INTEL SCIENTIFIC COMPUTERS
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
(503) 629-7600

December 29, 1989

Dear iPSC®/2 Customer:

This package contains your iPSC®/2 system software, Release 3.1. With this upgrade, you now have the latest version of the NX/2 operating system. For former Release 2.4 users, this upgrade is equivalent to two updates: the update from Release 2.4 to 3.0, and then the update from 3.0 to 3.1.

Before using your iPSC/2 System:

- **Read this letter completely.**
- **Verify the contents of this package.**
- **Read the *Release 3.1 Product Release Notes*.**

Release 3.0 represented a major milestone in the evolution of system software for the iPSC/2 system. It introduced substantial new product capability (including the Concurrent I/O Facility) and a new version of the UNIX operating system for the System Resource Manager (SRM). Release 3.0 also included an enhanced Ethernet card for your SRM. Release 3.1 added tape support to the Concurrent File System, enhancements to the vector software, and improvements to DECON, the concurrent debugger.

The Concurrent File System makes the Concurrent I/O Facility of the iPSC/2 system as easy to use as a conventional I/O system. With CFS, the system's array of disks appears as a single virtual disk that can be simultaneously accessed by processes on compute nodes. In short, the benefits of parallel disks — file storage in excess of 40G bytes and high-speed parallel access — are available without any special programming.

Release 3.0 introduced the latest version of UNIX software on the SRM -- Release 3.2, Version 2.1 of System V. The combination of the new UNIX release and the new iPSC/2 system software provides streams and sockets, international character sets, and improved SRM security. Other UNIX functions have been improved, including tape and disk support, a superior bad block mechanism for disks, and a way to run DOS programs.

Release 3.1 improves the DECON concurrent debugger. Systems with the Concurrent I/O Facility now support 9-track reel-to-reel tape drives.

New releases of the VAST2 vectorizer and the VX vector options are also available. With this release, VecLib contains over 70 new routines, new asynchronous calls, and support for C programs.

Also newly available is iPSC/2 Ada and VMSLink software. iPSC/2 Ada offers all the tools of Verdix Corporation's VADS (Verdix Ada Development System) tailored to the iPSC/2 system and VMSLink software. VMSLink lets you run computationally intensive portions of your applications on the iPSC/2 system while keeping the rest of your applications under the VMS operating system.

Package Contents

Your iPSC/2 software package is shipped in two boxes. Please verify that they include the following items:

Media	Documentation
iPSC/2 System Software R3.1 cartridge tape	iPSC/2 User's Reference Manual Set (Note that you receive only those manuals that have changed since Release 2.4).
Nameproc Update diskette	iPSC/2 System Administrator's Manual Set (Note that you receive only those manuals that have changed since Release 2.4)
UNIX V R3.2 V2.1 cartridge tape	UNIX Literature Set (including UNIX Quick Reference Guide)
UNIX boot diskette	
UNIX Remote Terminal Package diskette	
TCP/IP diskettes (four)	TCP/IP Manuals
Ethernet drivers diskette	

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

What is in This Release?

This is the latest version of iPSC/2 system software. Support has been added for the features described below.

- **Improved DECON debugger.** In addition to increased reliability, DECON has more support for Fortran programs. You no longer need to identify the COMMON block when displaying variable, and you can display any element in an EQUIVALENCE.
- **UNIX Profiler.** The Concurrent Workbench software tool set allows node programs to generate profiling data for use by *prof*, the standard UNIX profiling tool. The profiler gives Fortran and C users an easy way to evaluate their applications for load balancing and efficiency and thus to determine which parts of node programs might be improved to yield the greatest overall performance.
- **Enhanced Concurrent File System™ (CFS).** This software requires the Concurrent I/O Facility, an array of disks that appears as a single logical entity. iPSC/2 systems with the Concurrent I/O Facility now support nine-track reel-to-reel tape drives. Files in the Concurrent File System are no longer restricted to 2G bytes, new system calls are provided for extended files, and a file checker has been added.
- **Enhanced library for the Weitek-based SX scalar processor.** For certain math functions, performance increases two to four times.
- **Process Logging Function.** The iPSC/2 system administrator can log the allocation and deallocation of cubes and the loading and completion of node processes.
- **gcolx().** Under certain conditions, you can use **gcolx()** instead of **gcol()** for better performance. These calls construct a vector by collecting contributions from each node.

What Software Options are Available?

- **VX Vector Processor Software Release 1.1.** This includes over 70 new VecLib routines, including 24 complex vector operations, scatter/gather routines, and memory allocation routines. Overlapped vector and node execution is now supported with asynchronous VecLib calls for up to double the performance. C programs can now make VecLib calls using C conventions.
- **Enhanced VAST vectorizer.** VAST2 Release 1.2 offers better loop optimizations and asynchronous call support. Many calls have been optimized, doubling performance in some cases.
- **iPSC/2 Ada.** The iPSC/2 Ada development environment is based on Verdix Corporation's VADS (Verdix Ada Development System). The iPSC/2 Ada environment offers all the VADS 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.
- **iPSC/2 Lisp.** With this software installed on your iPSC/2 Concurrent Supercomputer, you can develop and run Lisp applications on your system.

- **iPSC/2 VMSLink.** VMSLink software lets you run computationally intensive portions of your applications on the iPSC/2 system while keeping the rest of your applications under the VMS operating system. Application users gain the best of both environments: the familiar user environment of the VAX/VMS minicomputer or workstation, and the supercomputing power of the iPSC/2 system.
- **Source Products.** Source code for the NX/2 operating system and the Concurrent File System is available.
- **Network File System (NFS).** With NFS installed on your iPSC/2 system, you can access file systems on other Ethernet nodes from your workstation as if they contained local files. Programs running on iPSC/2 nodes can also use NFS to access file systems on Ethernet nodes.

New Documentation

In addition to these new features, the documentation for the iPSC/2 System has been revised and expanded to be more readable and usable.

Restrictions and Limitations of Release 3.1

Every effort has been taken to assure the quality of this release, but at shipping time we are aware of a few problems. Please refer to the *Product Release Notes, Release 3.1* for known restrictions and limitations and their workarounds .

Bugs Fixed by Release 3.0

Bugs have been fixed in the iPSC/2 system software. These fixed bugs include the following:

1. **Opening files**
   A node process may now open more than twenty files at a time.
2. **Node no longer hangs with file I/O**
   If a node's message buffers are filled, but no receives are posted, attempts to do file I/O no longer fail.
3. **printf statement after an irecv no longer hangs**
   If a node process posts an **irecv()** using the type -1, the next print statement no longer causes the process to hang.

