INTEL CORPORATION
15201 N.W. Greenbrier Parkway
Beaverton, OR 97006
(503) 629-7600

June 18, 1991

Dear iPSC® System Update Customer:

This package contains your Release 3.3 system software. With this software installed on your iPSC® Supercomputer[1], your system is ready for use. Please read through the documentation and distribute it to anyone intending to use the system.

The R3.3 system software runs on iPSC®/2 (with Intel386™-microprocessor-based CX nodes) and iPSC®/860 systems (with i860™-microprocessor-based RX nodes or with a combination of CX and RX nodes). It includes a broad selection of software tools such as the Fortran and C programming language compilers, the Interactive Parallel Debugger, Network Queueing System and the Performance Analysis Tools. The Network File System, node TCP/IP, and X Window System software packages are offered as options.

Before using your iPSC System:

- **Read this letter completely.**
- **Verify the contents of this package.**
- **Read the *iPSC®/2 and iPSC®/860 System Software Release 3.3 Product Release Notes.***

1. The terms "iPSC Supercomputer" and "iPSC system" refer to any of the following SSD products: iPSC®/2, iPSC®/2S, iPSC®/860, and iPSC®/860S

Package Contents

Your iPSC system software package is shipped in a number of boxes. Please verify that it includes the following items:

Media

iPSC®/2 System Software R3.3
cartridge tape (part number 312059-001)

iPSC®/2/860 Extension Software R3.3
cartridge tape (part number 312060-001)

iPSC®/860 Compiler Libraries System R3.3
cartridge tape (part number 312129-001)

iPSC®/2/860 System S/W R3.3 Update 5/91
cartridge tape (part number 312186-001)

R3.3 Documentation

Refer to the "Documentation Included With iPSC® System Release 3.3 Update" section on page 7 for a complete list of documentation supplied. (Note that you have received only those manuals that have been revised or added since Release 3.2.)

If items are missing, or if you have any questions, contact Intel Supercomputer Systems Division immediately. Refer to "Comments and Assistance" for information about how to contact Intel Supercomputer Systems Division.

What is in This Release?

Release 3.3 contains iPSC system software for both iPSC/2 and iPSC/860 nodes. The iPSC Release 3.3 system software provides the following new features:

- **Improved iPSC®/860 C and Fortran compilers**
  The iPSC/860 C and Fortran compilers have been improved over those delivered with Release 3.2. The new compilers are called **icc** and **if77**, and can be run on Sun-3 and Sun-4 workstations as well as on the SRM.

- **Interactive Parallel Debugger**
  The new Interactive Parallel Debugger (IPD) replaces DECON as the source-level application debugger for the iPSC/2 and iPSC/860 systems. IPD offers conventional debugging facilities, such as breakpoints and the ability to display and modify memory, as well as features for examining the state of the message passing network. Differences from DECON include improved reliability, i860 machine-level debugging, and enhanced iPSC/860 Fortran support.

- **Parallel Performance Analysis Tools**
  Release 3.3 includes an early release of the new Performance Analysis Tools (PAT). These tools help you analyze your iPSC/860 application programs and improve their performance. PAT consists of three utilities: an execution profiler that monitors the time spent in individual routines, a communication trace tool that assesses the time spent in communication, and an event trace tool that shows the interactions between processors and can monitor user-specified events. Performance data collected by the PAT utilities can be displayed graphically on Sun-3 and Sun-4 workstations.

- **Profiler for iPSC®/860 node programs**
  The new **prof860** command lets you profile execution of RX node programs. It is based on the UNIX System V **prof** command and runs on Sun-3 and Sun-4 workstations as well as the SRM.

- **Basic Math Library**
  The new Basic Math Library is an implementation of the BLAS (Basic Linear Algebra Subprograms) levels 1, 2, and 3 for iPSC/860 programs. It includes some one-dimensional Fast Fourier Transform (FFT) operations as well as the vector-vector operations, matrix-matrix operations, and vector-matrix operations of BLAS.

- **Network Queueing System**
  The Network Queueing System (NQS), the *de facto* standard for batch queueing on supercomputers, is now supported on the SRM for iPSC/860 systems. It is completely compatible with the COSMIC version of NQS from NASA, and has been modified so that it can load cube jobs.

- **More memory on RX nodes**
  The system software now supports RX nodes with 32 and 64M bytes of memory on iPSC/860 systems, as well as the existing 8 and 16M byte configurations.

- **Node Shell on service node**
  The Node Shell (**nsh**) now executes on any I/O node without a SCSI interface (such as an ENET node or service node) rather than on the currently attached cube. This lets you invoke a shell on your iPSC/2 or iPSC/860 system without tying up any compute nodes. The new **-s** option to **nsh** makes the shell execute on the current cube, for backwards compatibility.

- **Improved remote host security**
  A new file, */usr/ipsc/lib/ruser*, lists the workstations and users or groups who are allowed to access the iPSC system's remote host facilities. This new feature prevents unauthorized users from using your iPSC system over the network; it is available on both iPSC/2 and iPSC/860 systems.

- **NX information structure**
  NX for the iPSC/860 system stores performance and status information (about message passing, interrupts, CPU usage, and so on) in a special read-only structure called *nxinfo*. This structure has now been made accessible to application programmers, who can use the information in it for performance improvements and checking the status of the nodes.

- **Non-IEEE arithmetic for iPSC®/860 programs**
  This release supports a faster, less accurate non-IEEE mode for divide and square root, and the flushing of denormal data values to zero for RX node programs, in addition to the IEEE operations provided in Release 3.2.

- **Enhanced accounting and security**
  The **getcube** command and *conmser* process have been enhanced to provide "hooks" for customer-developed accounting packages on both the iPSC/2 and iPSC/860 systems. You can use these hooks to keep track of system activity and control system access with a higher level of detail than the standard iPSC system administration tools provide. The **plogon** command and call also record more information than they did in Release 3.2, and have a new **-i** option to report the pathname of the current log file.

- **NX message buffering modification on iPSC®/860 systems**
  The **getcube** command has been enhanced to optionally disable message buffering on RX nodes. This provides a performance boost for some programs which overlap communication with computation. In addition, the default message buffering algorithm for RX nodes has been optimized for better performance.

What Software Options are Available?

- **Network File System (NFS)**
  With NFS installed on your iPSC system, you can access file systems on other Ethernet machines from your SRM and iPSC system as if they were local (usable on both iPSC/2 and iPSC/860 systems).

- **Source Products**
  Source code for the NX/2 operating system and the Concurrent File System™ (CFS) is available. Applies to both iPSC/2 and iPSC/860 systems).

- **CFS tape support**
  This Concurrent File System option supports both 9-track and 8-mm tape drives (usable on both iPSC/2 and iPSC/860 systems).

- **TCP/IP on the nodes**
  An optional TCP/IP connection enables direct access to a node from a local area network (LAN). The connection allows direct Ethernet access to CFS and to RX nodes via a concurrent I/O Ethernet node. Multiple TCP/IP connections can be installed in an iPSC/860 system to provide multiple network drops (usable on the iPSC/860 system only).

- **X Window System**
  The X Window System Version 11 Release 4 Patch level 18 client libraries are provided as an optional component, bundled with TCP/IP on the nodes. Release 3.3 supports the *MIT-supported libraries only* (usable on the iPSC/860 system only).

- **File Transfer Protocol (ftp) capability**
  As part of TCP/IP on the nodes, you can invoke **ftp** on your workstation and transfer files to the Concurrent File System. This capability does not require iPSC/860 nodes (usable on both iPSC/2 and iPSC/860 systems).

- **ProSolver™-SES**
  This is a package of routines used for assembling, factoring and solving large double-precision sparse systems of equations which can originate from any source (usable on the iPSC/860 system only).

- **iPSC®/2 Ada**
  The iPSC/2 Ada development environment offers an integrated family of tools — compiler, linker, make, debugger, and disassembler — tailored to the iPSC/2 system. In addition, the standard Ada runtime system is extended with a library of iPSC/2 communication and system calls (usable on the iPSC/2 system only).

New Documentation

The documentation for the iPSC system has been revised and expanded to cover the new features provided in this software release. Refer to the "Documentation Included With iPSC® System Release 3.3" section on page 7 for a complete list of documentation supplied.

Restrictions and Limitations of Release 3.3

Every effort has been made to assure the quality of this release, but at shipping time we are aware of a few problems. Please refer to the *iPSC®/2 and iPSC®/860 System Software Release 3.3 Product Release Notes* for known restrictions, limitations, and their workarounds.

Installation

To install all or any part of your system, refer to the installation instructions in the *iPSC®/2 and iPSC®/860 System Software Release 3.3 Product Release Notes*. This document has separate sections describing how to install the UNIX operating system, the iPSC system software, and the TCP/IP software. Also refer to the release notes for optional products (such as CIO Ethernet) for their installation instructions.

NOTE

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

Comments and Assistance

Intel Supercomputer Systems Division is eager to hear of your experiences with our latest software product. Please call us if you need assistance, have questions, or otherwise want to comment on your iPSC system.

1-800-421-2823 (Customer Support Hotline)
(44) 793 641 469 (in England)
Your Local Intel Sales Office (in Europe)
support@ssd.intel.com (Internet address)

Supercomputer Systems Division is trying to produce the best documentation for your needs. If you have comments about the iPSC manuals, you can send them electronically to the following address:

techpubs@ssd.intel.com (Internet address)

Sincerely,

Elliot Swan

Manager, Product and Technical Marketing
Intel Supercomputer Systems Division

Documentation Included With Your iPSC® System Release 3.3 Update

Included in this package you will find the following new and revised iPSC system documentation:

iPSC® User's Reference Manual Set

iPSC®/2 and iPSC®/860 Basic Math Library User's Guide (part number 312128-001)	**New**
iPSC®/2 and iPSC®/860 Interactive Parallel Debugger Manual (part number 312043-001)	**New**
iPSC®/2 and iPSC®/860 Programmer's Reference Manual (part number 311708-004)	**Revised**
iPSC®/2 Simulator Manual (part number 311534-003)	
iPSC®/2 and iPSC®/860 Network Queueing System Manual (part number 312061-001)	**New**
iPSC®/860 Parallel Performance Tools Manual (part number 312139-001)	**New**
iPSC®/2 and iPSC®/860 User's Guide (part number 311532-007)	**Revised**

iPSC® Quick Reference Guides

iPSC®/2 and iPSC®/860 C Commands and Routines Quick Reference Guide (part number 3311610-004)	**Revised**
iPSC®/2 and iPSC®/860 Fortran Commands and Routines Quick Reference Guide (part number 311615-004)	**Revised**
iPSC®/2 and iPSC®/860 Interactive Parallel Debugger Commands Quick Reference (part number 312042-001)	**New**

iPSC® System Administrator's Manual Set

iPSC®/2 and iPSC®/860 System Administrator's Guide (part number 311014-006)	**Revised**

Bugs Fixed by Release 3.3

The following bugs have been fixed for Release 3.3.

NX NODE EXECUTIVE

1. **flushmsg() now flushes all messages**
   The **flushmsg()** routine flushes all messages in transit.

2. **If a fork() on a node fails, subsequent forks would also fail until killcube**
   Failed forks no longer persist.

3. **All messages are flushed when a node process kills itself**
   When a node process calls **killcube()** (for example, **killcube(-1,-1)**), pending messages for that node are now flushed.

4. **Invalid buffer pointers**
   Invalid buffer pointers in host programs are more reliability detected.

5. **killcube no longer hangs when the host sends messages to non-existent nodes.**
   When a program hangs because of sending messages to non-existent nodes or because a node is sending messages to another node that is not receiving, the hung program can be killed by **killcube.**

6. **bootcube detection**
   The **bootcube** process is now detected by the cube utility commands such as **getcube, bootcube, cubeinfo,** and **showvol,** and any other host program that calls a **setpid()**. The programs now stop with an error message if a **bootcube** is in progress.

7. **cread() now correctly sets errno.**

8. **killsyslog() now closes files**
   The **killsyslog()** function now closes files properly.

RX NODE MATHEMATICAL OPERATIONS

1. **RX floating point operations**
   RX floating point operations on denormals, infinities and NaN's are now handled properly by the IEEE-754 floating point exception handler.

NODE SHELL

1. **Error message now generated when wrong type of executable is called**
   The node shell (**nsh**) now returns error messages when an executable of the wrong type is executed (for example, an iPSC/2 executable file in an iPSC/860 cube).

2. **nsh wildcard (*) now works properly for file names larger than 14 characters.**

3. **It is no longer possible to load and kill processes on nodes not assigned to you.**

4. **Filling the file system using lsize**
   When you fill the CFS file system using the **lsize** command in the **nsh** shell, you now receive the error message:

   ```
   No space left on device
   ```

5. **iPSC/860 version of nsh handles interrupts correctly**
   iPSC/860 version of **nsh** now handles user-generated interrupt correctly.

6. **nsh can now run on service node**
   If you have a service node or an ENET node, you no longer need to tie up four or more iPSC/860 nodes to run the node shell (**nsh**).

7. running tar() from within the node shell now correctly accesses files from Sun mounted file systems.

CONCURRENT FILE SYSTEM™

1. **Killing an application that is allocating a large amount of space to a file no longer hangs CFS**
   Using **relcube, killcube,** or interrupting **nsh** while an application is allocating a large amount of space to a file no longer hangs CFS.

2. **The load command run on the SRM now loads executables that reside on CFS**

3. **CFS now supports creation of very large files of 2G bytes or more.**

4. **mkdev now gives correct error message**
   If **mkdev** is used to create a special file that already exists, it prints the error message:

   ```
   File exists
   ```

5. **Execution of cbackup and crestore now properly restricted**
   The superuser is now the only one able to use these commands.

6. **Invalid file descriptor argument now displays the correct error message**
   If you use an invalid file descriptor as an argument to **setiomode()**, the following error message is now printed:

   ```
   Bad file number
   ```

7. **When given an invalid file descriptor, restrictvol() now sets errno to the correct value**

8. **The global restrictvol() call now works as documented**

9. **CFS lsize has improved error messages**

10. **Concentrated I/O activity no longer hangs CFS.**

11. **CFS tape support can now read over filemarks**
    CFS tape support now supports reading over filemarks.

12. **CFS returns incorrect bytes read count**
    While reading from a variable block mode tape and requesting block sizes larger than what was recorded on tape, CFS now returns correct byte count.

CONCURRENT FILE SYSTEM™ TAPE

1. **End of medium detection**
   End of medium is now properly detected.

2. **Tape operations no longer hang tape driver**
   Following a **write()**, any tape forward movement that is not a **write()** now returns an error message.

3. **Reading a tape using the wrong block mode no longer hangs the tape driver**

4. **Backing off from the beginning of a tape no longer hangs the driver**

5. **CFS SCSI driver has been improved**
   CFS SCSI driver has been modified to correctly handle the case where it may get a busy status in between read requests.

iPSC®/2 FORTRAN COMPILER

See the *iPSC®/860 Fortran Compiler User's Guide* for information on the iPSC/860 Fortran Compiler.

1. **Fortran etos() and stoe()calls have been corrected**
   The **stoe()** call has been changed to accept space-terminated strings.

REMOTE HOST

1. **load() or exec() processes executing on nodes now find files on remote host**
   load() and **exec()** calls executed on the nodes now look for the files on the remote host workstation.

2. **Improved message passing**
   Remote host message passing has improved reliability.

3. **Loading a large program from a remote host now works**
   Programs (greater than 1 Megabyte) can now be loaded from a remote host.

UNIX SOFTWARE

1. **The file command now recognizes iPSC/860 binaries.**

2. **NX now behaves as UNIX when creating new pages of memory**
   When NX creates new pages of memory, it zero fills them in the same manner as the UNIX operating system.

DOCUMENTATION

1. **esize() does not truncate the file for whence = 0 and whence = 1**
   The *iPSC®/2 and iPSC®/860 Programmer's Reference Manual* is changed to document actual operation.

2. **esize() no longer returns an error when size is greater than the disk space.**
   For offset greater than disk space, **esize()** allocates the available space and returns the new file size. The *iPSC®/2 and iPSC®/860 Programmer's Reference Manual* has been updated.

