you could load another interactive job in place of the admiral job, resulting in two interactive jobs that undergo gang scheduling.

The solution is to define a new allocation configuration called *DEGREE_OF_OVERLAP_PER_PRIORITY* and set it to 1. Now overlap among jobs of equal priority is forbidden. This means that interactive jobs cannot gang schedule among themselves and that NQS jobs cannot schedule among themselves. But if the *DEGREE_OF_OVERLAP* is 3, an admiral job can still preempt an interactive job and an NQS job.

If you increase *DEGREE_OF_OVERLAP* to 4, you allow yet another job (of even higher) to preempt the admiral job, the interactive job, and the NQS job.

Allocator Change 2: Overlap per Priority: Example

Assume the situation at the end of Figure A-10. Figure A-12 shows how this example extends that in Figure A-10.

Suppose someone loads a very high priority job A5. Because the standard priorities in this configuration are 1 and 2, the value 3 can represent very high priority.

If A5 fits in the space used by A3 in L1, it will do so. A3 will move to L2.

If A5 requires more computes nodes, the allocator will try to make another scheduling layer. But it cannot do so because this would require *DEGREE_OF_OVERLAP* to be 3.

So set *DEGREE_OF_OVERLAP* to 3. Also define a configuration variable called *DEGREE_OF_OVERLAP_PER_PRIORITY* as 1. Then when A5 (with priority 3) is loaded, it preempts all the other applications and runs to completion.

If you now load an A6 (also with priority 3), the load fails for two reasons. 1) *DEGREE_OF_OVERLAP_PER_PRIORITY* is 1, so you cannot overlap A5 and A6. 2) *DEGREE_OF_OVERLAP* is 3, so you can have only three scheduling layers.

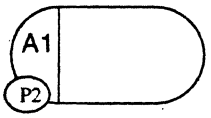
If you set *DEGREE_OF_OVERLAP* to some larger value (say 4), loading A6 would fail due to only the first reason. Most configurations (like the NQS/interactive utility configuration) do not allow overlap among applications of equal priority. This restriction prevents gang scheduling.

However, with *DEGREE_OF_OVERLAP* equal to 4, if A6 had an even higher priority (say 4), it would successfully preempt all other applications (including A5) and run.

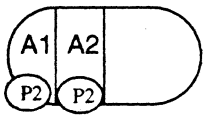
This figure illustrates how the allocator operates **with** the change in one of the standard configurations.

Represents a group of compute nodes organized as a partition.

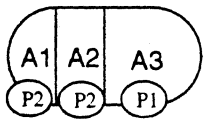
P2 means that the associated application has priority 2.



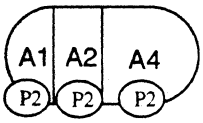
The interactive utility creates partition and loads application 1 called A1with priority 2.



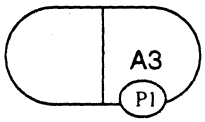
The interactive utility creates partition and loads application 2 called A2with priority 2. A1 runs on different compute nodes than A2 does, and so both applications can run at the same time.



NQS loads application A3 with priority 1. The compute nodes are available; so A3 runs.



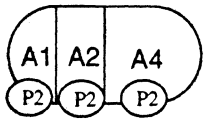
Scheduling Layer 1, L1



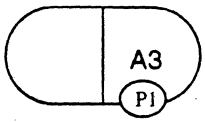
Scheduling Layer 2, L2

Now The interactive utility loads A4. Because this is prime-time and interactive jobs have higher priority than NQS jobs, A4 preempts A3. A3 is suspended.

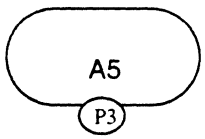
When A4 finishes, A3 is loaded back into L1.



Scheduling Layer 1, L1



Scheduling Layer 2, L2



Scheduling Layer 3, L3

Now someone loads A5, which takes up all compute nodes and is the highest priority job. The allocator creates another scheduling layer, L3 and runs A5.

This example requires that *DEGREE_OF_OVERLAP* be at least 3, because L1, L2, and L3 overlap. Note that, even if *DEGREE_OF_OVERLAP* were 4 and you tried to load an A6 with priority 3, you would be unable to because you would violate the *DEGREE_OF_OVERLAP_PER_PRIORITY*.

Figure A-12. How a Very High Priority Application Gets to Run