4. **killcube message no longer received by next executing process**
   If a node program kills itself by calling **killcube**, the next program executed on the node with the same pid will no longer receive the system response message for that **killcube** when it posts a receive with the type -1.

5. ***cubeconf* is now updated to show suspect boards**
   When **bootcube** displays an error message indicating that a board is suspected of being bad, the *cubeconf* slot information now reads "FAILED."

6. **SX and VX cube types**
   You may now mix upper and lower case letters when you specify SX and VX cube types in a **getcube** call or command.

7. **Unsupported flags in as assembler**
   The **-R** and **-m** flags are now supported by the assembler. The **-Y** flag still causes an error.

8. **Linking with switches -vec -vx, -vec -sx, -vec -sx -vx, -vecdb -vx, or -vecdb -vx -sx.**
   Routines from *libf* are now linked in when using switches **-vec -vx, -vec -sx, -vec -sx -vx, -vecdb -vx,** or **-vecdb -vx -sx.**

9. **Bad Block Mapper**
   The bad block mapper allows entry of a maximum of 256 bad blocks. Reloading the UNIX operating system no longer requires that the bad block information be typed in.

10. **head command now provided**
    There is now a **head** command on the System V/386 UNIX operating system.

11. **Path for mail set correctly**
    In the default .cshrc file, the path for **mail** is now set correctly to */usr/mail/$LOGNAME.*

12. **setsyslog call**
    setsyslog now works as expected.

Bugs Fixed by Release 3.1

Bugs have been fixed in the iPSC/2 system software. Refer to the Release 3.1 Product Release Notes of a complete list of fixed bugs. The following list contains the most important bug fixes.

1. **Maximum SRM shell file size**
   The node shell now executes the entire shell file, even if it is located on the SRM.

2. **Using CFS commands/routines in a non-CFS system may hang your application.**
   If you have a non-CFS system that was booted as a non-CFS system (**bootcube -o**), using CFS commands and routines no longer hangs the system.

3. **killcube fails if waitcube is in the background**
   killcube no longer fails if there is a **waitcube** in the background.

4. **Outstanding crecv() or msgwait() with hrecv()**
   An outstanding **crecv()** or **msgwait()** no longer blocks **hrecv()**.

5. **Multiple getcube calls**
   Multiple **getcube()** calls in a host program now leaves you attached only to the last cube allocated.

6. **First getcube may fail on a remote host**
   The first **getcube** after a **bootcube** no longer fails.

7. **Timeout table overflow**
   When you run **cdp** for extended periods of time, the UNIX software no longer produces a warning message and hangs the SRM.

8. ***TERM* variable in *.login***
   The *TERM* variable is now set correctly when you remote login to an SRM.

Installation

Your Intel Service Representative will install your updated iPSC/2 system. If you wish to reinstall all or any part of your system, refer to the installation instructions in the *iPSC®/2 System Administrator's Guide*. This manual has separate sections describing how to install the UNIX operating system, the iPSC/2 system software, and the TCP/IP software.

NOTE

Adding or removing any boards or components from your iPSC/2 system can damage the system and may invalidate your warranty. Please contact Intel Scientific Computers Customer Support for assistance in answering your questions.

Remote Host Software

The remote host software allows you to use workstations on the Ethernet in addition to the SRM. For information about installing remote host software, refer to the *iPSC®/2 System Administrator's Guide*. Refer to the *iPSC®/2 User's Guide* for information on using remote host.

Comments and Assistance

Intel Scientific Computers 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/2 system.

1-503-629-7777 (Customer Support Hotline in Oregon)
1-800-421-CUBE (Hotline outside of Oregon)
(44) 793 696 619 (in England)
Your Local Intel Sales Office (in Europe)
support@isc.intel.com (csnet address)

Sincerely,

Steve Cannon

Product Marketing Manager
Intel Scientific Computers

Documentation Included With iPSC®/2 Release 3.1

Included in this package you will find the documentation for the iPSC/2 system. In addition to the manuals shown as "Revised," you also receive three 2.5-inch binders and tabs for the manual set.

iPSC®/2 User's Reference Manual Set - Volume 1 (2-inch binder)

iPSC®/2 DECON User's Guide	Revised
iPSC®/2 Simulator Manual	Revised
iPSC®/2 User's Guide	Revised
iPSC®/2 VAST2 User's Guide	Revised
iPSC®/2 VX User's Guide	Revised

iPSC®/2 User's Reference Manual Set - Volume 2 (2.5-inch binder)

iPSC®/2 Programmer's Reference Manual	Revised

iPSC®/2 User's Reference Manual Set - Volume 3 (2-inch binder)

iPSC®/2 Ada Programmer's Reference Manual	New
iPSC®/2 Lisp Programmer's Reference Manual	
iPSC®/2 VMSLink Programmer's Reference Manual	New

iPSC®/2 User's Reference Manual Set - Volume 4 (2.5-inch binder)

iPSC®/2 C Language Reference Manual	Revised
iPSC®/2 Fortran Language Reference Manual	Revised
iPSC®/2 Lisp Language Reference Manual	
iPSC®/2 Lisp Language Reference Manual Change Notice	New

iPSC®/2 User's Reference Manual Set - Volume 5 (2.5-inch binder)

iPSC®/2 Ada Program Development Guide	New

iPSC®/2 Quick Reference Guides

iPSC®/2 C Commands and Routines Quick Reference Guide	
iPSC®/2 Fortran Commands and Routines Quick Reference Guide	Revised
iPSC®/2 Lisp Programming Quick Reference	Revised
iPSC®/2 DECON Commands Quick Reference Guide	New

iPSC®/2 System Administrator's Manual Set

iPSC®/2 Site Preparation Guide	New
iPSC®/2 Hardware Installation Manual	New
iPSC®/2 System Administrator's Guide	Revised
iPSC®/2 System Administrator's Guide Change Notice	New
iPSC®/2 VME Interface Reference Manual	New

TCP/IP Documentation

Intel® TCP/IP for System V/386 User's Guide and Reference
Intel® TCP/IP for System V/386 Administrator's Guide and Reference
Intel® TCP/IP for System V/386 Programmer's Guide and Reference

SRM Documentation

SYP301 Installation and User's Guide (packaged with System Resource Manager)

UNIX Literature Set

UNIX System V Network Programmer's Guide
UNIX System V Programmer's Guide, Vol. I
UNIX System V Programmer's Guide, Vol. II
UNIX System V Programmer's Reference Manual
UNIX System V User's Guide
UNIX System V Utilities Release Notes
UNIX System V Streams Primer
UNIX System V Streams Programmer's Guide
UNIX System V System Administrator's Guide
UNIX System V System Administrator's Reference Manual
UNIX V - The Quick Reference Guide

December 1989
Order Number: 311841

iPSC®/2 SYSTEM SOFTWARE

RELEASE 3.1

PRODUCT RELEASE NOTES

Intel® Corporation

REV.	REVISION HISTORY	DATE
---	Original Issue	12/89

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

PURPOSE/SCOPE

These release notes provide the latest information on features, limitations, and workarounds for Release 3.1 of the iPSC®/2 System. 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 3.1 Customer Letter that accompanied your software.