3. **Load from cfs now works as advertised in the User's Guide**
   The **load** command loads node executables from the cfs as is described in the *iPSC®/2 and iPSC®/860 User's Guide* on page 7-14.

June 1991
Order Number: 312189

iPSC®/2 AND iPSC®/860
RELEASE 3.3 SOFTWARE
PRODUCT RELEASE NOTES

intel® Corporation

REV.	REVISION HISTORY	DATE
-001	Original Issue	06/91

RESTRICTED RIGHTS

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the rights in Technical Data and Computer Software clause at 52.227-7013. Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051.

PREFACE

These release notes provide the latest information on features, limitations, workarounds, and installation for the Release 3.3 system software for the following Intel Supercomputer Systems Division products: iPSC®/2, iPSC®/2S, iPSC®/860, and iPSC®/860S.

NOTE

In the remainder of the manual, we will use the term "iPSC system(s)" to refer to these products.

These notes assume that you are an application programmer, familiar with the C or Fortran language and the UNIX operating system.

For more information, refer to the Release 3.3 system software customer letter that accompanied your software.

For installation instructions, refer to Chapter 2.

ORGANIZATION

Chapter 1	Describes features of the Release 3.3 system software.
Chapter 2	Provides system software installation information.
Chapter 3	Describes known limitations and their workarounds.
Chapter 4	Describes information necessary to convert PRM assembly code to ABI assembly code.
Chapter 5	Provides information on node memory usage.

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.

. . . (Ellipsis dots) 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 DOCUMENTS

For more information, refer to the following manuals:

iPSC® System Manuals

iPSC®/2 and iPSC®/860 C Commands and Routines Quick Reference
311610-004
Summarizes all C routines and commands for the iPSC system.

iPSC®/2 and iPSC®/860 C Language Reference Manual
311567-004
Describes the C compiler for the iPSC system.

iPSC®/2 and iPSC®/860 Hardware Installation Manual
311461-003
Describes installation and powering up of all iPSC system configurations.

iPSC®/2 and iPSC®/860 Interactive Parallel Debugger Commands Quick Reference
312042-001
Summarizes all Interactive Parallel Debugger commands.

iPSC®/2 and iPSC®/860 Interactive Parallel Debugger Manual
312043-001
Tells how to use the iPSC Interactive Parallel Debugger and provides command reference information.

iPSC®/2 and iPSC®/860 Math Libraries Reference Manual
311868-001
Describes the math libraries available on the iPSC system.

iPSC®/2 and iPSC®/860 Network Queueing System Manual
312061-002
Describes and tells how to use the Network Queueing System (NQS) software to queue and manage batch/device processes in the iPSC system.

iPSC®/2 and iPSC®/860 Programmer's Reference Manual
311708-004
Describes iPSC system commands and system calls (both C and Fortran).

iPSC®/2 and iPSC®/860 Site Preparation Guide
312028-001
Tells the customer how to prepare a site for the installation of an iPSC system.

iPSC®/2 and iPSC®/860 System Acceptance Test User's Guide
311870-001
Tells how to use the System Acceptance Test.

iPSC®/2 and iPSC®/860 System Administrator's Guide
311014-006
Describes the system administration tasks related to operating and maintaining an iPSC system.

iPSC®/2 and iPSC®/860 User's Guide
311532-007
Overviews the iPSC system, including hardware and software architectures. Tells how to develop and run programs.

iPSC®/2 Fortran Language Reference Manual
311020-004
Describes the Green Hills Fortran compiler for the iPSC/2 system.

iPSC®/2 Simulator Manual
311534-003
Tells how to use the iPSC/2 Simulator for software development.

iPSC®/860 Basic Math Library User's Guide
312128-001
Describes advanced math library routines for the iPSC system.

iPSC®/860 C Compiler User's Guide
312130-001
Tells how to use the iPSC/860 C compiler driver.

iPSC®/860 Fortran Compiler User's Guide
312131-001
Tells how to use the iPSC/860 Fortran compiler driver.

iPSC®/860 Parallel Performance Analysis Tools Manual
312139-001
Describes and tells how to use the Performance Analysis Tools (PAT) utilities to capture and analyze execution, communication, and event data on the iPSC/860 system.

Intel® Manuals

i860™ 64-Bit Microprocessor Assembler and Linker Reference Manual
240436-003
Tells how to use the i860 microprocessor assembler and linker.

i860™ 64-Bit Microprocessor Object File Utilities Reference Manual
464410-002
Provides reference information for using the i860 microprocessor object file utilities.

i860™ 64-Bit Microprocessor Simulator and Debugger Reference Manual
240437-003
Describes the i860 microprocessor debugger and simulator.

i860™ 64-Bit Microprocessor Programmer's Reference Manual
240329-002
Tells how to use the i860 microprocessor.

UNIX System V Manual Set
Describes UNIX System V.

UNIX System V Release 3.2 NFS User's/System Administrator's Guide and Reference
465725-001
Describes the NFS programming environment and provides user and system administration information.

UNIX System V Release 3.2 NFS Programmer's Guide and Reference
465726-001
Describes the NFS programming environment and tools.

UNIX System V Release 3.2 TCP/IP Administrator's Guide and Reference
465728-001
Describes TCP/IP Network administration.

UNIX System V Release 3.2 TCP/IP Programmer's Guide and Reference
465729-001
Describes the TCP/IP Network programming environment and provides information on programming tools.

UNIX System V Release 3.2 TCP/IP User's Guide and Reference
465727-001
Describes the TCP/IP Network programming environment and provides user information.

SYP301 Installation and User's Guide
451684-001
Tells how to install and start the System Resource Manager. Also provides hardware technical data.

Other Manuals

C: A Reference Manual - Harbison and Steele
480628-001
Describes the C programming language.

The C Programming Language - Kernighan and Ritchie
122008-002
Describes the C programming language.

UNIX V - The Quick Reference Guide
311533-003
Summarizes UNIX commands, buzzwords, C shell hints and standard directory layout.

TABLE OF CONTENTS

CHAPTER 1
PRODUCT FEATURES

CHAPTER 2
SOFTWARE INSTALLATION

CHAPTER 3
LIMITATIONS AND WORKAROUNDS

LIST OF TABLES

PRODUCT FEATURES 1

INTRODUCTION

The R3.3 system software runs on iPSC®/2 (with Intel386™-microprocessor-based CX nodes) and iPSC®/860 systems (with i860™-microprocessor-based RX nodes or with a combination of CX and RX nodes).

In general, the iPSC/860 system operates the same as the iPSC/2 system does, using the same commands and calls, but with added computational speed. Table 1-1 shows the major operational differences between the RX nodes and the CX nodes.

Table 1-1. Major Differences in the Operation of RX Nodes and CX Nodes

RX Nodes	CX Nodes
Peak performance of a 40-MHz i860 processor rated at 80 MFLOPS for single precision, 40 MFLOPS for double precision, and at 27 VAX MIPS	Peak performance of the processor rated at approximately 0.3 MFLOPS, and two or three VAX MIPS
Single process allowed on nodes, only pid 0 allowed	Multiple processes allowed on nodes
8 to 64M bytes of memory per node	4 to 16M bytes of memory per node
Fortran and C compilers available	Fortran, C, and Ada compilers available
On-board fast vector processing	Vector processing enhancement available with the VX option
On-board fast scalar processing	Scalar processing enhancement available with the SX option
Vectorizing preprocessor not available	VAST2 vectorizing preprocessor available
Node programs must be compiled and linked with the **icc** or **if77** compiler on the SRM or a Sun-3 or Sun-4 workstation, using the **-node** switch.	Node programs must be compiled and linked with the **cc** or **f77** compiler on the SRM, using the **-node** switch.

Host programs (for either node type) must be compiled and linked with the **cc** or **f77** compiler on the SRM, using the **-host** switch (and *not* using the **-node** switch).

Remote host programs (for either node type) must be compiled and linked with the appropriate C or Fortran compiler on the remote host, using the **-lhost** switch (and *not* using the **-node** switch).

NEW FEATURES

The iPSC Release 3.3 system software provides the following new features:

1. **Improved iPSC®/860 C and Fortran compilers**
   The iPSC/860 C and Fortran compilers have been improved over those delivered with Release 3.2. The new compilers are called **icc** and **if77**, and can be run on Sun-3 and Sun-4 workstations as well as on the SRM.

2. **Interactive Parallel Debugger**
   The new Interactive Parallel Debugger (IPD) replaces DECON as the source-level application debugger for the iPSC/2 and iPSC/860 systems. IPD offers conventional debugging facilities, such as breakpoints and the ability to display and modify memory, as well as features for examining the state of the message passing network. Differences from DECON include improved reliability, i860 machine-level debugging, and enhanced iPSC/860 Fortran support.

3. **Parallel Performance Analysis Tools**
   Release 3.3 includes an early release of the new Performance Analysis Tools (PAT). These tools help you analyze your iPSC/860 application programs and improve their performance. PAT consists of three utilities: an execution profiler that monitors the time spent in individual routines, a communication trace tool that assesses the time spent in communication, and an event trace tool that shows the interactions between processors and can monitor user-specified events. Performance data collected by the PAT utilities can be displayed graphically on Sun-3 and Sun-4 workstations.

4. **Profiler for iPSC®/860 node programs**
   The new **prof860** command lets you profile execution of RX node programs. It is based on the UNIX System V **prof** command and runs on Sun-3 and Sun-4 workstations as well as the SRM.

5. **Basic Math Library**
   The new Basic Math Library is an implementation of the BLAS (Basic Linear Algebra Subprograms) levels 1, 2, and 3 for iPSC/860 programs. It includes some one-dimensional Fast Fourier Transform (FFT) operations as well as the vector-vector operations, matrix-matrix operations, and vector-matrix operations of BLAS.

6. **Network Queueing System**
   The Network Queueing System (NQS), the *de facto* standard for batch queueing on supercomputers, is now supported on the SRM for iPSC/860 systems. It is completely compatible with the COSMIC version of NQS from NASA, and has been modified so that it can load cube jobs.

7. **More memory on RX nodes**
   The system software now supports RX nodes with 32 and 64M bytes of memory on iPSC/860 systems, as well as the existing 8 and 16M byte configurations.

8. **Node Shell on service node**
   The Node Shell (**nsh**) now executes on any I/O node without a SCSI interface (such as an ENET node or service node) rather than on the currently attached cube. This lets you invoke a shell on your iPSC/2 or iPSC/860 system without tying up any compute nodes. The new **-s** option to **nsh** makes the shell execute on the current cube, for backwards compatibility.

9. **Improved remote host security**
   A new file, */usr/ipsc/lib/ruser*, lists the workstations and users or groups who are allowed to access the iPSC system's remote host facilities. This new feature prevents unauthorized users from using your iPSC system over the network; it is available on both iPSC/2 and iPSC/860 systems.

10. **NX information structure**
    NX for the iPSC/860 system stores performance and status information (about message passing, interrupts, CPU usage, and so on) in a special read-only structure called *nxinfo*. This structure has now been made accessible to application programmers, who can use the information in it for performance improvements and checking the status of the nodes.

11. **Non-IEEE arithmetic for iPSC®/860 programs**
    This release supports a faster, less accurate non-IEEE mode for divide and square root, and the flushing of denormal data values to zero for RX node programs, in addition to the IEEE operations provided in Release 3.2.

12. **Enhanced accounting and security**
    The **getcube** command and *commser* process have been enhanced to provide hooks for user-developed accounting packages on both the iPSC/2 and iPSC/860 systems. You can use these hooks to keep track of system activity and control system access with a higher level of detail than the standard iPSC system administration tools provide. The **plogon** command and call also record more information than they did in Release 3.2, and the **plogon** command now has an **-i** option to report the pathname of the current log file.

13. **NX message buffering modification on iPSC®/860 systems**
    The **getcube** command has been enhanced to optionally disable message buffering on RX nodes. This provides a performance boost for some programs which overlap communication with computation. In addition, the default message buffering algorithm for RX nodes has been optimized for better performance.

SOFTWARE INSTALLATION 2

INTRODUCTION

This chapter describes how to install the Release 3.3 software.

CAUTION

If you use CIO Ethernet, Ada, or Lisp, *do not* install this Release 3.3 software on your system until you have the updated version of each package. Earlier versions of these software packages are not compatible with Release 3.3 of the system software.

The correct versions of these packages for use with Release 3.3 are as follows:

- CIO Ethernet Release 1.1
- Ada Release 1.2
- Lisp Release 1.4

If you have any questions, please contact SSD Customer Support:

1-800-421-2823 (Customer Support Hotline)
(44) 793 641 469 (in England)
Your Local Intel Sales Office (in Europe)
support@ssd.intel.com (Internet address)

There are three reasons to use the procedures in this chapter:

- You are upgrading your system from Release 3.2 to Release 3.3. In this case, begin with "REMOVING THE RELEASE 3.2 SYSTEM SOFTWARE" on page 2-21. You *do not* have to reinstall the UNIX operating system.
- You are upgrading your system from a release earlier than Release 3.2. In this case, you must reinstall all system software from scratch. Begin with "INSTALLING UNIX SYSTEM V/386 RELEASE 3.2 VERSION 2.1" on page 2-6.
- You are reinstalling your Release 3.3 system software because it has become damaged. In this case, begin with "INSTALLING UNIX SYSTEM V/386 RELEASE 3.2 VERSION 2.1" on page 2-6.

If the software on your system becomes damaged, please contact SSD Customer Support before performing any installations.

NOTE

Before performing any procedure in this chapter, read through the procedure completely to make certain that you understand what the procedure involves. If you have any questions before or during any procedure, please call SSD Customer Support for help.

If you perform the procedures in this chapter exactly as described, you will create a software system that is identical to the one originally installed by SSD.

The procedures in this chapter use the conventions described in the preface. You should also be aware of the following conventions used in these procedures:

- The instruction "Enter ***character(s)***" means type the indicated character(s), and then press the **<Enter>** key. For example, "Enter *y*" means type the letter `y`, and then press the key labeled `Enter`.
- In prompts, square brackets surround a default value. Pressing **<Enter>** selects the indicated default value.
- Some steps in these procedures cause a lot of information to be displayed on the system monitor. However, the step usually shows only the last message displayed. Do not be concerned if the indicated message does not appear immediately. Be patient. Some steps take several minutes to complete.

SOFTWARE DEPENDENCIES

Table 2-1 describes the dependencies between the various software packages in this release.

Table 2-1. Software Dependencies for Release 3.3 Installation

If you want to install this:	You must install this first:
TCP - Intel Corp. Release 3.0	UNIX System V Release 3.2
Ethernet Drivers Package for Intel TCP/IP Version 3.0	UNIX System V Release 3.2 TCP - Intel Corp. Release 3.0
iPSC System Software Release 3.3	UNIX System V Release 3.2 Cartridge Tape Utilities Version 2.1 TCP - Intel Corp. Release 3.0 Ethernet Drivers Package for Intel TCP/IP Version 3.0
iPSC Compiler Libraries System i860 Release 3.3	UNIX System V Release 3.2 Cartridge Tape Utilities Version 2.1
iPSC Remote Host Software Release 3.3	UNIX System V Release 3.2 Cartridge Tape Utilities Version 2.1
iPSC Simulator Software Release 3.0	UNIX System V Release 3.2 Cartridge Tape Utilities Version 2.1
iPSC NQS Software Release 1.0	UNIX System V Release 3.2 Cartridge Tape Utilities Version 2.1 iPSC System Software Release 3.3
iPSC Performance Analysis Tools i860 Release 1.0	iPSC Compiler Libraries System i860 (any release)
iPSC i860 Extension Software Release 3.3	iPSC System Software Release 3.3
iPSC C Compiler i860 Release 2.0	iPSC Compiler Libraries System i860 (any release)
iPSC FORTRAN Compiler i860 Release 2.0	iPSC Compiler Libraries System i860 (any release)
iPSC System S/W Release 3.3Update 5/91	UNIX System V Release 3.2 iPSC System Software Release 3.3 iPSC Compiler Libraries iPSC i860 Extension Software Release 3.3

Table 2-1 describes only the dependencies to *install* the named package. Further dependencies may exist in order to *use* the software in the package.

DISK SPACE REQUIREMENTS

Table 2-2 lists the disk space required by each iPSC software package in this release.