For installation instructions, refer to the *iPSC®/2 System Administrator's Guide*. When installing or removing the iPSC software, make sure that *no user* on the system (including *root*) is running **csh**.

ORGANIZATION

Chapter 1	Describes features of the iPSC/2 System Software that are new with Release 3.1.
Chapter 2	Describes known limitations and their workarounds.
Chapter 3	Provides suggestions for using the Concurrent File System™.
Chapter 4	Provides information on node memory usage and suggestions for optimizing message passing performance.
Appendix A	Provides reference manual pages for the Fortran and C versions of the **gcolx()** system call.

NOTATIONAL CONVENTIONS

- **Bold** font is used for anything that must be entered exactly as shown, including:

  Command names (including absolute pathname versions)
  Command switches and options
  System Call names
  Routine names (predefined *and* user-defined)
  Reserved words

- *Italic* font is used for:

  Variables
  File names (including absolute pathname versions)
  Directory names
  Process names
  User names
  Emphasis

- ***Bold-italic*** font is used for user input (what the user enters in response to some prompt).

- `Plain-Courier` font is used for:

  Code examples
  Computer output (prompts and messages)
  Error messages
  Values of variables

- **`Bold-Courier`** font is used for the names of keyboard keys (which are enclosed in angle brackets). For example:

  `<Alt>` **`<Backspace>`**
  `<Break>` **`<Ctrl>`**
  `<Delete>` or **`<Del>`** **`<Enter>`**
  `<Esc>` **`<Tab>`**

  A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. This has the effect of producing a single keystroke (which is why there is only one set of angle brackets). For example:

  `<Ctrl-S>`
  `<Ctrl-@>`
  `<Ctrl-Alt-Del>`

TABLE OF CONTENTS

CHAPTER 1 PRODUCT FEATURES

CHAPTER 2 LIMITATIONS AND WORKAROUNDS

CHAPTER 3 CONCURRENT FILE SYSTEM™

CHAPTER 4 USER INFORMATION

APPENDIX A MANUAL UPDATES

PRODUCT FEATURES 1

Release 3.1 of the iPSC/2 system software runs on all hardware configurations of the iPSC/2 System that include the PC586 Ethernet Controller. The following sections describe the features that are new in Release 3.1:

- Concurrent File System™
  - 9-track tape support
  - File fixer
  - Extended file size
  - Enhanced **restrictvol()**
  - Asynchronous I/O improvements
- New routines
  - **gcolx()** system call
  - Extended integer math system calls
  - **plogon** and **plogoff** commands and system calls (for system administrator only)
- New DECON **wait** command
- Profiling on the nodes
- Bug fixes

CONCURRENT FILE SYSTEM™

Release 3.1 contains these improvements to the Concurrent File System™:

1. **9-track Tape Support**
   Release 3.1 supports 9-track tape access as a CFS device.

2. **File Fixer**
   The **bootcube** command now finds and *fixes* most concurrent file system inconsistencies. Previously, the **bootcube** command only found and *reported* these inconsistencies.

3. **Extended file size**
   The size of a file is now limited only by the number of disks available. Previously, file sizes were constrained to less than 2G bytes.

4. **Enhanced restrictvol()**
   restrictvol() now lets you select the target volume for all subsequent **opens**.

5. **Asynchronous I/O**
   The software now supports a larger number of asynchronous I/O requests (limited only by available memory). In addition, all I/O is faster.

NEW ROUTINES

Release 3.1 contains these new routines:

1. **gcolx() system call**
   gcolx() is a faster version of **gcol**; it requires fewer parameters. Appendix A contains reference manual descriptions for the C and Fortran versions of this system call. Please add these pages to your *iPSC®/2 Programmer's Reference Manual.*

2. **Extended integer math system calls**
   With Release 3.1, you can do 64-bit integer arithmetic. These calls support the extended file sizes in CFS.

eadd()	Add two extended numbers.
ecmp()	Compare two extended numbers.
ediv()	Divide extended number by integer.

emod() Get remainder of **ediv()**.

emul() Multiply extended number and integer.

esub() Subtract two extended numbers.

3. **plogon and plogoff commands and system calls**
   These commands and system calls let the system administrator start and stop the logging of process-related information (to monitor cube usage).

For more information on these new routines, refer to the *iPSC®/2 Programmer's Reference Manual.*

NEW DECON WAIT COMMAND

When you have several node processes running, the DECON prompt returns after the first process hits its breakpoint and displays its breakpoint message. Previously, the only way to check for the other messages was to issue repeated **status** commands. With Release 3.1, DECON provides a **wait** command that waits until all nodes have stopped, and then displays all messages and returns the DECON prompt.

For more information, refer to the *iPSC®/2 DECON User's Guide.*

ANSI C COMPILER

NOTE

Some iPSC/2 documentation may refer to the ANSI version of the C compiler. Regrettably, the ANSI version of the C compiler is not available with this release. It will be available in the near future.

PROFILING ON THE NODES

You can incorporate runtime profiling into your node program by compiling your application using the **-p** option. Then, load the program and run it as usual. At the end of execution, the profiling data is stored in a file which you can analyze using the UNIX **prof** tool.

For more information, refer to the *iPSC®/2 User's Guide* and the *UNIX System V/386 Programmer's Reference Manual.*

BUG FIXES

With Release 3.1, the following NX/2 Node Executive bugs have been fixed:

1. **Printed NaNs from SX applications**
   NaNs (not a number) are now formatted correctly when printed from an SX node application.

2. **Rewinding a file with lseek()**
   Using the C **lseek()** call on the nodes to rewind a file now returns the correct file pointer value.

3. **strcmp() on last environment variable**
   In node applications, **strcmp()** no longer generates a "Memory fault" error when given the last environment variable.

4. **Handler parameters after global hsend()**
   After a global **hsend()**, the *node* and *pid* parameters in the handler routine now contain the expected values.

With Release 3.1, the following Node Shell (**nsh**) bugs have been fixed:

5. **ls -s on the nodes prints incorrect values**
   ls -s now prints correct values, even if the number of bytes is in the multiple-gigabyte range.

6. **Maximum SRM shell file size**
   The node shell now executes the entire shell file, even if the file is located on the SRM.

With Release 3.1, the following Concurrent File System™ bugs have been fixed:

7. **Using CFS commands/routines in a non-CFS system may hang your application**
   If you booted a non-CFS system or a cube as a non-CFS system (using **bootcube -o**), using CFS commands and routines no longer hangs the system.

8. **restrictvol() applies to file block allocation after file initially created**
   With Release 3.1, setting *fd* to `-1`, causes all subsequent file creations to happen on specified volumes.

9. **Number of disks per I/O node**
   bootcube no longer requires that every I/O node have exactly the same number of disks.

With Release 3.1, the following Concurrent Workbench™ bugs have been fixed:

10. **load and killcube**
    Calling **killcube** immediately after calling **load** no longer causes **killcube** to hang.

11. **killcube fails if waitcube is in the background**
    If a **waitcube** is executing in the background, a **killcube** on the same cube no longer fails.

12. **Outstanding crecv() or msgwait() will block any hrecv()**
    An outstanding **crecv()** or **msgwait()** no longer blocks any **hrecv()** from being serviced.

13. **Multiple getcube calls**
    In a host program, multiple calls to **getcube** now leave you attached to only the last cube allocated.

14. **Loading vector applications on nonvector nodes**
    Loading a vector application into a nonvector node no longer causes the **load** to hang. An error message is returned and the load is aborted.

15. **Running multiple processes on vector boards**
    Attempting to load a second process on the vector board no longer corrupts the first process. The second load is aborted.

With Release 3.1, the following Concurrent Workbench™ assembler and linker bugs have been fixed:

16. **Unsupported assembler switch**
    The **-Y** assembler switch is now supported.

17. **Data alignment by the compiler and linker**
    The **-vx** switch on the Fortran and C compilers now causes the linker to correctly align all variables.

With Release 3.1, the following Concurrent Workbench™ Fortran and C compiler bugs have been fixed:

18. **Compiler -p switch**
    The Fortran compiler's **-p** profiling switch now causes the correct value to be returned by the function **IARGC**.
    The C compiler's **-p** profiling switch now places the correct value in *argc*.

With Release 3.1, the following Concurrent Workbench™ remote host bugs have been fixed:

19. **First getcube may fail**
    The first **getcube** after a **bootcube** no longer fails.

With Release 3.1, the following DECON concurrent debugger bug has been fixed:

20. **Using DECON to access Fortran variables declared in COMMON**
    To access (i.e., **display** or **assign**) a variable declared in **COMMON**, you no longer need to indicate the name of the common block in which it is found. Also, you can now display either member of an **EQUIVALENCE**.

With Release 3.1, the following Cube Diagnostic Program bug has been fixed:

21. **Timeout Table Overflow**
    When you run **cdp** for extended periods of time, the UNIX software no longer hangs the SRM.

With Release 3.1, the following TCP/IP bug has been fixed:

22. ***TERM* variable in .login**
    The *TERM* variable is now set correctly when you **rlogin**.

LIMITATIONS AND WORKAROUNDS 2

This chapter describes known limitations and suggested workarounds for the following Release 3.1 iPSC/2 system components:

NX/2 node executive
Node shell (**nsh**)
Concurrent File System
Concurrent File System tape
Concurrent Workbench
Concurrent Workbench assembler and linker
Concurrent Workbench C
Concurrent Workbench Fortran
Concurrent Workbench remote host
DECON Concurrent Debugger
iPSC/2 simulator
UNIX software
Network File System (NFS)

NOTE

Read the following sections carefully. Report any problems you encounter while using your iPSC/2 system to iSC Technical Support at:

1-503-629-7777 (Customer Support Hotline in Oregon)
1-800-421-CUBE (Hotline outside of Oregon)
(44) 793 696 619 (in England)
Your Local Intel Sales Office (in Europe)
support@isc.intel.com (csnet address)

NX/2 NODE EXECUTIVE

1. **Message length**
   Check that the *len* parameter in a node send or receive call does not exceed the size of the buffer. Unpredictable results can occur if *len* exceeds the buffer size. (Refer to the *iPSC®/2 Programmer's Reference Manual* for more information on these parameters.)

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

3. **Node may hang 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.

4. **Memory fault when more than 300 channels opened**
   Using the Fortran node compatibility library **copen()** routine to open more than 300 channels aborts the program and causes a "Memory Fault" error message.

5. **flushmsg() may not flush all messages**
   flushmsg() may not flush a message that is in transit.

6. **cubeinfo command displays only system name**
   The **cubeinfo()** routine 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.

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

8. **waitone() call completion code error**
   When a node process stops due to an overflow exception, the **waitone()** call incorrectly returns a completion code of `-243`. It should return `-252`. (Refer to the *iPSC®/2 Programmer's Reference Manual* for a description of the completion codes.)

9. **Incorrect error message displayed during handler() routine use**
   If the **handler()** routine is used to catch a general protection violation (exception 13), a stack overflow error message is incorrectly displayed.

10. **Node programs that cannot be profiled**
    Node programs that call **sleep()** or **alarm()** cannot be profiled.
    Workaround: Don't use **sleep()** or **alarm()** when profiling a node program.

11. **load() error message misleading**
    If you try to load a nonexistent or null-length file, you will get the error message "Exec format error."

12. **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.

NODE SHELL (nsh)

1. **startcube command not available on nodes in this release**
   The node command **startcube**, documented in the *iPSC®/2 Programmer's Reference Manual*, is not available in this release. It is available on the SRM.

2. **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.

3. **Interrupting node stream command during tape read can hang tape driver**
   Pressing the **<Del>** key while using the node **stream** command to read a tape into CFS can cause the tape driver to hang. If this happens, kill the *fserver* process.

4. **nsh wildcard (*) does not work properly for filenames larger than 14 characters**
   The **nsh** wildcard character (*) does not expand properly for CFS filenames longer than 14 characters.