Table 2-2. Disk Space Requirements for Release 3.3

Package	Size	Difference from 3.2
TCP - Intel Corp. Release 3.0	3.0M bytes	0 [1]
Ethernet Drivers Package for Intel TCP/IP Version 3.0	1.1M bytes	0 [1]
iPSC System Software Release 3.3	19.3M bytes	+ 1.2M bytes
iPSC Compiler Libraries System i860 Release 3.3	4.6M bytes	+ 4.6M bytes [2]
iPSC Remote Host Software Release 3.3	1.3M bytes	+ 0.4M bytes
iPSC Simulator Software Release 3.0	0.7M bytes	0 [1]
iPSC NQS Software Release 1.0	2.6M bytes	+ 2.6M bytes [2]
iPSC Performance Analysis Tools i860 Release 1.0	6.8M bytes	+ 6.8M bytes [2]
iPSC i860 Extension Software Release 3.3	7.9M bytes	– 4.3M bytes
iPSC C Compiler i860 Release 2.0	7.2M bytes	+ 7.2M bytes [2]
iPSC FORTRAN Compiler i860 Release 2.0	7.9M bytes	+ 7.9M bytes [2]
Both i860 compilers together	11.2M bytes	+ 11.2M bytes [2]

1. This package has not changed from Release 3.2.
2. This package did not exist in Release 3.2.

To determine the free disk space you will need to upgrade your system from Release 3.2 to Release 3.3, add the numbers in the "Difference from 3.2" column for the packages you are installing.

NOTE

The packaging of system software has changed from Release 3.2 to Release 3.3, so you may have to install packages that did not exist in the previous release.

For example, if you had the iPSC System Software Release 3.2 and iPSC i860 Extension Software Release 3.2 packages installed, to get the equivalent functionality you would have to install the following packages:

- iPSC System Software Release 3.3 (+ 0.6M bytes)
- iPSC Compiler Libraries System i860 Release 3.3 (+ 3.9M bytes)
- iPSC i860 Extension Software Release 3.3 (– 4.1M bytes)
- Both i860 compiler packages (+ 11.2M bytes)

This means that installing Release 3.3 will take up an additional 11.6M bytes (0.6M bytes + 3.9M bytes – 4.1M bytes + 11.2M bytes) on your SRM's hard disk. If you do not have at least 11.6M bytes of free space, you will have to remove some files before you can perform the upgrade from Release 3.2 to Release 3.3.

If you are upgrading your system from Release 3.2 to Release 3.3, turn to "REMOVING THE RELEASE 3.2 SYSTEM SOFTWARE" on page 2-21. You *do not* have to reinstall the UNIX operating system.

If you are upgrading your system from a release earlier than Release 3.2 or reinstalling your Release 3.3 system software because it has become damaged, begin the installation by following the procedure in the next section.

INSTALLING UNIX SYSTEM V/386 RELEASE 3.2 VERSION 2.1

CAUTION

Installing the UNIX software reformats the hard disk. To prevent loss of data, save all user files and any system files that have been modified *before* installing the UNIX software.

If you are upgrading from UNIX Release 3.0, *do not replace the new system files with the modified system files* that you saved. Doing so downgrades those system files to UNIX Release 3.0. Instead, after installing the new UNIX software, make the same modifications to the new system files.

There are six major steps to installing the UNIX software:

1. Install the root and user filesystems
2. Install the base UNIX operating system
3. Install the Cartridge Tape Utilities
4. Install the UNIX add-on packages
5. Set up user and system logins
6. Install the remote terminal package

To install the UNIX software, you must perform all parts in the order specified.

Installing the Root and User Filesystems

Installation Time:	Approximately 30 minutes.
Installation Medium:	Floppy diskette (1) labeled "Boot Floppy."
Information you need:	Size of your hard disk (140MB or 380MB).

1. If the system is on, turn it off as described in Chapter 2 in the *iPSC®/2 and iPSC®/860 System Administrator's Guide*, in the sections "Stopping the UNIX Operating System" and "Removing Power."
2. Insert the installation diskette into the diskette drive.
3. Turn the system on.
4. When the following message appears:

   ```
   Strike ENTER to install the UNIX System on your hard disk.
   ```

   Press **<Enter>**.
5. When the following message appears:

   ```
   WARNING: A new installation of the UNIX System will destroy
   all files currently on the system.  Do you wish to continue
   (y or n)?
   ```

   Enter *y*.
6. When the following message appears:

   ```
   SELECT ONE OF THE FOLLOWING:

      1.  Create a partition
      2.  Change Active (Boot from) partition
      3.  Delete a partition
      4.  Exit (Update disk configuration and exit)
      5.  Cancel (Exit without updating disk configuration)
   Enter Selection:
   ```

   Enter *5*. (The partitioning was done when your system was originally installed.)

7. When the following message appears:

```
A surface analysis will now be done.
This will destroy all data on the hard disk.
Strike ENTER to continue or DEL to abort.
```

   Press **<Enter>**.

8. Following a message describing a suggested partitioning, this message appears:

```
Is this allocation acceptable to you (y/n)?
```

   Before responding to this message, make a note of the suggested number of cylinders for the root file system, and for the swap/paging area. You will use these to answer the questions in steps 10 and 11 below.

   Then, to respond to the allocation question, enter ***n***. This keeps the partitioning that was originally sent to you.

9. When the following message appears:

```
Do you wish to have separate root and usr filesystems (y/n)?
```

   Enter ***y***.

10. When the following message appears:

```
Do you want an additional /usr2 filesystem (y/n)?
```

   Enter ***n***.

11. When the following message appears:

```
How many cylinders would you like for swap/paging (1-xxx)?
```

   Enter the number of swap/paging cylinders you noted in step 7 above. Typical numbers are ***109*** for a 140M-byte drive, and ***78*** for a 380M byte drive.

12. When the following message appears:

```
How many cylinders would you like for root (1-xxx)?
```

Enter the number of root cylinders you noted in step 7 above. Typical numbers are ***201*** for a 140M-byte drive, and ***143*** for a 380M byte drive.

13. When the following message appears:

```
Is this allocation acceptable to you (y/n)?
```

Enter ***y*** if the cylinder quantities for root and swap/paging are what you entered.

14. When the following message appears:

```
Reboot the system now.
```

The installation of the root and user filesystems is complete.

15. Remove the diskette from the diskette drive.

To complete the installation of the UNIX software, install the base UNIX operating system, as described in the next procedure.

Installing the Base UNIX Operating System

NOTE

You must install the root and user filesystems (as described in the previous procedure) *before* you install the base UNIX operating system.

Installation Time:	Approximately 20 minutes.
Installation Medium:	Cartridge tape labeled "UNIX System V/386, R3.2 V2.1". Diskette labeled "Cartridge Tape Utilities"
Information you need:	*root* password. *install* password.

---

16. Press **<Ctrl-Alt-Del>** to reboot the system.

17. When the following message appears:

```
Installation from Cartridge Tape is available using Interrupt
#5 and Address Range 300 through 301. Are you installing from
tape (y/n)?
```

    Enter *y*.

18. When the following message appears:

```
Please make sure your Cartridge Tape hardware is configured
correctly. Insert System Installation Tape in drive and press
<RETURN>
```

    A. Insert the installation tape into the tape drive (label up, exposed tape to the left; push until the tape cartridge locks into place).

    B. Press **<Enter>**.

19. When the system prompts for a *root* password, enter the password that you have chosen for the *root* login.

20. When the system prompts for an *install* password, enter the password that you have chosen for the *install* login.

21. When the following message appears:

```
Strike ENTER when ready
or ESC to stop.
```

    Press **<Esc>**.

22. When the following message appears:

```
Reboot the system now
```

    Press **<Ctrl-Alt-Del>** to reboot the system.

23. When the following message appears:

```
Console login:
```

    Login as *root*.

24. Enter ***installpkg***.

25. Insert the Cartridge Tape Utilities installation diskette into the diskette drive and turn the handle down.

26. When the following message appears:

```
Strike ENTER when ready
or ESC to stop
```

    Press **<Enter>**.

27. When the following message appears:

```
Type the interrupt number and strike the ENTER key or type Q
to cancel installation.
```

    Enter ***5***.

28. When the following message appears:

```
Strike ENTER when ready
or ESC to stop
```

Press **<Enter>**.

29. Again, when the following message appears:

```
Strike ENTER when ready
or ESC to stop
```

Press **<Enter>**.

30. When the following message appears:

```
Reboot the system now.
```

A. Remove the Cartridge Tape Utilities diskette from the diskette drive.

B. Leave the tape in the tape drive for the next procedure.

C. Press **<Ctrl-Alt-Del>** to reboot the system.

31. When the following message appears:

```
Console login:
```

The installation of the base UNIX operating system is complete.

To complete the installation of the UNIX software, install the UNIX add-on packages, as described in the next procedure.

Installing the UNIX Add-On Packages

NOTE

You must install the base UNIX operating system (as described in the previous procedure) *before* you install the UNIX add-on packages.

Also, if any of the UNIX Add-On packages are currently installed, you must remove them before installing the new software. Refer to the section "Remove Add-On Software Packages" in the *AT&T UNIX System V/386 Release 3.2 System Administrator's Guide.*

Installation Time:	Approximately 45 minutes.
Installation Medium:	Cartridge tape labeled "UNIX System V/386, R3.2 V2.1."
Information you need:	*root* password.

---

32. Login as *root.*

33. Enter ***installpkg***.

34. When the following message appears:

```
Are you installing from tape (y/n)?
```

    Enter *y*.

35. When the following message appears:

```
Insert Installation Tape in drive and press <RETURN>.
```

    (If you removed the tape after the last procedure, reinsert the tape into the tape drive.)

    Press **<Enter>**.

36. When the following message appears:

```
Do you want to install all of the above packages? <y/[n]>:
```

Press **<Enter>** (to accept the default *n*).

37. The system then prompts you to select the packages to install. Enter *y* after each of the following packages, and *n* after all the other packages (or press **<Enter>** to accept the default *n*):

```
Editing Package Version 2.0
Extended Terminal Interface Package Version 2.0
C Software Development Set 4.1.6
Network Support Utilities Package (1.2) Version 2.0
2 Kilobyte File System Utility Package Version 2.0
Kernel Debugger(s) - Version 2.0
PC586 Ethernet Driver - Version 1.0
System Administration Software
```

A. When the Kernel Debugger package is being installed, the following appears:

```
Which kernel debugger(s) do you want to install?

   1) DEBUGGER (polish calculator style)
   2) GDEBUGGER (traditional)
   3) both DEBUGGER and GDEBUGGER

Choose 1, 2, or 3
```

Enter *1*.

B. When the System Administration Software package is being installed, the following appears:

```
Do you want to give passwords to administration login?
(y/n) [n]
```

Press **<Enter>** (to accept the default *n*).

38. When the following message appears:

```
Strike ENTER when ready
or ESC to stop.
```

Press **<Enter>**.

39. When the following message appears:

```
Reboot the system now.
```

Press **<Ctrl-Alt-Del>** to reboot the system.

40. When the following message appears:

```
Console login:
```

The installation of the UNIX add-on packages is complete.

41. Remove the tape from the tape drive (push tape in to release catch).

To complete the installation of the UNIX software, set up the user and system logins, as described in the next procedure.

Setting Up User and System Logins

NOTE

You must install the UNIX add-on packages (as described in the previous procedure) *before* you set up the user and system logins.

Installation Time:	Approximately 10 minutes, depending on number of user logins being set up.
Installation Medium:	None.
Information you need:	Name of your time zone. Whether you observe daylight savings time. User login names, IDs, passwords, etc. Administrative passwords. System passwords. *node name* (name by which other machines know this machine).

---

42. Login as *setup*.

43. Answer the questions that **setup** asks.

NOTE

One of the questions asks you to select your time zone from a list. If your time zone is not included in this list, select GMT, and then enter the correct local time when a later question asks for the time.

To complete the installation of the UNIX software, install the remote terminal package, as described in the next procedure.

Installing the Remote Terminal Package

Installation Time:	Approximately 5 minutes.
Installation Medium:	Floppy diskette (1) labeled "Remote Terminal Package."
Information you need:	*root* password.

44. Login as *root*.
45. Enter ***installpkg***.
46. When the following message appears:

```
Are you installing from tape (y/n)?
```

    Enter ***n***.

47. Insert the installation diskette into the diskette drive, and turn the handle down.
48. When the following message appears:

```
Strike ENTER when ready
or ESC to stop.
```

    Press **<Enter>**.

49. When the following message appears:

```
Enter option:
```

    Enter ***1***.

50. When the following message appears:

```
Enter a file name, 'all', 'done', or 'files':
```

Enter ***all***.

51. When the following message appears:

```
Enter a file name, 'all', 'done', or 'files':
```

Enter ***done***.

52. When the following message appears:

```
Enter option:
```

Enter ***0***.

53. Remove the diskette from the diskette drive.

54. If you are not planning to install the TCP/IP and Ethernet drivers (as described in the next section), press **<Ctrl-D>** to log out.

This completes the installation of the UNIX software.

INSTALLING TCP/IP AND ETHERNET DRIVERS

NOTE

The UNIX software must be installed *before* you can install the TCP/IP software. Refer to the discussion of "INSTALLING UNIX SYSTEM V/386 RELEASE 3.2 VERSION 2.1" on page 2-6 of this release note.

Also, if the TCP/IP and Ethernet Drivers packages are currently installed, you must remove them before installing the new software. Refer to Section 4.2, "Removing TCP/IP," on page 1-6 of the *Intel® TCP/IP for System V/386 Administrator's Guide and Reference.*

TCP/IP software *must* be installed *before* the iPSC Release 3.3 system software. If you install the TCP/IP software *after* the iPSC Release 3.3 system software package has been installed, you *must* re-install Release 3.3 System software.

CAUTION

When removing the TCP/IP and Ethernet Drivers packages, be careful *not* to remove the package called "PC586 Ethernet Driver." Removing this package will prevent you from installing *any* other software, and once removed it cannot easily be restored.

Installation Time:	Approximately 30 minutes.
Installation Media:	Floppy diskettes (3) labeled "Intel TCP/IP System V/386 Release 3.2." Floppy diskette (1) labeled "Ethernet Drivers Release 3.2."
Information you need:	*root* password. Internet address. Network name. Whether network is subnetted. Domain name.

CAUTION

The *remove* and *install* scripts for TCP/IP remove/overwrite the following configuration files:

/etc/hosts
/etc/networks
/etc/gateways
/etc/hosts.equiv
/usr/lib/named/named.hosts
/usr/lib/named/named.local
/usr/lib/named/named.rev
/usr/lib/named/named.soa
/usr/lib/named/root.cache

If you are updating a previous release of Intel TCP/IP, you may want to rename or backup these files *before* removing the previous release. After you successfully install TCP/IP Release 3.0, you can restore these files.

Perform the installation procedure described on page 1-6 of the *Intel® TCP/IP for System V/386 Administrator's Guide and Reference* in the section entitled, "Sample Installation: A Three System Local Area TCP Network."

You have now completed installation of all UNIX operating system software. If you are upgrading your system from a release earlier than Release 3.2 or reinstalling your Release 3.3 system software because it has become damaged, the next section does not apply to you. Turn to "INSTALLING THE RELEASE 3.3 SYSTEM SOFTWARE" on page 2-22 to complete installation of Release 3.3.

REMOVING THE RELEASE 3.2 SYSTEM SOFTWARE

If you are upgrading your software from a release prior to Release 3.2, skip this section. This section only applies to you if you are upgrading your system from Release 3.2 to Release 3.3.

Before installing Release 3.3, you must remove the Release 3.2 iPSC software packages. Intel SSD recommends that the packages be removed in the following order:

1. iPSC X Window System Release 1.1 (if installed)
2. iPSC CIO Ethernet Software Release 1.1 (if installed)
3. iPSC Simulator Software Release 3.2 (if installed)
4. iPSC Remote Host Software Release 3.2 (if installed)
5. iPSC i860 Extension Software Release 3.2 (if installed)
6. iPSC System Software Release 3.2

It is *not* necessary to remove any of the packages that are part of UNIX System V Release 3.2.