5. **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. For example, to move */cfs/dirname/progs* to */cfs/dirname/src*, **cd** to */cfs/dirname*, and use the following command sequence:

```
mkdir src
cd progs
tar cf - . | (cd ../src; tar xvf -)
cd ..
rm -r progs
```

   This can also be used to move or copy directories to other places in the file system.

6. **Loading and killing processes on nodes not assigned to you**
   Under certain circumstances, it is possible to load and kill processes on nodes that are not assigned to you. However, this will not occur if you use the iPSC/2 system as documented.

7. **-c option not available for nsh**
   The **csh**'s **-c** option is not supported for the node shell.

CONCURRENT FILE SYSTEM™

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

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

3. **Maximum number of files allowed on CFS is approximately 39,000**
   Attempting to create more than approximately 39,000 files in CFS causes a "File table overflow" error and hangs the file system. To recover, use **bootcube** and **mkcfs**.

4. **mkdev gives incorrect error message**
   If **mkdev** is used to create a special file that already exists, it prints the error message "Not a directory," rather than the expected message "File exists."

5. **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.

6. **cbackup and crestore must use the CFS tape device from within nsh**
   cbackup and **crestore** executed on the SRM (outside of the node shell) do not know about the CFS tape device.
   Workaround: Use **cbackup** and **crestore** from within the node shell.

7. **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**.

8. **Execution of cbackup and crestore not properly restricted**
   Because the **cbackup** and **crestore** commands can change the entire file system, only the superuser should be able to use them. However, the commands do not currently enforce this restriction.

9. **iowait() returns incorrect error message**
   If an error occurred when using **iread()** (such as reading past EOF), **iowait()** prints "Error 0."

10. **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.

11. **cwrite() incorrectly allows writing to TTY device**
    cwrite() currently succeeds when writing to a file descriptor opened on */dev/tty*. This should fail with an error message.

12. **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.

CONCURRENT FILE SYSTEM™ TAPE

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

2. **End of medium detection**
   End of medium detection works reliably only for writes to tape. When the end of medium is reached, the **write()** returns a `-1` and sets *errno* to `ENOSPC`. The application must explicitly close the device and then reopen it after a new tape has been mounted.

3. **Exabyte not supported**
   The Exabyte 8mm tape driver is not supported in this release.

4. **Opening CFS tape device**
   Two nodes may open a CFS tape device at the same time, which may result in interleaved output to the tape.

5. **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.

6. **Setting I/O mode on a tape device**
   Using **setiomode()** on a tape device should be illegal, but it currently succeeds.

7. **Multiple names for a tape drive treated the same**
   If there is more than one name for a tape drive, and one name gets set to `no rewind`, then they will all be set to `no rewind`.

8. **Invalid tape operations may hang tape driver**
   Following a **write()**, any tape forward movement that is not a **write()** should return an error, but does not. If this occurs and you do a **rewind()**, the next forward movement may cause the tape driver to hang. To recover, use **bootcube**.

9. **Attempting to write on a write-protected tape**
   Writing to a write-protected tape does not return an error or write to the tape. A second attempt will hang the tape driver. To recover, use **bootcube**.

10. **Reading a tape using the wrong block mode hangs the tape driver**
    Reading a tape in a fixed block mode that is different than the one in which it was created hangs the tape driver. To recover, use **bootcube**.

11. **Accessing an off-line tape drive**
    Attempts to access a tape driver that is off-line cause CFS to hang. To recover, use **bootcube**.

12. **Must read same size blocks as were written**
    Using **cread()** to read blocks on a tape device that are larger than those written results in the message, "Attempt to read past end of file," even when this is not the case.

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. **Writing variable length records may hang tape driver**
    Writing to the tape in variable block mode may hang the tape driver if there is a mixture of large records (greater or equal to 100 bytes) and small records (less than 100 bytes). The problem is that small records can arrive at the I/O node *before* large records that were sent earlier. To recover, use **bootcube**.

15. **Backing off the beginning of a tape hangs the driver**
    Using a backspace file (MTBSF) or backspace record (MTBSR) command when the tape is at the beginning of the tape hangs the tape driver. To recover, use **bootcube**.

16. **Incorrect output from tapemode command**
    The tape density displayed by the **tapemode** command never reflects a density change from 6250.

17. ***diskproc* may hang**
    Occasionally, the *diskproc* will hang and write either `ESP error status` or `ESP FIFO not clear in MSGIW` in */usr/ipsc/log/iocube.log*. This is a fatal error for which there is no workaround. To recover, use **bootcube**.

CONCURRENT WORKBENCH™

1. **waitcube and killcube**
   The **waitcube** and **killcube** commands occasionally fail to return. To recover, press the **<Del>** key to kill the command, clean up unwanted *fservers*, and then do a **relcube**. If the **relcube** fails to return, press the **<Del>** key, then do a **bootcube**.

2. **Nodes marked unusable**
   If you get a message saying, "*nn* node(s) not responding" (where *nn* is the number of nodes), do a **bootcube** to reset the nodes. This message appears only on the system console. If problems persist, use **cdp** to check for bad node boards. The *iPSC®/2 System Administrator's Guide* describes **cdp**.

3. **Message length**
   Check that the *len* parameter in a host send or receive call does not exceed the size of the buffer. Unpredictable results can occur if *len* exceeds the buffer size.

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

5. **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);
```

6. **Releasing a cube**
   Use **killcube** prior to releasing the cube. **relcube** alone does not always clean up processes properly, causing subsequent programs to fail. Using **killcube** avoids this problem. Use the **ps -ad** node command to verify that the clean-up is complete.

7. **setsyslog()**
   Calling **setsyslog()** when the host program is already piping the output through **syslog** causes the host program to hang. To recover, press the **<Del>** key.

8. **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, press the **<Del>** key.

9. **"Not enough memory" error message**
   killcube and **relcube** may fail to recover node memory, resulting in "not enough memory" error messages. Use **bootcube** to recover.

10. **killcube flushes 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.**

11. **getcube may allocate one or more SX nodes**
    When you have a mixture of SX and 387 nodes, a **getcube** with no *cubetype* specified may allocate a mixture of node types, without telling you what types of nodes it allocated. This can cause floating-point exceptions, if you execute 387 code on an SX node, or vice versa.
    Workaround: Use **getcube** with either the **-tf** or **-tsx** option, to force it to allocate a cube of either all 387 or all SX nodes.

12. **bootcube must complete before you execute getcube**
    bootcube must be complete before attempting to execute **getcube**, otherwise the whole cube may be marked as bad.
    Workaround: To recover, do another **bootcube.**

13. **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.

14. **Maximum of 11 outstanding irecv()s may be posted on an SRM**
    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.

15. **Bad vector board**
If the cube contains a bad vector board, **bootcube** may do one or both of the following:

    - Not detect the board as being bad
    - Indicate that it is bad, using */tmp/cubeconf* instead of */usr/ipsc/conf/cubeconf*, even though these two files are identical

16. **Using -c switch on cube commands changes your default cube**
Using the **-c** switch with the **killcube, load, startcube,** or **waitcube** command leaves you attached to the specified cube.

17. **killcube hangs if load cannot complete**
If the node message buffers are filled before a load is complete, a subsequent **killcube** hangs. This situation can be avoided by ensuring that the load is complete before any process sends a large number of messages to the nodes.

18. **Profiling host applications may cause problems**
The getcube() and newserver() routines abort if the host application in which they are called is compiled using the profiling switch **-p.**

19. **Redirecting I/O to node programs**
The **getcube** and **newserver** commands can no longer be used to redirect I/O to an application running on the nodes. Use the **waitcube** command instead.

20. **load error message misleading**
If you try to load a nonexistent or null-length file, you will get the error message "Exec format error."

CONCURRENT WORKBENCH™ ASSEMBLER AND LINKER

1. **Directive incorrectly used**
In assembling data objects declared with the **.data** instruction, the 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 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.

3. **Data alignment by the compiler and linker**
   If you use the vector board and have misaligned data, you may get a parity error under certain circumstances. This is due to a hardware problem with the 386 node's iLBX®-II interface and the Vortex iLBX-II interface. To avoid this problem, you must ensure that all variables start on their natural boundaries, and that all strings start on a 32-bit boundary. The Fortran compiler does this for all variables except character strings.
   Workaround: When sending and receiving character strings, you should ensure that these strings are multiples of four bytes. Also, the string's starting address should be on a 32-bit word.

CONCURRENT WORKBENCH™ C

The default C compiler has been changed from the AT&T version of the pcc compiler to the Green Hills C compiler, gcc. This was done for performance and compatibility with Green Hills Fortran.

UNIX software on the SRM is built with the AT&T pcc compiler, rather than the Green Hills compiler. The support libraries for the Green Hills compiler have been remade with the AT&T include files for compatibility with UNIX software. These libraries are now the same as those used by the pcc compiler. Problems encountered with the libraries can be checked against the pcc compiler.

AT&T's pcc compiler is located in:

/usr/bin/pcc	A script that calls the pcc compiler located in */lib/cc*
/lib/cc	AT&T's pcc compiler

To invoke AT&T's pcc compiler, enter:

```
pcc
```

or

```
/lib/cc
```

Green Hill's compiler is located in:

/bin/cc	Copy of */usr/bin/gcc*
/usr/bin/gcc	Copy of */bin/cc*

To invoke Green Hill's compiler, enter:

cc

or

gcc

The following are Concurrent Workbench C limitations and workarounds:

1. **asm directive support**
   The Green Hills compiler is missing some support for the **asm** directives.

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

3. **-B compiler switch**
   The **-B** compiler switch should be used when compiling programs for use with DECON (the debugger). The **-B** switch creates the optimum debugger environment by setting the appropriate switches. By listing **-B** last in the sequence of compiler switches, it will have precedence over the previous switches, except for the **-O** and **-OLM** switches. These switches should be removed.

   The **-B** switch sets the following switches: **-g, -ga, -X9, -X18, -X39, -X168, -X211, -X219, -X230, -X307, -X356.**

   If a program works correctly with these switches, but incorrectly in more optimized versions, you should report a compiler bug to iSC.

4. **SX compilation**
   Compiling for the SX requires that the **-sx** switch be on both the compile and link commands. Modules compiled with the **-sx** compiler switch may not be linked with those compiled without the **-sx** switch, and vice versa.

   When you call math routines that are in */lib/libsxm.a*, you must also link them using the **-lm** switch (i.e, **-lm -sx**).

5. **Data alignment by the compiler and linker**
   Refer to item 3 in the section "Concurrent Workbench™ Assembler and Linker."

6. **Assignments to unused variables**
   Occasionally, assignments to unused variables are optimized out of the code, even when optimization is not invoked.

7. **Illegal external variable names**
   You may not use the following names as external variables in your host programs: **align, any, C, D, E, e, f, FP, herror, host, hostbuf, HOSTDB, hostf, index, IP, KS, L, line, LogFile, LogMask, nbuf, net, P, proto, R, recv, S, s, send, shifts, syslog,** or **token.** Network library */usr/lib/libsocket.a* has defined these names to be external. Errors such as "Bus error - Core dump" may occur at runtime if you use them in an application.

CONCURRENT WORKBENCH™ FORTRAN

1. **Fortran INCLUDE statement**
   The Fortran **INCLUDE** statement is supported in the compiler. The absolute path of the include file must be used if the file is not located in the current directory. For example:

   ```
   INCLUDE '/usr/include/fcube.h'
   ```

2. **gray(), ginv(), cubeinfo(), and csendrecv() return integers**
   The iPSC/2 functions **gray(), ginv(), cubeinfo(),** and **csendrecv()** must be declared as integers, either with the include file */usr/include/fcube.h* or with user-provided declarations.

3. **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.

4. **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**
   - 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
   - Mix of character and any other type within a common block

5. **-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.

6. **SX compilation**
   Compiling for the SX requires that the **-sx** switch be on both the compile and link commands. Modules compiled with the **-sx** compiler switch may not be linked with those compiled without the **-sx** switch, and vice versa.

7. **Invalid invocation line switches**
   Invalid compiler switches are silently ignored.

8. **VAX/VMS extensions**
   Compiling with the **-vms** switch makes VAX/VMS Fortran extensions available during both compilation and linking. For more information, refer to the *VAX-11 Fortran User's Guide and Language Reference Manual*, available from Digital Equipment Corporation.

9. **-B compiler switch**
   The **-B** compiler switch should be used when compiling programs for use with DECON (the debugger). The **-B** switch creates the optimum debugger environment by setting the appropriate switches. By listing **-B** last in the sequence of compiler switches, it will have precedence over the previous switches, except for the **-O** and **-OLM** switches. These switches should be removed.

   The **-B** switch sets the following switches: **-g, -ga, -X9, -X18, -X39, -X168, -X211, -X219, -X230, -X307, -X356.**

   If a program works correctly with these switches, but incorrectly in more optimized versions, you should report a compiler bug to iSC.

10. **Do not link Fortran node programs using -lnode switch**
    Fortran node programs should not be linked to the node library using **-lnode**, because it links the libraries in the wrong order. This may result in "Memory Fault" errors at runtime. Use **-node** instead.

11. **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.

12. **I = MOD(J, 0) causes error message**
    The expression I = MOD(J, 0) causes an "Illegal instruction - core dumped" message.

13. **-OL switches occasionally cause an error message on SX**
    The -OL (but not -O) optimization switches occasionally cause an "Illegal address" error while executing on the SX.

14. **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.

15. **Fortran scratch files**
    Fortran scratch files do not have unique names. They are created in the current directory of the executing program. This means that unless the node program changes directories, the scratch file is created on the SRM's disk, and all node programs share the same scratch file. The scratch file is deleted when the program terminates.

16. **Missing "FILE=" specifier**
    In Fortran, if you are missing the **FILE=** specifier in the **open()** call, unique file names cannot be created. The file is created in the current directory of the executing program. This means that unless the node program changes directories, the file is created on the SRM's disk.

17. **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.

18. **Duplicate variable definitions overlooked**
    The compiler does not catch duplicate variable definitions when one variable is declared in COMMON within an include file and the other is declared as a local in-line.

19. **Using -vms switch changes control character for formatted EOL**
    Compiling with the **-vms** switch changes the end-of-line character in formatted output from **<Ctrl-M>** to **<Ctrl-J>**.

CONCURRENT WORKBENCH™ REMOTE HOST

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

2. **Maximum of seven cube partitions per remote host**
   Some remote host operating systems (e.g., Sun OS 3.0) support a maximum of only seven cube partitions per remote host. Attempting to allocate an eighth cube may cause the remote host to hang. To recover, reboot the remote host and cube.

3. **setenv TTY 'tty'**
   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
   ```