The procedure for removing packages is as follows:

1. Log in as *root* on the system console
2. Enter single user mode. This prevents other users from logging in.

   ```
   shutdown -is
   ```

3. Mount the user partition. The **removepkg** command resides in this partition. The single-user shutdown does not automatically mount the user partition.

   ```
   /etc/mount /dev/dsk/0s3 /usr
   ```

4. Invoke **removepkg**.

   ```
   /usr/bin/removepkg
   ```

5. From the menu, choose the package to be removed.
6. After the package is removed, press **<Esc>** to get the root prompt back. D*o not* press **<Enter>**. Pressing **<Enter>** results in a shutdown.
7. Repeat steps 4 through 6 for each package to be removed.

After the desired packages are removed, install the system software as described in the following sections. Installation instructions for options are described in the release notes for that product.

INSTALLING THE RELEASE 3.3 SYSTEM SOFTWARE

NOTE

You *must* install the TCP/IP software (as described in the *Intel® TCP/IP for System V/386 Administrator's Guide and Reference*) *before* you install the Release 3.3 system software. Refer to the section entitled "INSTALLING TCP/IP AND ETHERNET DRIVERS" on page 2-19 of this manual.

The install script modifies the three standard *crontab* files and creates a new **/usr/bin/man** command. If you want to keep your existing *crontab* files or **man** command, save them before performing this procedure, and then restore them afterwards.

Installation Time:	Approximately 30 minutes.
Installation Medium:	1 cartridge tape labeled "iPSC®/2 System Software R3.3"
Information you need:	*root* password.

---

1. Login as *root*.
2. Enter single user mode. This prevents other users from logging in.

   ```
   shutdown -is
   ```

3. Mount the user partition. The **removepkg** command resides in this partition. The single-user shutdown does not automatically mount the user partition.

   ```
   /etc/mount /dev/dsk/0s3 /usr
   ```

4. Enter ***installpkg***.
5. When the following message appears:

   ```
   Are you installing from tape (y/n)?
   ```

   Enter *y*.

6. When the following message appears:

```
Insert Installation Tape in drive and press <RETURN>.
```

   Insert the installation tape labeled "iPSC®/2 System Software R3.3" into the tape drive (label up, exposed tape to the left; push until the tape cartridge locks into place).

7. Press **<Enter>**.

8. Eventually, the following message appears:

```
Do you want to install all of the above packages? <y/[n]>:
```

   Enter *y*.

9. When the following message appears:

```
Strike ENTER when ready or
ESC to stop
```

   Press **<Enter>**.

10. When the following message appears:

```
Reboot the system now.
```

   A. Remove the tape from the tape drive (push the tape in to release the catch).

   B. Press **<Ctrl-Alt-Del>** to reboot the system.

11. When the following message appears:

```
Console login:
```

   log in as *root* again.

12. Enter the following command to remove permission for ordinary users to use the **rebootcube** command:

```
chmod 700 /usr/bin/rebootcube
```

   This **chmod** command means that only the superuser can reboot the cube, making it less likely that someone will reboot the cube without understanding the consequences of their actions. This is necessary because in Release 3.3 it is more likely for users to be accessing the cube over the network, and **rebootcube** does not check for network access before rebooting.

The installation of the Release 3.3 system software is now complete.

INSTALLING THE RELEASE 3.3 COMPILER LIBRARIES

NOTE

You must install the iPSC/2 System Software R3.3 software before you install the iPSC/860 compiler libraries from the iPSC®/860 Compiler Libraries System R3.3 tape.

Installation Time:	Approximately 10 minutes.
Installation Medium:	1 cartridge tape labeled "iPSC®/860 Compiler Libraries System R3.3"
Information you need:	*root* password.

---

1. Login as *root*.
2. Enter ***installpkg***.
3. When the following message appears:

   ```
   Are you installing from tape (y/n)?
   ```

   Enter *y*.
4. When the following message appears:

   ```
   Insert Installation Tape in drive and press <RETURN>.
   ```

   Insert the installation tape labeled "iPSC®/860 Compiler Libraries System R3.3" into the tape drive (label up, exposed tape to the left; push until the tape cartridge locks into place).
5. Press **<Enter>**.
6. Eventually, the following message appears:

   ```
   Do you want to install all of the above packages? <y/[n]>:
   ```

   Enter *y*.

When the prompt returns, the installation of the iPSC/860 Release 3.3 compiler libraries is complete.

INSTALLING THE RELEASE 3.3 iPSC/860 COMPILERS

At this point, you install the iPSC/860 Fortran and/or C compilers on the SRM. See the *iPSC®/860 Fortran Compiler Release 2.0 Software Product Release Notes* and/or *iPSC®/860 C Compiler Release 2.0 Software Product Release Notes* for instructions, then return to this document to complete your software installation.

NOTE

The Compiler Release Notes describe only how to install the compilers on the SRM. Installation of the cross-compilers on Sun-3 or Sun-4 workstations is performed later, as part of the procedure described under "INSTALLING THE CROSS-DEVELOPMENT ENVIRONMENT" on page 2-31.

INSTALLING THE RELEASE 3.3 EXTENSION SOFTWARE

NOTE

You *must* install the iPSC/860 Release 3.3 Compiler Libraries *before* you install the iPSC/860 Performance Analysis Tools package from the iPSC/2/860 Extension Software R3.3 tape.

If you have an iPSC/2 system, *do not* install the iPSC/860 extension software from the iPSC/2/860 Extension Software R3.3 tape. This software is unnecessary for the iPSC/2 system.

If you have a 140M-byte disk on your SRM, please see "DISK SPACE REQUIREMENTS" on page 2-4 to determine how much disk space is required to install the extension software. You may need to upgrade to a larger disk to install all the software you need.

Installation Time:	Approximately 30 minutes.
Installation Medium:	1 cartridge tape labeled "iPSC®/2/860 Extension Software R3.3"
Information you need:	*root* password.

1. Login as *root*.
2. Enter ***installpkg***.
3. When the following message appears:

   ```
   Are you installing from tape (y/n)?
   ```

   Enter *y*.
4. When the following message appears:

   ```
   Insert Installation Tape in drive and press <RETURN>.
   ```

   Insert the installation tape labeled "iPSC®/2/860 Extension Software R3.3" into the tape drive (label up, exposed tape to the left; push until the tape cartridge locks into place).
5. Press **<Enter>**.
6. Eventually, the following message appears:

   ```
   Do you want to install all of the above packages? <y/[n]>:
   ```

   What you install depends on what you use. To save disk space, do not install the Remote Host, Simulator, NQS, or PAT software on your system unless you need it. You can install this software later if desired. However, you *must* install the iPSC i860 extension software before you can use your iPSC/860 system.

   If you don't want to install all of the packages, press **<Enter>** (to accept the default ***n***). The system then prompts you to select the packages to install. Enter *y* for the packages you want and ***n*** for the ones you don't want.

When the prompt returns, the installation of the iPSC/860 Release 3.3 extension software is complete.

- If you installed PAT, you must install the cross-development environment, as described under "INSTALLING THE CROSS-DEVELOPMENT ENVIRONMENT" on page 2-31, then perform the configuration procedure described under "Configuring PAT Software" on page 2-32 to make PAT usable.

- If you installed NQS, you must perform the configuration procedure described in the following section to make NQS usable.

Configuring NQS Software

The NQS software is on the iPSC®/2/860 Extension Software R3.3 tape; you choose whether or not to install it when you install this tape. If you do install the NQS package, you need to configure it. This section describes a procedure that configures the NQS software for use with a basic iPSC system. The procedure describes the configuration steps necessary at the SRM.

After the procedures in this section are complete, the SRM will be configured as machine number one (mid=1) in a "locally functional" NQS environment with a single batch queue named `bq`. The locally functional NQS environmental will be able to accept batch jobs submitted from the SRM, but it will not yet be able to accept jobs submitted from other machines. In order to use NQS with other machines or in more complex systems, you must perform the more detailed configuration procedures described in the *iPSC®/2 and iPSC®/860 Network Queueing System Manual*. Refer to Appendix A, "Reconfiguring NQS" in the *iPSC®/2 and iPSC®/860 Network Queueing System Manual* for more information.

Installation Time: Approximately 30 minutes.

Installation Medium: None.

Information you need: Root password.

NOTE

All pathnames must be absolute pathnames (i.e., they must start with `/`).

Perform the following steps at the iPSC SRM:

1. Login as *root* on the SRM.

2. Change directories to the NQS source directory, */usr/src/nqs*, as follows:

```
cd /usr/src/nqs
```

3. Make sure that your umask is set to 0 by entering the following:

```
umask 0
```

4. Compile NQS code as follows:

```
make
```

5. Install the NQS executable code by entering the following:

```
make install
```

Before the NQS daemon can be started, the host machine must be in the NQS database. Once the daemon is running, the **qmgr** command is functional and you can use the **qmgr** command to create a queue for job requests. There is an entry point in the *make* file that will allow you to do this automatically. The automatic method will accomplish the following tasks:

1. Insert the SRM machine name as machine #1.
2. Start the NQS daemon.
3. Create a single batch queue by the name of `bq`.
4. Make the `bq` batch queue the default queue for the **qsub** command.

If you want your machine to accept requests from other machines on the network or accomplish other tasks, you must set up the machine IDs and the queues by hand. Refer to Appendix A, "Reconfiguring NQS," of the *iPSC®/2 and iPSC®/860 Network Queueing System Manual* for instructions on setting up the machine IDs and queues by hand.

Perform the following steps to start NQS using the automatic method:

1. Login as *root* on the SRM
2. Change directories to the source directory, */usr/src/nqs* as follows:

```
cd /usr/src/nqs
```

3. Make sure that your umask is set to 0 by entering the following:

```
umask 0
```

4. Start the make file at the start-up tag as follows:

```
make startup
```

You are now able to submit local jobs to the batch queue bq by using the **qsub** command. In order to submit jobs from remote locations in the network, you must set each remote machine up using the detailed procedures presented in Appendix A, "Reconfiguring NQS," of the *iPSC®/2 and iPSC®/860 Network Queueing System Manual.*

INSTALLING THE RELEASE 3.3 SYSTEM SOFTWARE UPDATE 5/91

NOTE

You *must* install the iPSC System Software R3.3, Compiler Libraries, and Release 3.3 Extension Software before you install the iPSC System Software Update tape. The update software only updates these packages if they are installed on your system. If you reinstall any of these packages, you must also reinstall the update.

Installation Time:	Approximately 15 minutes.
Installation Medium:	1 cartridge tape labeled "iPSC®/2/860 System S/W R3.3 Update 5/91" (part number 312186-001)
Information you need:	*root* password.

---

1. Login as *root.*

2. Execute the following to put the SRM in the maintenance mode.

```
shutdown -g0 -y -is
```

The following message is displayed:

```
Type Ctrl-d to proceed with normal startup
(or give root password for system maintenance):
```

3. Enter the root password. The following message is displayed:

```
Entering System Maintenance Mode
```

4. Mount the *usr* partition:

```
/etc/mount /dev/dsk/0s3 /usr
```

The following message is displayed:

```
mount -f s51K /dev/dsk/0s3 /usr
```

5. Change to the */usr/tmp* directory:

```
cd /usr/tmp
```

6. Insert the cartridge tape labeled "iPSC/2/860 System S/W R3.3 Update 5/91" in the SRM tape drive.

7. Copy the contents of the tape to the hard disk:

```
cpio -icmB -I/dev/rmt/c0s0n
```

NOTE

You *must* specify the no-rewind device **c0s0n**; otherwise, the tape will rewind and you will need to reinstall from the beginning.

8. Run the installation script:

```
sh install
```

9. Enter **<Ctrl-D>** to boot the system back to multi-user level. The following message is displayed:

```
Enter run level (0-6, s or S):
```

Enter *2*. This sets the run level for multi-user mode. A is series of startup messages is displayed. When `Console Login:` is displayed, the R3.3 Update Software installation is complete.

INSTALLING THE CROSS-DEVELOPMENT ENVIRONMENT

These are the basic instructions for installing the cross-development environment on an NFS network. This should be done after all the optional iPSC software packages have been installed on the SRM.

Installation Time:	Approximately 25 minutes.
Installation Medium:	None.
Information you need:	Root password for the file server.

---

1. Have the system administrator of the network create an SSD tools root directory. There needs to be about 40 megabytes of disk space on the NFS file server that has this directory. The directory should be owned by *root* and writable only by *root* but readable and executable by all.

   For example, if the SSD tools root directory is to be */vol/tools/ssd*, perform the following commands as *root* on the file server:

   ```
   mkdir /vol/tools/ssd
   chmod 755 /vol/tools/ssd
   ```

2. Set the environment variable **IPSC_XDEV** to the full pathname of the SSD tools root directory. For example:

   ```
   setenv IPSC_XDEV /vol/tools/ssd
   ```

   This environment variable is used by all of SSD's cross-development tools.

3. Copy all the files and directories from the directory */usr/ipsc/XDEV/i860* on the SRM to the SSD tools root directory on the file server.

   For example, using **rcp** from a Sun network:

   ```
   rcp -p -r SRM:/usr/ipsc/XDEV/i860 $IPSC_XDEV
   ```

   Or using **tar** from a Sun network:

   ```
   cd $IPSC_XDEV
   rsh SRM "cd /usr/ipsc/XDEV/i860;tar cf - ."|tar xf -
   ```

   In both of the above commands, *SRM* represents the network name of the SRM.

The installation of the cross-development environment is now complete.

Contents of the Cross-Development Environment

The subdirectories in the **$IPSC_XDEV** directory created by the procedure in the previous section are as follows:

$IPSC_XDEV/i860/bin	Executables for the SRM
$IPSC_XDEV/i860/bin.sun3	Executables for the Sun3
$IPSC_XDEV/i860/bin.sun4	Executables for the Sun4
$IPSC_XDEV/i860/include	Standard include files
$IPSC_XDEV/i860/include-ipsc	iPSC include files
$IPSC_XDEV/i860/lib-coff	COFF libraries
$IPSC_XDEV/i860/lib	Files for SRM executables (ipd daemon)
$IPSC_XDEV/i860/lib.sun3	Files/libraries for Sun3 executables
$IPSC_XDEV/i860/lib.sun4	Files/libraries for Sun4 executables

Using the Cross-Development Environment

Once you have installed the cross-development environment, the users must set the environment variable **IPSC_XDEV** to the full pathname of the SSD tools root directory and then add the appropriate "bin" directory to their execution search paths. For example, C-shell users could place the following commands in their *.cshrc* files:

```
setenv IPSC_XDEV /vol/tools/ssd
set path=( $path $IPSC_XDEV/i860/bin.`arch` )
```

Configuring PAT Software

The PAT software is on the iPSC®/2/860 Extension Software R3.3 tape; you choose whether or not to install it when you install this tape. If you do install the PAT package, you need to configure it. This section describes a procedure that configures the PAT software for use with a basic iPSC system. The procedure describes the configuration steps necessary at a Sun workstation.

Installation Time:	Approximately 10 minutes.
Installation Medium:	None.
Information you need:	Pathnames to the cross-development directories.

NOTE

All pathnames must be absolute pathnames (i.e., they must start with /). The cross-development environment must also be installed on your Sun workstation, as described under "INSTALLING THE CROSS-DEVELOPMENT ENVIRONMENT" on page 2-31, before you can configure the PAT software.

The software is installed in the following cross-development directories on the SRM.

/usr/ipsc/XDEV/i860/bin.sun4
/usr/ipsc/XDEV/i860/bin.sun3
/usr/ipsc/XDEV/i860/lib.sun4
/usr/ipsc/XDEV/i860/lib.sun3

Perform the following steps to configure your Sun workstation for use with the PAT utilities.

1. After installing the cross-development environment on your Sun workstation, edit the following two files:

   $IPSC_XDEV/i860/lib.sun4/express.cst
   $IPSC_XDEV/i860/lib.sun3/express.cst

2. In each of the above files, change the following line:

   ```
   PARASOFT:=/usr/ipsc/XDEV/i860
   ```

   to reflect the location of the cross-development directories on your Sun system. For example, if the cross-development directories on your Sun system are installed under */vol/tools/ssd*, change the line to:

   ```
   PARASOFT:=/vol/tools/ssd/i860
   ```

NOTE

You must use the full path name of the directory.