4. **Remote host include files**
   To use remote host include files in remote host programs, you must compile with the remote compile command **rcc -cpp...**, and do one of the following:

   A. Use absolute paths for the include files:
      In application: `#include "/usr/include/suntool.h"`

   B. Use relative paths in the **#include** statement, and use the **-I** switch on the compile command line:
      In application: `#include "suntool.h"`
      command line: `rcc -cpp -I/usr/include`

   C. Have a *rhostinc.h* file included in the application that contains the standard includes:
      In application: `#include "rhostinc.h"`
      In *rhostinc.h*: `#include <suntool.h>`

   D. The iPSC/2 include files *cube.h*, *fcube.h*, and *errno.h* are installed in the user-specified *INCDIR* directory. To modify the actual location of *INCDIR*, edit the top-level makefile found in the *rhost* source code, and run **make** on the *rhost* software.

5. **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.

6. **Heavy message passing**
   Heavy message passing between a remote host and the nodes for a sustained period of greater than 15 hours may cause the remote *commser* process to abort. To recover, execute a **bootcube** on the remote host as root and release the cube on the SRM.

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

8. **FORCETYPE range messages**
   Message types in the FORCETYPE 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. Force messages sent from a node to a **crecv()** or **irecv()** on a remote host will be lost. Force messages can be sent from host program to host program, and received using **crecv()** or **irecv()**. Force messages can also be sent from node program to node program, and received using **crecv()** or **irecv()**.

9. **Nodes cannot load or exec files on remote host**
   Node processes cannot **load()** or **exec()** files on the remote host. The **load()** and **exec()** calls executed on the node try to find the files on the SRM, not the remote host workstation.
   Workaround: Any files that need to be **loaded** or **execed** from the nodes should be resident in the SRM file system. Either **rcp** the files from the remote host to the SRM, or use NFS to mount the remote host file system onto the SRM.

10. **load command environment enlarged to 3K bytes**
    The **load** command environment has been increased from 1K bytes to 3K bytes to handle large argument lists.

DECON CONCURRENT DEBUGGER

1. **-B compiler switch**
   The **-B** compiler switch should be used when compiling Fortran and C program for use with DECON. The **-B** switch creates the optimum debugger environment by setting the appropriate switches. By listing **-B** last in the sequence of compiler switches, it will have precedence over the previous switches, except for the **-O** and **-OLM** switches. These switches should be removed.

   The **-B** switch sets the following switches: **-g, -ga, -X9, -X18, -X39, -X168, -X211, -X219, -X230, -X307, -X356.**

Compiling object files for use with DECON may be done as follows:

```
f77 -c -B host.f
```

or

```
cc -c -B host.c
```

2. **Compiler optimizations may limit access to variables**
Some compiler optimizations don't get turned off by **-B**. Consequently:

   - You can't access variables that are never used; the compiler optimizes them away.
   - DECON may have difficulty accessing Fortran implicit variables and loop counters. You may appear to access a loop counter when you're actually not.
   - You can't access subroutine arguments that are passed by reference.

3. **Unsupported features**

   - Data breakpoints on host processes
   - Label breakpoints on host and node processes
   - Assigning and displaying C bit variables
   - Assigning and displaying Fortran character variables

4. **Unsupported commands** The following commands were available in Release 3.0, but are not supported in Release 3.1:

   act
   break after
   break when
   deact
   kill
   reset
   scope
   step *source_line_number*
   trace

5. **Register variables in C**
   Displaying and assigning register variables may produce inaccurate results because the compiler does not guarantee to keep these variables in registers at all times. In general, if you can successfully display a register variable, you may then assign a value to it.

6. **step command limitations**
   Occasionally, the compiler maps multiple source statements of a C or Fortran program into a single code address, especially when these statements are short (e.g., `cnt = 0`). When this is the case, DECON will execute more than one statement when it is asked to do a single step.

   If a program hangs during a step (for example, if it is waiting on a message), a keyboard interrupt may confuse DECON. Take care not to step into such a situation.

7. **type command limitations**
   The type of a Fortran **character*n** variable is displayed as **character**. The type of a **logical*1** Fortran variable is displayed as **character**, and the type of a **logical*4** is displayed as **integer*4**.

8. **Interrupt while host process is doing a system call**
   If you interrupt a host process while it is executing a **csend()**, **crecv()**, **cprobe()**, or **msgwait()** system call, the host process will hang.