3. In your *.login* or *.cshrc* file, set the environment variable EXPRESS to point to the file express.cst. For example:

```
setenv EXPRESS $IPSC_XDEV/i860/lib.`arch`/express.cst
```

Your Sun workstation should now be configured properly to use any of the PAT utilities. Refer to the *iPSC®/860 Parallel Performance Analysis Tools Manual* for instructions on using the PAT utilities.

INSTALLING REMOTE HOST SOFTWARE ON A DISKFUL HOST

NOTE

The remote host software package must be installed on the SRM *before* you can install the remote host software on a remote host. Refer to the section "INSTALLING THE RELEASE 3.3 EXTENSION SOFTWARE" on page 2-25 of this manual.

Use the following procedure to install the remote host software on a remote host that has its own disk. If installing on a diskless host, see "INSTALLING REMOTE HOST SOFTWARE ON A DISKLESS HOST" on page 2-36.

Installation Time: Approximately 30 minutes.

Installation Medium: None.

Information you need: *root* password.

---

1. Login as *root* on the remote host.

2. Create a directory on the remote host to contain the remote host source code. This directory can be located wherever you choose.

3. Copy the remote host source code from the directory */usr/ipsc/rhost* on the SRM to the directory you created in the previous step.

4. On the remote host, edit the makefile in the top-level remote host source code directory so that the installation directories are correct for your system:

NOTE

All pathnames must be absolute pathnames (i.e., they must start with `/`).

IPSCDIR Contains daemons and named sockets. The default is */usr/ipsc* and the subdirectories *lib* and *log* will be created.

BINDIR Contains cube binaries. On the SRM these files are in */usr/bin*. Any directory can be used for the binaries, as long as the cube users have it in their search path. The default directory on the remote host is *IPSCDIR/bin*.

LIBDIR Contains *libhost.a*. The default location for the library is in */usr/lib*. Again, any directory can be used as long as the user application makefiles reflect the correct directory.

INCDIR Contains the iPSC/2 include files *cube.h* and *fcube.h*. The default is *IPSCDIR/include*.

5. Kill any existing *commser* and *fserver* daemons on the remote host.

6. Enter ***make*** to build binaries on the remote host.

7. Enter ***make install*** to install the binaries in the prescribed directories.

8. Edit the *srms* file (*IPSCDIR/lib/srms*) to include the names of all SRMs that the remote host may access.

9. Edit the file */usr/ipsc/lib/ruser* on the SRM to include the host and user names for all hosts and users that may access the SRM using remote host software. (The format of this file is the same as that of */etc/hosts.equiv*).

10. Edit the user's *.cshrc* file to include the following line:

```
setenv TTY 'tty' >& /dev/null
```

This completes the installation of the remote host software on a remote host. In order to compile programs for RX nodes on the remote host, you must also install the cross-development environment, as described under "INSTALLING THE CROSS-DEVELOPMENT ENVIRONMENT" on page 2-31. This will make the cross-compilers **icc** and **if77** available to you on your Sun-3 or Sun-4 workstation.

INSTALLING REMOTE HOST SOFTWARE ON A DISKLESS HOST

NOTE

The remote host software package must be installed on the SRM *before* you can install the remote host software on a remote host. Refer to the section "INSTALLING THE RELEASE 3.3 EXTENSION SOFTWARE" on page 2-25 of this manual.

Use the following procedure to install the remote host software on a remote host that does not have its own disk. If installing on a diskful host, see "INSTALLING REMOTE HOST SOFTWARE ON A DISKFUL HOST" on page 2-34.

Installation Time: Approximately 30 minutes.

Installation Medium: None.

Information you need: *root* password.

NFS security features conflict with the installation procedure for remote host software described in the previous section. NFS does not allow *root* access to remotely-mounted filesystems. The remote host software requires the builder and installer to have *root* access to the filesystems on which it is installed. Since a diskless client cannot have *root* access, a different installation procedure must be followed.

NOTE

Some NFS mounted file systems also do not allow **setuid** programs to be installed on them. The remote host software executables *must* be installed on a mounted file system that allows a **setuid** program to execute.

In the examples in this procedure, the NFS file server (called **server**) has a directory called */export/sun386/local* which its diskless clients (one is called **client**) mount as */usr/local*. The remote host software for the clients is to be installed in a directory called */usr/local/ipsc* from the client's point of view. This path will be compiled into several of the remote host programs.

1. Become *root* on the NFS file server and **cd** to the parent directory that will contain the subdirectory where the remote host software is to be installed. For example:

```
{server:49} cd /export/sun386/local
```

2. Create the directory to contain the remote host source code, and set its protection so anyone can write to this directory: For example:

```
{server:50} mkdir ipsc
{server:51} chmod a+w ipsc
```

3. Become *root* on the client machine and **cd** to the directory you created in the previous step. For example:

```
{client:17} cd /usr/local/ipsc
```

4. Copy the remote host source code from the directory */usr/ipsc/rhost* on the SRM to the current directory.

5. Edit the makefile in the current directory so that the installation directories are correct for your system:

NOTE

All pathnames must be absolute pathnames (i.e., they must start with /) and must be from the client's view of the file system (i.e., */usr/local/ipsc* instead of */export/sun386/local/ipsc*).

IPSCDIR Contains daemons and named sockets. The default is */usr/ipsc* and the subdirectories *lib* and *log* will be created.

BINDIR Contains cube binaries. On the SRM these files are in */usr/bin*. Any directory can be used for the binaries, as long as the cube users have it in their search path. The default directory on the remote host is *IPSCDIR/bin*.

LIBDIR Contains *libhost.a*. The default location for the library is in */usr/lib*. Again, any directory can be used as long as the user application makefiles reflect the correct directory.

INCDIR Contains the iPSC/2 include files *cube.h* and *fcube.h*. The default is *IPSCDIR/include*.

6. Kill any existing *commser* and *fserver* daemons on the remote host.

7. Type ***make***. Some directories will be created in */usr/local/ipsc* and some header files installed. The remote host software libraries and executables will be created.

8. When the **make** is completed, become *root* on the NFS file server again and remove the world write permissions from the directory created in step 2. For example:

```
{server:52} cd /export/sun386/local
{server:53} chmod o-w ipsc
```

9. **cd** to the directory containing the remote host source. For example:

```
{server:54} cd /export/sun386/local/ipsc
```

10. Edit the **makefile** and set **IPSCDIR** correctly for the server's view of the filesystem (in this example, this would be */usr/sun386/local/ipsc*).

11. Install the software by typing ***make install***.

12. Edit the *srms* file (*IPSCDIR/lib/srms*) to include the names of all SRMs that the remote host may access.

13. Edit the file */usr/ipsc/lib/ruser* on the SRM to include the host and user names for all hosts and users that may access the SRM using remote host software. (The format of this file is the same as that of */etc/hosts.equiv*).

14. Edit the user's *.cshrc* file to include the following line:

```
setenv TTY `tty` >& /dev/null
```

This completes the installation of the remote host software on a remote host. In order to compile programs for RX nodes on the remote host, you must also install the cross-development environment, as described under "INSTALLING THE CROSS-DEVELOPMENT ENVIRONMENT" on page 2-31. This will make the cross-compilers **icc** and **if77** available to you on your Sun-3 or Sun-4 workstation.

LIMITATIONS AND WORKAROUNDS 3

This chapter describes known limitations and suggested workarounds for the following Release 3.3 system software components:

Documentation
NX Node Executive
Node Shell
Concurrent File System™
Concurrent File System™ Tape
Intel386™ Assembler and Linker
iPSC®/2 C Compiler
iPSC®/2 Fortran Compiler
Interactive Parallel Debugger
Compiler Libraries
RX Vector Library
Basic Math Library (*libkmath.a*)
Remote Host
Network Queueing System (NQS)
iPSC®/2 Simulator
UNIX Software
TCP/IP on the SRM
Cube Diagnostic Program (CDP)

NOTE

Read the following sections carefully. Report any problems you encounter while using your iPSC system to SSD Customer Support at:

1-800-421-2823 (Customer Support Hotline)
(44) 793 641 469 (in England)
Your Local Intel Sales Office (in Europe)
support@ssd.intel.com (Internet address)

DOCUMENTATION

1. **Remove some i860 manuals from your manual set**
   Four Intel i860 manuals are shipped as part of the iPSC documentation package. These four manuals are shrink-wrapped in a single package. Only two of these manuals apply to the iPSC/860 systems. The two manuals that apply are the *i860™ 64-Bit Microprocessor Assembler and Linker Reference Manual* and the *i860™ 64-Bit Microprocessor Simulator and Debugger Reference Manual.*

   There are two manuals in the package that are not supported and should not be used:

   - *i860™ 64-Bit Microprocessor Math Library Reference Manual*
     The iPSC system does not support the i860 math libraries as documented in this manual. The iPSC supports the math libraries described in the UNIX System V/386 documentation.

   - *i860™ 64-Bit Microprocessor Object File Utilities Reference Manual.* The object file utilities described in this manual have much the same functions as those that the iPSC supports, but the names are different. iPSC object file utilities are described in the *iPSC®/2 and iPSC®/860 User's Guide.*

2. **Description of *scale* parameter of xprof_init() in PAT is incorrect**
   The description of the *scale* parameter in the manual page of the **xprof_init()** call in the *iPSC®/860 Parallel Performance Analysis Tools Manual* refers to program locations without explaining that program locations are two-byte values.

   In addition, the description on page 3-4 of the manual should read as follows:

   The final argument, *scale*, specifies how many bytes to "bin" together. When profiling, the system actually builds a histogram of memory locations and the *scale* argument specifies how wide the histogram bins should be. The scale is interpreted as an unsigned, fixed-point fraction with the binary point at the left: 0x10000 gives a mapping of two bytes to each bin to words in the data buffer. The simplest way to explain this parameter is just to demonstrate how various values work, and then let your imagination take over:

`scale = 0x10000`	Maps each pair of bytes into separate bins.
`scale = 0x8000`	Maps four bytes into a single bin in the data buffer.
`scale = 0x4000`	Maps two instruction words (eight bytes) together into a single profiling bin.
`scale = 0x0002`	Maps all instruction words into a single bin in the data buffer, thus producing a non-interrupting core clock.

3. **Discussion of account_hook() is unclear**
   The discussion of the **account_hook()** routine in the "Accounting Software Support" section in chapter 6 of the *iPSC®/2 and iPSC®/860 System Administrator's Guide* does not make clear that **account_hook()** is a routine that is *written* by the user and *called* by the system. The file */usr/ispc/lib/commser.bin/hooks.c* is not the source code for the system **account_hook()**; it is an *example* of the kind of **account_hook()** function you might use. Most of this file is surrounded by an **#ifdef DEBUG/#endif** block; to use this example file, compile and install it with the command **make -DDEBUG.**

4. **rebootcube is no longer usable by ordinary users**
   Intel SSD strongly recommends that you remove permission for ordinary users to use the **rebootcube** command, as discussed on page 2-23. However, the documentation still describes the **rebootcube** command as a command that can be used by ordinary users.

5. **b switch in getcube (getcube -tb)**
   The intent of this switch is to allow users to easily try their programs with buffering disabled to see if it makes any performance difference. Use "**b**" in the cube type specifier to disable message buffering, which is normally enabled by default. Message buffering is required in some programs, especially simple ones which interchange messages. However, it can be demonstrated that it adds a penalty of about 20% in some kinds of ring programs, due to the extra overhead of copying a message when it would have been better to wait before sending it.

6. **Improving the performance of exchange message passing**
   The following Fortran code uses exchange message passing technique among the processes. In Release 3.3, it runs about 30% slower than Release 3.2 due to some synchronization. This type of exchange is used in applications such as 2-dimensional FFTs, matrix transpose, matrix-vector multiply, and so on.

```
                .
                .
                .
          do 10 i = 1, p-1
                .
                .
                .
               call csend(i, buf1, size, xor(me, i), pid)
               call crecv(i, buf2, size)
                .
                .
                .
10        continue
```

To improve the performance of this code, you can use force type in addition to some synchronization, as follows:

```
                    •
                    •
                    •
              do 10 i = 1, p-1
                    •
                    •
                    •
                   id = irecv(forcetype, buf2, size)
                   call csend(type, 0, 0, xor(me, i), pid)
                   call crecv(type, 0, 0)
                  call csend(forcetype, buf1, size, xor(me, i), pid)
                   call msgwait(id)
                    •
                    •
                    •
        10    continue
```

7. **Erroneous Example in IPD manual**
   The example program **gauss.f** in the *iPSC®/2 and iPSC®/860 Interactive Parallel Debugger Manual* assumes a stopping criterion that may cause a hang if the initial value of the matrix elements are zero. During the initial iterations, for some internal nodes, the boundary conditions have not yet had the time to propagate to the node. To fix this problem change the code as follows:

   Old version:

```
71      residtest = dabs(resid - residprev)/amax
72      call gdhigh(residtest,1,residtemp)
73      if( residtest .lt. eps) then
74         write(*,1003)iam,k
```

   New version:

```
71         if(amax .eq. 0.0) then
72            residtest = 1.0 + eps
73         else
74            residtest = dabs( (resid - residprev)/amax )
75         endif
76         call gdhigh(residtest,1,residtemp)
77         if( residtest .lt. eps) then
78            if(iam .eq. 0) print *,''
79            write(*,1003)iam,k
```

   The new line 78 is only for cosmetic purposes.

In addition, the **shadow()** routine must type its messages to ensure that the shadow buffer on each node contains the correct overlap values. The integer *type* is deleted and replaced with the two more specific constants *fromright* and *fromleft*. To fix this problem change the code as follows:

Old version (but after the first change):

```
170         subroutine shadow(ndim,n,totdim, iam,nbrnodes,range,a)
171     c
172     c  Shadow performs the exchange of neighbor columns into the shadow buffers
173     c
174         integer          ndim,n,totdim
175         integer          iam, nbrnodes, range
176         double precision a(ndim,range+2)
177
178         integer          length,type,leftnode,rightnode,leftid,rightid
179
180         leftnode  = iam - 1
181         rightnode = iam + 1
182         length    = (totdim+2)*8
183         type      = 100
                        .
                        .
                        .
190            rightid = irecv(type, a(1,range+2), length)
191            call csend(type, a(1,range+1), length, rightnode,0)
                        .
                        .
                        .
199            leftid = irecv(type, a(1,1), length)
200            call csend(type, a(1,2), length, leftnode,0)
                        .
                        .
                        .
207            leftid  = irecv(type, a(1,1), length)
208            rightid = irecv(type,a(1,range+2), length)
209
210            call csend(type, a(1,2), length, leftnode,0)
211            call csend(type, a(1,range+1), length, rightnode,0)
```

New version:

```
170         subroutine shadow(ndim,n,totdim, iam,nbrnodes,range,a)
171   c
172   c  Shadow performs the exchange of neighbor columns into the shadow buffers
173   c
174         integer           fromleft, fromright
175         parameter (fromleft = 200, fromright = 300)
176         integer           ndim,n,totdim
177         integer           iam, nbrnodes, range
178         double precision a(ndim,range+2)
179
180         integer           length,leftnode,rightnode,leftid,rightid
181
182         leftnode  = iam - 1
183         rightnode = iam + 1
184         length    = (totdim+2)*8
185
                              .
                              .
                              .
191           rightid = irecv(fromright, a(1,range+2), length)
192           call csend(fromleft, a(1,range+1), length, rightnode,0)
                              .
                              .
                              .
200           leftid = irecv(fromleft, a(1,1), length)
201           call csend(fromright, a(1,2), length, leftnode,0)
                              .
                              .
                              .
208           leftid  = irecv(fromleft, a(1,1), length)
209           rightid = irecv(fromright,a(1,range+2), length)
210
211           call csend(fromright, a(1,2), length, leftnode,0)
212           call csend(fromleft, a(1,range+1), length, rightnode,0)
```

NX NODE EXECUTIVE

1. **Use of a node PID number other than 0** ***(RX nodes only)***
   If you send a message to an RX node using a PID (process id) other than 0, the message will be delivered to PID 0 and no error message will be returned.

2. **Mixing long and short messages may result in messages being received out of order**
   Long and short messages are queued separately on the iPSC. (A "long" message is one that is greater than about 100 bytes in length.) As a result, if a series of long and short messages of the same type are sent from one node (or the host) to one or more other nodes (or the host), the messages may not always arrive in the same order as they were sent. For example, if the receiving node has several received long messages pending but no short ones, and a short message arrives, the node will process the short message immediately. This can cause unexpected results.

   Workaround: To enforce message order, use unique message types for messages of different sizes.

3. **Can't use csend() or crecv() within a signal handler**
   A call to **csend()** or **crecv()** within a signal handler may fail with errno **EINVAL** if there are pending **csend()**s or **crecv()**s. Note that some library calls, such as **printf()**, generate **csend()**s.

   Workaround: Keep your signal handlers extremely simple. Have them set flags and let the surrounding application handle the condition.

4. **Invalid buffer pointers**
   Invalid buffer pointers in C programs will sometimes be accepted as valid and not return an error message.

5. **Node may deadlock with file I/O**
   If a node's message buffers are filled but no receives are posted, attempts to do file I/O may fail. To avoid this failure, you must make sure that the node receives pending messages.

6. Host program hangs when sending messages to nonexistent nodes
   If a host program sends messages to nodes that are not part of the cube, the host program hangs. Use killcube to flush all the messages.

7. **cubeinfo system call and command give slightly different results**
   The **cubeinfo()** system call returns the system and domain name for the *srmname* and *hostname* fields. The **cubeinfo** command truncates the domain name, and displays just the system name.

8. **Prints originating within a handler**
   Prints originating from within a handler routine may overwrite or merge with the prints from other nodes.

9. **Sending long messages to RX nodes**
   If you send a long message to an RX node other than yourself, the send will not return until the receiving process has been loaded.

10. **Differences between RX and CX nodes when using hrecv()**
    On RX nodes, using **hrecv()** to receive a message that is too long to fit in the buffer causes the program to terminate with an error message. On CX nodes, the receive completes, the message is lost, and no error is returned.

11. **Handler routine gets different node value for global calls on RX and CX nodes**
    If the **hsend()** system call is used on an RX node with a negative destination node value, the handler receives the given value in its *node* parameter. If the same **hsend()** call is used on a CX node, the handler always receives -1 in its *node* parameter.

12. **Calling sleep() and hsend(), hrecv(), or hsendrecv() in a node program**
    Calling **sleep()** and **hsend()**, **hrecv()**, or **hsendrecv()** in a node program may cause the program to fail to return from the handler routine.

    Workaround: Don't call **sleep()**. Instead, you can use **mclock()** in a loop.

13. **pipe(), fork(), and exec() calls not fully supported**
    UNIX-compatible calls **pipe()**, **fork()**, the **exec()** family, **setuid()**, and **setgid()** for the nodes are only partially implemented. They are used by the node shell (**nsh**) and may be useful in porting programs from the UNIX environment to run under **nsh**. Using them in node programs is *strongly* discouraged.

14. **setenv TTY 'tty' in .cshrc file**
    The **TTY** environment variable is used by several cube commands and must be set before you use the cube. This variable should be set in *.cshrc*, not in *.login*. However, for the remote copy command (**rcp**) to work properly, you must redirect the standard error from the **setenv TTY 'tty'** to null as follows:

    ```
    setenv TTY 'tty' >& /dev/null
    ```

15. **archcube -c gives spurious error message if no cube is attached**
    If you do not have a current cube (for example, if you have just done a **relcube** without a following **attachcube**), the command **archcube -c** will give the error message "(host) attachcube: cubename does not exist." This message can safely be ignored.

16. **relcube -c does not reattach to original cube**
    If you are attached to a cube and you use the command **relcube -c** to release a different cube, it leaves you unattached to any cube.

17. **Rebooting the cube stops process logging**
    When the cube is rebooted, any previous process logging (started with **plogon**) is stopped.

    Workaround: Write a short C program that uses the **system()** system call to call **bootcube**, then **plogon -a**. Use this program instead of **bootcube** to boot the cube.

18. **syslog hangs when node sends many messages to the host**
    If a lot of messages are being sent to the host and have not yet been received, the host's buffers can fill up. If you attempt to use the **syslog** command or system call while the buffers are full, it can hang.

    Workaround: To unlock **syslog**, flush or receive all messages on the host.

19. **getcube returns "no more buffers" but allocates a cube**
    If a lot of messages are being sent to the host and have not yet been received, the host's buffers can fill up. If you attempt to get a cube using **getcube** while the buffers are full, it gives the error message "no more buffers," but it may actually allocate a usable cube.

20. **RX nodes cannot skip slots**
    If your hardware consists only of RX nodes, you cannot skip any slots. Any RX nodes after the first empty slot will be ignored by **bootcube**. (If your hardware contains any CX nodes, you can have empty slots.)

21. **waitcube and killcube**
    The **waitcube** and **killcube** commands occasionally fail to return. To recover, wait at least 30 seconds, press your interrupt key (default **<Del>**) to kill the command, and then do a **relcube**. If the **relcube** fails to return after 30 seconds, a **bootcube** may be required.

22. **Host pid 0 may cause cube commands to fail**
    If there is a host program in the background that has done a **setpid(0)**, some cube commands will die early with the following message:

    ```
    (host) ___: Pid already in use
    ```

    This is because **waitcube**, **startcube**, **load**, and **killcube** try to use **setpid(0)**.

23. **Nodes marked unusable**
    If you get a message saying,

    ```
    nn node(s) not responding
    ```

    (where *nn* is the number of nodes), use **bootcube** to reset the nodes. If problems persist, use the Cube Diagnostic Program, **cdp**, to check for bad node boards. The *iPSC®/2 and iPSC®/860 System Administrator's Guide* describes **cdp**.

24. **Incorrect cubeconf settings can cause red LEDs to flash continuously**
    If *cubeconf* slot fields are set to EMPTY but contain CX nodes, the red LEDs flash continuously after a **bootcube**. See the *iPSC®/2 and iPSC®/860 System Administrator's Guide* for more information on the *cubeconf* file.

25. **Host long messages**
Host programs cannot use **csend()** or **isend()** followed by **crecv()** to send long messages (greater than 100 bytes) to themselves. Also, a node program sending a message of greater than 100 bytes to the host will not complete the send operation until the host application posts a receive for the message.

Workaround: Use **irecv()**, **csend()**, **msgdone()**, or **msgwait()**.

The following example shows a code fragment from a host program:

```
id = irecv ( 1, buf, sizeof(buf));
.
.
.
csend ( 1, msg, sizeof(msg), myhost(), mypid());
msgwait(id);
```

The following host code works only if the send and receive buffers are different:

```
id = isend ( 1, msg, sizeof(msg), myhost(), mypid());
.
.
.
crecv ( 1, buf, sizeof(buf));
msgwait(id);
```

26. **Receiving process must be alive at time of send for long message buffering to work**
Long message buffering on the nodes only works if the receiving process is alive at the time of the send.

27. **Fatal error on RX nodes when a message too long for buffer received by hrecv()**
Using **hrecv()** to receive a message that's too long to fit in the buffer can cause a fatal error on RX nodes.

28. **setsyslog()**
Calling **setsyslog()** when the host program is already piping the output through **syslog** causes the host program to hang. To recover, wait 10 seconds and then press your interrupt key (default **<Del>**).

29. **newserver()**
A host program cannot call **newserver()** if its output is being piped through **syslog**. Doing so causes the host program to hang or be killed. To recover, wait 10 seconds and then press your interrupt key (default **<Del>**).

30. **killcube doesn't flush file server messages**
If a host program calls **killcube** before nodes complete file I/O, some output may be lost.

Workaround: Call **waitcube** before calling **killcube**.

31. **Cube allocated by bootcube**
When **bootcube** is executed on an SRM, a zero-node cube named "iocube" is allocated. This is necessary for all systems. It reduces the number of cubes that users can allocate from 10 to 9.

32. **Maximum number of outstanding irecv()s varies between SRM and remote host**
A maximum of 11 outstanding **irecv()**s per cube may be posted on an SRM by a host program. If you attempt to post more than this, an error message displays, and the process aborts. The maximum for the remote host is 12 outstanding **irecv()**s per cube.

33. **Profiling host applications may cause problems**
The **getcube()** and **newserver()** routines abort with the message "(host): cannot start fserver process" if the host application in which they are called is compiled using the profiling switch **-p**.

34. **Running bootcube under rsh never returns**
The command **rsh** *srmname* **bootcube** on a remote workstation appears to hang, because the **rsh** process waits for the **bootcube** process and *all* its children to finish, and some of the child processes created by **bootcube** do not finish until the next time the cube is rebooted. To use **bootcube** from a remote workstation, don't use **rsh**; log into the SRM with **rlogin**.

NODE SHELL

1. **Don't use load command under nsh**
If you use the **load** command under **nsh**, it may fail with the message "invalid node" or "pid already in use." After the "pid already in use" error, **bootcube** may be required in order to use those nodes.

Workaround: The system administrator is advised to remove read and execute permissions from the file */usr/i860/ipsc/bin/load*. Executable files on the CFS can be loaded using the **load** command on the SRM.

2. **nsh standard output can go to the wrong window**
If you are doing a **getcube** in one window and running **nsh** on a service node in another window (or using two terminals), under some circumstances the standard output of **nsh** can go to the **getcube** window. (Standard input comes from the correct window.)

Workaround: To prevent this problem, be sure the **TTY** variable is properly set in each window. To do this, add the following line to your *.cshrc* (not *.login*) file:

```
setenv TTY `tty` >& /dev/null
```

3. **nsh will not start up if IPD is running in another window**
   If you are logged into the same SRM twice with the same user ID in two different windows (or on two different terminals), and you are running the Interactive Parallel Debugger in one window, any attempt to start **nsh** on a service node in the other window fails with the error "setpid: PID already in use."

   Workaround: To prevent this problem, be sure the **TTY** variable is properly set in each window, as described in the previous item, or get a small cube and use **nsh -s** to run **nsh** on the cube instead of the service node.

4. **At least four nodes are required to run the node shell *(nsh -s on RX nodes only)***
   To run the node shell on your current cube (**nsh -s**), if your current cube is made up of RX nodes it must have at least four nodes. The node shell can be run on one service node (which is what you get if you do not specify the **-s** option) or one CX node. If you use **nsh -s** in a cube consisting of a single RX node, you cannot use any pipes (|) or non-built-in commands (such as **ls** or **cat**) in the node shell. To use pipes and non-built-in commands in the node shell, you must have additional nodes for the additional processes that result from pipes and non-built-in commands.

5. **star command can fail with "permission denied" if there is no tape in the SRM tape drive**
   The **star** command defaults to the tape device */dev/tape*, which is the tape drive on the SRM. If you do not specify an output device when you use **star** under **nsh** and there is no tape in the SRM's tape drive, you will get the error message "permission denied." (This error just means "cannot open */dev/tape*.")

6. **Changing directory names**
   The **mv** command does not work on directories, even to change the name of the directory. The only way to do this is to copy the whole directory structure to the new name and remove the old structure.

7. **Simultaneous tar commands (backgrounded) hang nsh**
   Simultaneous **tar** commands in the background may hang **nsh**. If this occurs, kill **nsh** with **<Ctrl-\>**. If you were using **nsh -s**, you may also have to use **killcube** and **relcube** to recover.

8. **star cv gives "directory checksum error"**
   If you use the **c** and **v** switches of the **star** command together, the resulting tape will give you a "directory checksum error" message when you read it with a **star x** command. Do not use these two switches in a single **star** command.

9. **cbackup and crestore now require root permissions**
   The **cbackup** and **crestore** commands have been changed so that the user must have a numeric user ID of zero (*root*) in order to run them.

CONCURRENT FILE SYSTEM™

1. **CFS can be "full" even if there is space on some volumes**
   If even one of the physical disks in the CFS is full, you will not be allowed to allocate any space on the CFS. This can lead to "disk full" errors even when there is still space left on the CFS.

   Workaround: Use **restrictvol()** to restrict your files to disks that still have space left.

2. **Allocating several small areas of disk is less efficient that allocating one large area**
   If you repeatedly increase the size of a file by a small amount, the disk space is allocated in small chunks. This can create a lot of overhead and fragmentation, causing the CFS to fill up faster. When possible, use **esize()** to pre-allocate all the disk space you expect you will need in one large chunk.

   For example, suppose the CFS has just over 200M bytes free. If you allocate 200M bytes in one **esize()** call, you'll get it. But if you allocate 100K bytes at a time, the disk might fill up after you get only 180M bytes.

3. **Incorrect error return on attempt to read from a write-only file connection**
   An attempt to read from a write-only file connection created with the **create()** or **open()** system calls will not return an error status of -1. Instead, a random positive integer value is returned. The global *errno* value will be set correctly in this error situation. This could cause the program to behave as though the read operation succeeded with invalid data in the input buffer it provided.

4. **Parity error while reading a CFS device cause a hang**
   If a parity error occurs while reading a CFS tape device, the CFS could hang, forcing the system to be rebooted.

5. **Deleting large CFS files can take a while**
   Because the information on the CFS is spread over multiple disks, it can take a long time (up to several minutes) for a very large file to be deleted.

6. **Bad blocks are not automatically mapped out**
   If a disk drive detects a media error, the driver does not automatically remap the block. If a disk block goes bad (which causes hardware-related I/O errors and possible disk hangs), you must manually reformat the disk to get rid of the bad block.

7. **lsize() returns more than what is available on the file system**
   If you attempt to increase the size of a file by more than the remaining free space on the CFS, **lsize()** returns the size you asked for rather than the space left on the file system or -1.

8. **Cannot use open() on a directory name**
   You cannot use **open()** on a directory name in CFS. Use **opendir()** instead.

9. **bootcube must be done after mkcfs for showvol to give correct results**
   After creating a new CFS with the **mkcfs** command, you must reboot the cube with the **bootcube** command. If you do not do this, the bytes free count shown by **showvol** will be incorrect.

10. **umask() works only on CFS**
    The **umask()** routine on the nodes works only on CFS. It is not supported for SRM files.

11. **Cannot modify permissions or owner of devices created by mkdev**
    There is no way to change the permissions or owner of links that are created by **mkdev**. This prevents giving restricted access to some devices.

12. **Changing disk configuration between a cbackup and crestore**
    If a disk is added between a full **cbackup** and **crestore**, the **crestore** damages the file system, but the damage isn't visible until the next **bootcube**.

13. **iowait() returns incorrect error message**
    If an error occurs when you are using **iread()** (such as reading past EOF), **iowait()** may print:

    ```
    Error 0
    ```

14. **iread()/iowait() on empty file does not give any error message**
    if **iread()** is used to read an empty file, the following **iowait()** returns without any errors (unlike **cread()**, which prints an "I/O error" message if you attempt to read from an empty file). The buffer retains the value it had before the **iread()**.

    Workaround: Use **iseof()** to detect an empty file before calling **iread()**.

15. **No way to tell how much iread() or cread() actually read**
    The **iread()** and **cread()** system calls do not return the amount of data that was actually read.

    Workaround: If you need to know how much data was read, use **read()**.

16. **lsize() may set errno incorrectly**
    lsize() sets *errno* to an incorrect value when it encounters an `insufficient space` error or an invalid `whence` value error.

17. **File date information not updated until file is closed**
    The date indicating when a file was last modified is not updated until the file is closed. Therefore, checking this date on a file that is currently open and being written to produces misleading information.

18. **File system permissions are not always handled correctly**
    In some cases, file system permissions are not handled correctly. In particular, you can not remove the files of others from your directory.

CONCURRENT FILE SYSTEM™ TAPE

1. **Tape device file must exist before using CFS tape drive**
   Refer to the **mkdev** command in the *iPSC®/2 and iPSC®/860 Programmer's Reference Manual.*

2. **Tape devices must be remade after mkcfs**
   The **mkcfs** command destroys the entire contents of the CFS, including any tape devices (such as */cfs/tape*). You must use the **mkdev** command after **mkcfs** to recreate each tape device.

3. **Don't reposition the tape head when the tape device is in fixed block mode**
   Using the **mt** command or **ioctl()** system call to reposition the tape head is not recommended when the tape device is in fixed block mode. This is because the CFS reads ahead, copying tape data to memory buffers, so the tape head position will not correspond to the position of the file pointer. If you read from the tape after repositioning the head, the resulting data will not be what was expected.

4. **Only one node at a time can access a CFS tape device**
   A tape device on CFS, such as */cfs/tape*, cannot be opened by more than one process at a time.

5. **Fortran tape control statements are not supported for tape devices**
   Tape control is not supported in Fortran. This meant that if you open a tape device with the Fortran **open** statement, Fortran tape control statements will not work. For example, the statement **rewind 1** will have no effect if unit 1 is opened as a tape device.

   Workaround: Instead of **rewind**, you can simply close and re-open the tape device.

6. **Multiple tar commands in nsh script gives tape read error**
   Using the **tar** command several times within a shell script under **nsh** can cause a "tape read error" to occur.

7. **Writing large blocks to tape in variable block mode causes I/O errors**
   When the tape drive is in variable block mode, writing a block that is larger than half the available memory on the controlling I/O node can cause an I/O error. The available memory on the I/O node is the memory size of the node minus approximately 3M bytes, so if you use 4M-byte I/O nodes, the largest block that can be written in variable block mode is approximately 0.5M bytes.

8. **CFS tape device file entry date is not updated**
   The date field in a CFS tape device file entry does not reflect the last write time. This is because it represents the tape device itself, not the tape loaded in that device.

9. **Multiple names for a tape drive treated the same**
   If there is more than one name for a tape drive, any changes to one name affect the other as well. For example, if the Volume 1 drive is named both */cfs/tape1* and */cfs/tape2*, the command **tapemode -n /cfs/tape1** sets both */cfs/tape1* and */cfs/tape2* to "no rewind."

10. **Attempting to write on a write-protected tape in fixed-block mode**
    Writing to a write-protected tape that is in fixed-block mode may not return an error. (This is because data may be buffered before writing to the tape.) A second attempt to write to a write-protected tape in fixed-block mode may hang the tape driver. To recover, use **bootcube**.

11. **Accessing an off-line tape drive**
    Attempts to access a tape drive that is off-line cause CFS to hang. If this occurs, try closing the tape door. If this does not allow CFS to continue, recover using **bootcube**.

12. **Must read same size blocks as were written**
    If you use **cread()** to read data from a tape device whose block size is set larger than the block size of the device that was used to write the data onto the tape, you see the message

    ```
    Attempt to read past end of file
    ```

    even when this is not the case.

    Workaround: Use the **tapemode** command to reduce the block size of your tape device.

13. **Using crestore with the CFS tape device in variable block mode**
    Using the node **crestore** command with the CFS tape device in variable block mode damages the CFS file system.

    Workaround: Use the **cbackup** and **crestore** commands with the CFS tape device set to a fixed block mode.

14. Certain mt commands work only in variable block mode
    The commands mt bsf and mt bsr work only when the tape device is in variable block mode, and not when it is in fixed block mode.

INTEL386™ ASSEMBLER AND LINKER

1. **Directive incorrectly used**
   In assembling data objects declared with the **.data** instruction, the Intel386 assembler creates a large temporary file on the SRM. This file can exhaust all available file space on the SRM and abort the assembly.

   Workaround: Such data objects will not create large files during assembly if they are declared using the equivalent **.bss** instruction.

2. **The Intel386 assembler is not documented**
   as is documented only as **as(1)** in the UNIX System V/386 manuals. The actual assembly instructions are not documented.

iPSC®/2 C COMPILER

The iPSC/2 C compiler, **gcc**, is located in */usr/bin* on the SRM and is linked to */usr/bin/cc*. (The UNIX-provided C compiler is in */bin/pcc*.) You use this compiler to compile and link host programs and CX node programs written in C.

The gcc compiler cannot be used to compile programs for RX nodes, and no longer accepts the **-i860** switch. The **-i386** switch is the default, and should be used to compile host programs that will run on the SRM. To compile programs for RX nodes, use the optional iPSC/860 C compiler, **icc**. See the *iPSC®/860 C Compiler Software Product Release Notes* for release note information on **icc**.

The following limitations and workarounds apply to the iPSC/2 C compiler:

1. **Deeply nested macros and -ansi switch cause internal compiler error**
   When macros are deeply nested (greater than 4 levels) and the undocumented and unsupported **-ansi** switch is used, the compiler quits with the following error message:

   ```
   Internal Compiler Error
   ```

2. **asm386 directive support**
   The C compiler is missing some support for the **asm386** directives.

3. **-C compiler switch**
   The **-C** compiler switch is not implemented.

iPSC®/2 FORTRAN COMPILER

The iPSC/2 Fortran compiler, **f77**, is located in */usr/bin* on the SRM. You use this compiler to compile and link host programs and CX node programs written in Fortran.

The **f77** compiler cannot be used to compile programs for RX nodes, and no longer accepts the **-i860** switch. The **-i386** switch is the default, and should be used to compile host programs that will run on the SRM. To compile programs for RX nodes, use the optional iPSC/860 Fortran compiler, **if77**. See the *iPSC®/860 Fortran Compiler Software Product Release Notes* for release note information on **if77**.

The following limitations and workarounds apply to the iPSC/2 Fortran compiler:

1. **Output lines from write or print statement**
   Instead of using the first character in a **write** or **print** statement as a printer control character, the character is printed. Also, an extra space is added for each continuation line in these statements. Use the **-vms** compiler switch to make the **write** and **print** statements use the first character as a printer control character and to eliminate the extra space on continuation lines.

2. **Largest binary record is 65536 values**
   The largest binary record that can be written is 2**16 values, or 65536 values.

3. Error messages for the following are not provided:

   - Extra characters on a **DO** statement
   - **DATA** statements used with a variable declared in a common block, but not in **BLOCK DATA** (VMS extensions)
   - Trigonometric function whose arguments are greater than 2**63
   - Same formal argument name used in a subroutine statement more than once
   - Logical variable used as array index
   - Integer variable used as logical (VMS extension)
   - Mix of character and any other type within a common block (VMS extension)

4. **-i2 compiler switch**
   The **-i2** compiler switch (integers default to 2 bytes) returns an incorrect error message for large integer arrays, and the compilation aborts.

5. **Invalid invocation line switches**
   Invalid compiler switches are silently ignored by the compiler.

6. **VAX/VMS extensions are not enabled by default**
   Normally, the **f77** compiler does not accept VAX/VMS extensions. Compiling with the **-vms** switch makes VAX/VMS Fortran extensions available during compilation. For more information, refer to the *VAX-11 Fortran User's Guide and Language Reference Manual*, available from Digital Equipment Corporation.

7. **Running out of node board memory**
   If you run out of node board memory while running a Fortran application, you may receive one or more of the following messages:

```
Fortran runtime error on external file "" (106): Buffer too large
Stack overflow
Memory fault
```

   This can happen when making the first write to a file, as the program tries to allocate a buffer for the file.

8. **I = MOD(J, 0) causes error message**
   The expression **I = MOD(J, 0)** in a host program causes a runtime divide-by-zero exception. This generates illegal assembler instructions, causing the assembler to issue a warning as follows:

```
WARNING: Shift count isn't in 0-31 range
```

9. **Formatted print statements on multiple nodes**
   Using Fortran formatted print statements that contain carriage return control characters ("/") on multiple nodes may cause prints from multiple nodes to merge together. That is, a message line from one node may have portions of a message from another node mixed in.

10. **Node and SRM versions of rewind**
    The node and SRM versions of the Fortran **rewind()** routine only reset the file pointer to the beginning of the file. They do not shrink the file, when appropriate, as does the standard **rewind()** routine.

11. **Character string padding in Fortran etos() and stoe() routines**
    The Fortran **etos()** routine pads a string with NULL characters (ASCII 0) instead of blanks (ASCII 32). Fortran **stoe()** expects strings to be padded with NULL characters, not blanks. This may cause the following unexpected behavior:

    - Using a string padded with blanks in **stoe()** may cause the following error:

```
stoe: Invalid size
```

      Workaround: Use strings generated by **etos()** in **stoe()**.

    - String comparisons may fail. A string padded with NULLs will not match a string padded with blanks.

12. **New names for Fortran unnamed and scratch files**
The implementation of Fortran unnamed files and scratch files has changed from the previous release.

    - When you open a file without specifying a name in the **open** statement, the file is created in your current working directory on the host system with the name *fort.ddd*, where *ddd* is the Fortran unit number in decimal.

      Because the naming convention for unnamed files is based on the unit number alone, if multiple nodes open unnamed files on the same unit number, they will all be connected to the same file (each with its own file pointer).

    - When you open a scratch file (**status='scratch'**), the file is created with the name *FTNpppppppp.uu*, where *pppppppp* is the process number that opened the file in hexadecimal and *uu* is the Fortran unit number in decimal. If there is a CFS file system, the scratch file is created in */cfs* (of the value of the **CFS_MOUNT** environment variable); if there is no CFS, the file is created in */usr/tmp* on the host file system.

      Because the naming convention for scratch files is based on the process number and unit number, if multiple nodes open scratch files on the same unit number, each has its own file.

INTERACTIVE PARALLEL DEBUGGER

1. **No dimension information is available for some array arguments**
The iPSC/2 Fortran compiler does not provide symbol table information for assumed-size array arguments (such as **array(*)**) and adjustable array arguments (such as **array(*n*)**) concerning the size of the dimension. Therefore, when debugging CX node programs written in Fortran, a size of 1 is assumed with a lower bound of 1. A warning message is displayed whenever the debugger detects this.

   With the iPSC/860 Fortran compiler, only assumed-size arrays (**array(*)**) have no dimension size information available. Adjustable arrays (**array(*n*)**) are sized correctly.

   Workaround: Use the **type** command to determine what the debugger thinks the size of an array is for a particular scope. Use a count specification to look beyond the end of the assumed size of the array and thus get access to all of the elements.

2. **Incorrect symbol table information is generated for some adjustable array arguments**
The iPSC/860 Fortran compiler provides incorrect symbol table information for array arguments that have adjustable dimensions and only one bound that is a variable (such as **array(5:*n*)**). Using **display** or **assign** will have unpredictable results.

   Workaround: Use the **type** command to determine what the debugger thinks the dimension and lower bound are.

3. **Incorrect results when accessing large arrays in CX node programs**
The iPSC/2 compilers generate incorrect symbol table information for arrays larger than 64K bytes. If the array has no dimensions with more than 64K elements, but the array size is greater than 64K bytes, a warning is printed and the array size is calculated correctly. If the array has any dimensions with more than 64K elements, the condition may not be detected and any access to the array may be incorrect.

   Workaround: Don't trust results of accessing any array in a CX node program declared with more than 64K elements in any dimension.

4. **Killing program being debugged can abort debugger**
If a program under debug is killed via **killproc()** or **killcube()**, any subsequent attempt to debug that process will cause the debugger to abort.

   Workaround: Either do not call **killproc()** or **killcube()**, or set breakpoints on these routines to intercept any calls to them.

5. **IPD errors can result from omission of .file directives in assembly language routines**
If your program contains routines written in assembly language, you may see spurious warning messages about "multiple .text records" if the assembly-language routines do not contain **.file** directives identifying the name of the source file. The Basic Math Library (*libkmath.a*) is known to have this problem.

   Workaround: Add **.file** directives to all assembly-language routines, giving the name of the source file.

6. **Display and assignment of addresses in CX node programs is inconsistent**
All application addresses on CX nodes are offset in the linear address space by 0x40000000 by NX. The debugger does not perform this translation consistently. When displaying a C language pointer variable, the offset is not included (e.g., 0x004046f4) while displaying the address of a C language variable using an ampersand (&) the offset is included (e.g., 0x404046f4).

   Workaround: When displaying the address of a variable using the ampersand syntax, subtract 0x40000000 from the value displayed. When assigning a pointer value, do not add 0x40000000.

7. **Problems displaying and assigning C character strings**
The debugger has the following problems displaying and assigning character strings in C programs:

   - Using **-string** when displaying a pointer to a character string results in the address of the character string being displayed rather than the string itself.

     Workaround: Display the pointer using a subscript (such as **ptr[0]**), include a count specification indicating the number of characters in the string to display, and omit the **-string** switch.

- When **-string** is used to display an array of pointers to strings, only the first string is displayed, the subscript value displayed is incorrect, and any count specification is ignored.

  Workaround: Display each element individually.

- Assigning a string value to a pointer to a character string works incorrectly.

  Workaround: Access the pointer as an array and assign each element individually.

- The display command's **-string** switch is only significant when displaying C character arrays or pointers to characters. It is ignored otherwise.

- Assigning a character to a C **char** array has different results from assigning a string.

  If a character value (denoted by single quotes) is assigned to a string, the entire string (or *count* elements) will be filled with that character. If a string value (denoted by double quotes) is assigned, a single copy of the null terminated string is assigned, and any count specification is ignored.

8. **Limitations in display command**
   The **display** command has the following limitations:

   - The **-complex** and **-dcomplex** switches are unsupported.

   - The **-alpha** switch incorrectly converts the contents of Fortran symbols in RX node programs to a boolean value rather than an alphanumeric character.

   - Radix conversion is limited. The switches **-hexadecimal**, **-octal**, and **-decimal** are recognized only for integer variables.

9. **Can't display C variables of type unsigned char**
   Display of C variables whose type is **unsigned char** is unsupported.

   Workaround: Get the address of the variable using the ampersand syntax (&) and display the address. This will result in a hex dump with alphanumeric equivalent values on the right.

10. **type, assign, and display commands don't support C bitfields**
    If a structure contains bitfield elements, any attempt to use the **type**, **display**, or **assign** command on the bitfield will result in a "member of struct/union not found" error.

11. **Data breakpoints on RX node programs may corrupt floating point results register**
    If a process hits a data breakpoint on a core instruction while a floating point operation is in progress, the floating point result register may be invalid. However, the program will run correctly.

    Workaround: Don't trust the contents of the floating point result register when stopped at a data breakpoint.

12. **Can't set breakpoints on non-executable lines**
Breakpoints that are set on non-executable lines are actually set on the next executable line in the program. The display of breakpoint information will show the non-executable line, but if the process stops at the breakpoint it will stop at the executable line.

Workaround: Set breakpoints only on executable lines.

13. **Problems with set and alias**
The debugger has the following problems with the **set** and **alias** commands:

- Using a command line **set** variable twice on the same command line will cause the debugger to abort.
- Using a recursive alias will cause the debugger to abort.
- Using a command line **set** variable in an **alias** command will cause an internal debugger error.

Workaround: Do not use **set** or **alias** in the ways described above.

14. **Can't use break -after on a procedure**
The **break** command's **-after** switch is ignored when setting a breakpoint on a procedure.

Workaround: Do not use the **-after** switch when setting a breakpoint on a procedure.

15. **Extra lines in log files for commands containing semicolons**
Command lines that contain semicolons (;) are echoed to log files an extra time for every semicolon.

Workaround: Ignore extra lines in log file.

16. **A space may not precede the command ! or ?**
A syntax error is generated if a space character precedes either the ! or ? commands.

COMPILER LIBRARIES

1. **Standard I/O library may have trouble printing very large numbers**
The standard I/O library may produce incorrect results when printing very large numbers. The only known example is 2.0**53, which does not print as an integer.

2. **Exceptions enabled using fpsetmask(3C) may not be reported correctly on RX nodes**
If you use the fpsetmask(3C) function to enable ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic exceptions, any exceptions that occur may not be reported correctly.

3. **Spurious warnings from time.h header file**
An erroneous **typedef** in the header file *time.h* causes warning messages about "more than one type specified" and "useless typedef declaration" when programs that use this file are compiled with the iPSC/860 compilers. These warnings can safely be ignored.

RX VECTOR LIBRARY

RX vector library is not optimized
The vector library for the iPSC/860 system is included in the software to provide a migration path from the iPSC/2 vector product to the iPSC/860 system. These libraries are not currently optimized to run at their peak speed on the iPSC/860 system.

BASIC MATH LIBRARY (libkmath.a)

1. **IPD errors when debugging programs using Basic Math Library**
When using the Interactive Parallel Debugger (IPD) to debug a program that uses the Basic Math Library, you may encounter the following problems:

   - Warning messages about multiple .text records in a file
   - Inability to set breakpoints within Basic Math Library subroutines written in assembly language
   - Inability to trace the call chain to Basic Math Library subroutines written in assembly language.

2. **Two problems with IZAMAX**
The **IZAMAX** routine in the Basic Math Library has the following problems:

   - If called with a length of zero or less, **IZAMAX** returns 1 instead of 0.
   - If called with an increment of other than one, **IZAMAX** returns an incorrect answer.

REMOTE HOST

1. Compile programs for RX nodes with icc and if77, not rcc and rf77
Currently, the rcc and rf77 commands can only be used to compile programs for the SRM and CX nodes. You can now cross-compile programs for RX nodes using the icc and if77 commands, which run locally on your Sun-3 or Sun-4 workstation or on the SRM. If your remote host is not a Sun-3 or Sun-4 workstation, you must currently rlogin to the SRM to compile programs for RX nodes.

2. **relcube may return before cube is completely released**
   If you release a cube with the **relcube** command, and then immediately try to get the same cube or an overlapping cube with the **getcube** command, you will sometimes get the message:

   ```
   (host) getcube: No SRM that matched your request was found.
   ```

   Workaround: Wait a few seconds after a **relcube** before doing a **getcube**.

3. **Size of environment for node programs from remote host limited to 2K bytes**
   When loading node programs from a remote host, the space available for environment variables is limited to a total of approximately 2K bytes. If the environment is larger than this, only the first 2K bytes of the environment is copied to the node program.

   Workaround: Use the command

   env **-** *environment* ***load*** *command*

   to specify the environment for the command. For example:

   ```
   env - HOME=/cfs/me TZ=PST8PDT load myprog
   ```

4. **The load command on a remote host fails to load executables that reside on CFS**
   Attempting to load a node program from a file on the CFS using the **load** command on a remote host fails with the message "no such file or directory."

   Workaround: Use the **load** command under **nsh** or on the SRM instead.

5. **Incorrect error message from remote host**
   The error message "Need 'ripscduser' entry in passwd file" which may appear on the SRM console should read "Need 'ripscd' user entry in passwd file."

   This error message appears when a remote user executing remote language commands (such as **rcc**, **rf77**, and **rld**) does not have an entry in the SRM's */etc/passwd* file. If the remote language commands cannot find an entry in */etc/passwd* for the remote user, the server (*ripscd*) will try to use the */etc/passwd* entry for the user *ripscd*. This entry will be used to set the user and group IDs of the remote language command process, thus controlling its file system access.

   Workaround: Add an entry for the pseudo-user *ripscd* to the */etc/passwd* file, or add entries for all remote users who use remote language commands. (Intel SSD recommends that every remote host user have an entry in the */etc/passwd* file. To prevent these users from logging into the SRM, you can set their login shell to */dev/null*.)

6. **Node program and directory must be world-executable on NFS-mounted file systems**
   For the **load** command or **load()** system call to load a file on the remote host, both the file and the directory containing it must be executable by "other" if the file system containing the file is NFS-mounted.

7. **Cube ownership**
   Cubes that you own are not automatically released when you log out from the remote workstation. You must use **relcube** to release the cube.

8. **Maximum of six cube partitions per remote host**
   Some remote host operating systems (for example, Sun OS 3.5) support a maximum of only six cube partitions per remote host. Attempting to allocate a seventh cube may cause the remote host to hang. To recover, reboot the remote host.

9. **setenv TTY 'tty' on Sun workstations**
   The **TTY** environment variable is used by several cube commands and must be set before you use the cube. Because each newly created window on a Sun workstation inherits its parent's shell variables, you should set **TTY** in *.cshrc* and not in *.login*. However, for the remote copy command (**rcp**) to work properly, you must redirect the standard error from the **setenv TTY 'tty'** to null as follows:

   ```
   setenv TTY 'tty' >& /dev/null
   ```

10. **cubeinfo's SRM field does not contain complete name**
    On a remote host, the **cubeinfo** command's *srmname* field contains the SRM name specified by **getcube -h**, which may not be the complete name.

11. **FORCE TYPE range messages**
    Message types in the FORCE TYPE range (types that are greater than 40000000 hexadecimal), sent from a node to a remote host, must be received using a **csendrecv()** or **isendrecv()** call in the host program. Forced messages sent from a node to a **crecv()** or **irecv()** on a remote host will be lost. Force messages *can* be sent from a host program to a host program, a node program to a node program, or from a remote host program to a node program and received using **crecv()** or **irecv()**.

12. **Merged error messages**
    On a remote host, iPSC/860 error messages may be merged with prints from user applications.

13. **Invalid buffer pointers**
    Invalid buffer pointers in a C host program are sometimes accepted as valid and the system does not return an error message. This may cause unpredictable behavior.

14. **SSD syslog command clashes with SunOS syslog command**
    The SSD Remote Host command **syslog** has the same name as the SunOS command **syslog**. This can lead to confusion about which **syslog** command is being used.

    Workaround: If you want to be sure you are using the SSD **syslog** command, but don't want to overwrite the SunOS **syslog** command: create a special directory for SSD commands such as */usr/ipsc/bin*, install the SSD **syslog** command there, and put that directory ahead of */usr/bin* and */usr/ucb* in your search path.

NETWORK QUEUEING SYSTEM (NQS)

1. **Devices are not supported in this release**
   Device support has not been implemented for this release of the NQS software. The Network Queueing System Manual refers to devices, device queues, and print queues in numerous areas. While the underlying NQS software may support devices and the queue structures associated with devices, Intel has not evaluated this area of the software and does not support this feature at this time.

2. **qpr command does not work**
   Since device queues are not supported in this release, the **qpr** command does not work.

3. **Times displayed for qsub -a may be inaccurate during Daylight Savings Time**
   The times displayed by the command **qsub -a** may be an hour off if Daylight Savings Time is currently in effect. After submitting a job, verify the time to run by using the **qmgr** command **show long queue.**

4. **qsub -s should always be used with -x**
   Always use the **-x** option (export environment variables) with the **-s** option (use alternate shell) to the **qsub** command. Omitting the **-x** option will cause the default environment on the remote machine to be used, which will probably not give the desired results with the new shell.

5. **qmgr command "set mail *userid*" does not work**
   The **qmgr** command **set mail**, which is supposed to change the user to whom mail is sent when a job is executed, does not work. For example, if user *chris* uses the command **set mail scott**, user *scott* does not receive the mail. Instead, *chris* receives it.

6. **Invalid cubetype for remote job submission on pipe queue fails silently**
   If a job in a pipe queue asks for a cubetype that is not valid or not installed, and the user who submitted the job is on a different system from the intended batch queue, the job is deleted but the user doesn't get any indication of why it was deleted.

   Workaround: Always submit pipe queue jobs on the same system as the intended batch queue.

iPSC®/2 SIMULATOR

1. **Process creation**
   The simulator creates one UNIX/XENIX process for every simulated node or host process. Therefore, the size of a simulation is limited by the maximum number of processes allowed by the operating system being used. XENIX software allows 14 processes to be created; and UNIX 4.2 BSD, UNIX 5.2 ATT, and UNIX 5.3 ATT allow 23 processes to be created. Limit the cube's dimension or the number of processes per node to conform to these limitations.

2. **CFS calls and three node calls not supported**
   The CFS calls (such as **cread()**, **cwrite()**, **iowrite()**, **iodone()**, and **iomode()**) are not supported by the simulator. The NX **handler()**, **dclock()**, and **hwclock()** calls are not supported either.

3. **Clock calls and timing**
   Note that all timing uses a common time base. Because the simulated processes are executed in sequential timeslices by UNIX software, the values from the clock are different from those obtained on the iPSC/860 system.

4. **Remote host and cube sharing**
   Remote host and cube sharing functions are not supported by the simulator. Related calls (such as **getcube()**) should be commented out before you compile.

5. **System Resource Manager (SRM) programs**
   System Resource Manager programs are started within the simulator. Thus, the command line cannot be used to supply arguments to the SRM programs or redirect their standard input or output.

6. **File descriptors**
   File descriptors 9, 10, 11, and 12 are reserved for the simulator.

7. **Signals**
   Signal number 28 in the UNIX 4.2 BSD environment; and signal number 16 on XENIX, UNIX 5.2 ATT, and UNIX 5.3 ATT are reserved for the simulator.

8. **Interchanging calls**
   The simulator accepts all host and node calls in both host and node programs, whereas the iPSC system does not. Therefore, it is possible to write simulator programs that will not run on the iPSC system. Refer to the *iPSC®/2 and iPSC®/860 Programmer's Reference Manual* for more information on the calls supported on the host and nodes.

UNIX SOFTWARE

1. **Using the right shell for a script under csh**
   If you use the C-shell (**csh**):

   - To make a script execute under **csh**, make the first two characters of the script a pound sign and an exclamation mark (#!).
   - To make a script execute under **sh**, make the first character of the script a colon (:).

   If the first character is neither a pound sign nor a colon, the script executes under **sh**.

2. Scripts in sh always run under sh
   If you use the Bourne shell (sh), all scripts execute under sh. To execute a script under csh, use the command *csh -c script*.

3. **unsetenv and ulimit not supported**
   The **unsetenv** command is not supported in UNIX System V version 3.2. The **unsetenv** and **ulimit** built-in commands are not included in the **csh**.

4. **Duplicate symbol defined in include files**
   The symbol `SYMESZ` is defined in both */usr/include/syms.h* and */usr/include/ldfcn.h*. Including both of these files in your application causes a compiler warning.

TCP/IP ON THE SRM

1. **telnet problem**
   telnet will not work from a shell layer (**shl(1)**).

CUBE DIAGNOSTIC PROGRAM (CDP)

1. **SAT help descriptions scroll off screen**
   Some help descriptions in the System Acceptance Test scroll off the top of the screen without pausing.

   Workaround: Use `<Ctrl-S>/<Ctrl-Q>` to control scrolling.

2. **SAT cannot test mixed cubes**
   The System Acceptance Test cannot currently test a cube that contains a mixture of CX and RX nodes.

THE APPLICATION BINARY INTERFACE 4

INTRODUCTION

This chapter contains information pertinent to i860 software developers, especially those writing in assembly language.

Intel Supercomputer Systems Division uses the register usage and calling conventions and the data alignment rules of the Application Binary Interface (ABI). The ABI is a hardware and software standard that is intended to allow programs to run without recompiling or relinking on any machine adhering completely to the standard. The ABI standard replaces the previous convention, which is referred to as the PRM (Programmer's Reference Manual) compliant binary convention.

To ensure ABI compatibility among the various i860 environments, a set of binary conventions have been established. This standard defines calling and data alignment conventions.

Existing i860 PRM-compliant NX/2 application binaries need to be converted to the i860 ABI. This chapter describes the differences between the i860 PRM and the i860 ABI and the modifications that are required to convert a PRM compliant application into an ABI compliant application.

NOTE

Chapter 8 of the *i860™ 64-Bit Microprocessor Programmer's Reference Manual* (240239-002), published in 1989, documents the PRM-compliant calling and data alignment conventions. New manual releases, starting with 240239-003, published in 1990, reflect the new ABI calling and data alignment conventions.

CALLING CONVENTIONS

The calling conventions consist of the register allocation and parameter passing conventions. The difference between PRM and ABI are described in Tables 4-1 and 4-2.

Table 4-1. Integer Register Allocation Changes

	PRM Allocation	ABI Allocation
Parameters and Temporaries	r16-r28	r16-r27
Return values	r16	r16-r17
Temporaries	r28-r30	r30-r31, r28 and r29 (if they are not used for argument or environment pointers)
Memory argument area pointer	--	r28
Environment pointer	--	r29

Table 4-2. Floating Point Register Allocation Changes

	PRM Allocation	ABI Allocation
Local values	f2-f15	f2-f7
Parameters and Temporaries	f16-f27	f8-f15
Return value	f16-f17	f8-f11
Temporaries	f28-f31	f16-f31

There are two types of parameter-passing changes.

- The first change occurs when there are more parameters than registers.
- The second change occurs when there are mixed mode parameters.

If there are more parameters than the number of parameter registers, the extra parameters go on the top of the stack in the PRM register allocation scheme. In the ABI allocation scheme, the extra arguments are placed in memory and the integer register r28 (called the memory argument pointer) is set to point to this argument area in memory. This memory may either be statically allocated or allocated off the stack.

The other type of parameter passing change involves the passing of mixed mode parameters. In PRM, the passing of mixed integer and floating point parameters in registers is treated as a special case. If parameter number N is an integer parameter, it goes into the integer register 16 + N, and the floating point register pair at 16 +2N is not used for passing parameters. If parameter number M is a floating point value, it goes into the floating point register pair at 16 + 2M, and the integer register 16 + M is not used for passing parameters. In ABI, no such register skipping is required for mixed integer and floating point parameter passing. The integer and floating point registers are treated independently. For example, if Parm1 and Parm2 are integer variables and Parm2 and Parm4 are floating point variables, the register allocations are as shown in Table 4-3.

Table 4-3. Mixed Mode Parameter Allocation

PRM			
Integer Registers		**Floating Point Registers**	
Parm1	r16		f16, f17
	r17	Parm2	f18, f19
Parm3	r18		f20, f21
	r19	Parm4	f22, f23
ABI			
Integer Registers		**Floating Point Registers**	
Parm1	r16	Parm2	f8, f9
Parm3	r17	Parm4	f10, f11
	r18		f12, f13
	r19		f14, f15

DATA ALIGNMENT CONVENTIONS

The alignment requirements of the PRM and ABI conventions differ for structures, unions, and Fortran common blocks. The PRM requires structures, unions, and common blocks to be aligned on16-byte boundaries. But in ABI, the most restrictive alignment from among the members of structures and unions is used as the alignment. The most restrictive alignment for ABI is eight bytes, rather than 16 bytes. However, alignment on 16-byte boundaries is still recommended because it usually results in higher performance.

CONVERSION

The calling and data alignment changes need to be made to the source code of existing assembly language applications. The code must then be reassembled using the Intel assembler.

High-level language applications must be recompiled and relinked with the new compiler and linker.

USER INFORMATION 5

INTRODUCTION

This chapter contains information on node memory usage.

NODE MEMORY USAGE

Table 5-1 shows the amount of free memory in each node when no application processes are loaded. It also shows the size of the operating system and short message buffers. Short message buffers are used to hold system control messages and application messages up to 100 bytes long.

Table 5-1. Memory Usage Per Node in Release 3.3

Node Type	Number of Message Buffers	Free Memory		NX/2 Code		NX/2 Data		Message Buffer Size		Total Node Memory
8M RX	1024	7,794,688	+	138,336	+	316,320	+	139,264	=	8,388,608
16M RX	2048	16,044,032	+	138,336	+	316,320	+	278,528	=	16,777,216
32M RX	4096	32,542,720	+	138,336	+	316,320	+	557,056	=	33,554,432
64M RX	8192	65,540,096	+	138,336	+	316,320	+	1,114,112	=	67,108,864
1M CX	512	610,284	+	122,844	+	213,048	+	102,400	=	1,048,576
4M CX	634	3,723,244	+	122,844	+	221,240	+	126,976	=	4,194,304
8M CX	1146	7,806,976	+	122,844	+	229,412	+	229,376	=	8,388,608
16M CX	2170	15,970,304	+	122,844	+	249,892	+	434,176	=	16,777,216

To calculate the size of the application process, use the UNIX size command to obtain code, data, and BSS sizes. Round the code size up to a multiple of 4,096, then add 4,096 for each 4,194,304 or less of the result. Add the data and BSS sizes, round the result up to a multiple of 4,096, and again add 4,096 for each 4,194,304 or less of that result. Add an additional 12,288 for the stack and other overhead to the other subtotals to obtain the total memory required. For example:

```
% size node
node: 101272 + 46912 + 521280 = 669464
```

Calculation:

	Size (actual)	Size (to next 4,096)		Overhead		Total
Code	101,272	102,400	+	4,096	=	106,496
Data + BSS	568,192	569,344	+	4,096	=	573,430
Stack and other				12,288	=	12,288
Grand Total						692,224

This program would require a 4M-byte or larger CX node or an 8M-byte or larger RX node.