9. **Fortran array starting index**
   COFF only records the number of elements in an array. DECON assumes that the first element has index 1 for Fortran arrays and 0 for C arrays. If you have a starting index other than 1 in your Fortran program, you need to adjust the index to 1 so that **assign** and **display** can identify the correct element.

10. **Maximum pathname**
    The maximum length of the pathname used in the **source** command is 32 characters.

11. **unset and unalias**
    Both **unset all** *and* **unalias all** delete all aliases *and* all variables.

12. **Setting data breakpoints**
    You can't set data breakpoints on two distinct elements of the same object (e.g., two array or structure elements).

13. **Only four data breakpoints per scope**
    DECON supports only four data breakpoints in any scope, but doesn't complain if you set more.

14. **Breakpoints on Fortran common block elements**
A data breakpoint set on an element of a Fortran common block, when encountered, will print the name of the variable where the breakpoint was set, which does not necessarily match the name where the breakpoint occurred.

15. **Assigning or displaying large arrays**
Arrays with more than 64K bytes in a dimension cannot be displayed or assigned with predictable results.

16. **Setting breakpoints on main program**
If you set a breakpoint on **main**() (in C) or **MAIN**() (in Fortran), the debugger will hang when you try to run.

17. **Cube size limitations**
DECON is not reliable on cubes larger than eight nodes.

18. **Erroneous error message**
The following error message may appear inappropriately:

```
error: stopped at different statements
```

Ignore the message, and use the appropriate commands (**break, display, list**) on one process at a time to verify correct operation.

19. **Termination after errors**
DECON doesn't usually recognize when a node program gets an exception, and does not cause normal error exit processing to occur. Use the **<Del>** key to get back to the DECON prompt, if necessary.

20. **alias limitation**
Using a debug variable to define an alias aborts DECON.

21. **assign limitations**

- Avoid prompted assigns for more than one array element per assign command. In some cases, DECON stores an entered value in the wrong array element.
- Complex variables must be assigned in prompted mode, not on the **assign** command line.
- In prompted assigning of complex variables, pressing <Enter> does not retain the variable's current displayed value, as with other types. Always enter complex variable values when prompted for them, even if you do not want to change the value.
- When entering character values, enclose them in single quotes.

22. **break limitations**

- In Fortran host programs, set breakpoints on the line following a GOTO label, instead of on the labeled line itself. The actual jump point may compile to a location just after the address of the label.
- Do not use two break statements where one will do. You cannot set a breakpoint on an object if a breakpoint for that object already exists in another context. Remove the first breakpoint and define a new one that includes both contexts.
- Do not set breakpoints on Fortran END statements.
- In some cases, DECON may not be able to break at the first statement of a program.

23. **display limitations**

- Displaying whole C structures for multiple nodes may show incorrect values. Choose one member at a time (on any number of nodes) or one node at a time (when displaying more than one member).
- Displaying C character pointer strings must be done one character at a time.
- Fortran logical variables are not displayed with true/false interpretation, `1` is `.TRUE.` and `0` is `.FALSE.`.

24. **load limitations**

- Do not use two load commands where one would do the job. Using multiple load commands for different instances of the same program with the same *pid* on different nodes may corrupt DECON's internal tables.
- To restart or reload your application, exit DECON and then invoke it again.

25. **source limitation**
To set the source directory back to the current directory, use the pathname of the current directory (full or relative) or a period enclosed in single quotes. For example:

```
source '.'
```

Without the quotes, DECON may consider the directory file to be the source file for listings.

26. **step limitation**
Stepping off the end of host or node programs causes erroneous error messages.

27. **wait limitation**
When used with **step** commands, the **wait** command may hang. An interrupt should return a DECON prompt.

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. **New CFS calls and some node calls not supported**
The new I/O calls (such as **cread()**, **cwrite()**, **iowrite()**, **iodone()**, and **iomode()**) are not supported in the simulator. The NX **handler()**, **hsendrecv()**, **csendrecv()**, **isendrecv()**, and **hsend()** calls are not supported either.

3. **FORCE_TYPE messages not supported**
FORCE_TYPE messages are not supported. Types greater than 40000000 hex are treated the same way as any other message.

4. **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/2 system.

5. **Remote host and cube sharing**
Remote host and cube sharing functions are not supported by the simulator.

6. **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.

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

8. **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.

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

10. **UNIX 5.2 ATT version software**
    To make and install UNIX 5.2 ATT version software, enter:

    ```
    make att52
    make instlatt
    ```

    Other versions are documented under "Installing the Simulator" in Chapter 2 of the *iPSC®/2 Simulator Manual.*

UNIX SOFTWARE

1. **Exit value converted by sh**
   /bin/sh converts a C program exit (`-1`) value to `255`.

2. **Cartridge Tape**
   The cartridge tape occasionally hangs if the **<Del>** key is pressed while reading or writing the tape. To recover, you must turn the SRM off and back on.
   Workaround: Avoid using the interrupt key while the cartridge tape is just beginning to read or write the tape. Wait for 10 seconds after the transfer to or from the tape has started.

3. **Cannot use cpio with noclobber set**
   You cannot have **noclobber** set in your environment when you wish to use **cpio** to write to a disk device.

4. **Forcing a script to execute in sh**
   To force a script to execute in **sh**, the first line of the script must be ":".

5. **Forcing a script to execute in csh**
   To force a script to execute in **csh**, the first line of the script must be "#!". (UNIX System V version 3.0 software required only a "#" as the first character in the file.)

6. **unsetenv not supported**
   The **unsetenv** command is not supported in UNIX System V version 3.2. **unsetenv** is not included in the **csh**.

7. **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.

8. **Trying to rewind tape when not tape is present hangs the system**
   Attempting to rewind a tape with **tapecntl -w** when no tape is in the drive hangs the system. To recover, power-cycle the SRM.

9. **lint does not work under csh**
   The **lint** command must be executed from the Bourne shell (**sh**).

NETWORK FILE SYSTEM (NFS)

1. **Keyboard interrupt kills background process**
   A keyboard interrupt kills a remote **make** that is running in the background.

2. **cron killed by at command**
   Executing the **at** command kills the *cron* process if NFS has been installed.

CONCURRENT FILE SYSTEM™ 3

INTRODUCTION

The iPSC/2 Concurrent File System™ can be a very useful addition to your concurrent environment. Information on creating an concurrent file system, and backing up and restoring your files is contained in the iPSC®/2 System Administrator's Guide. Here are some further suggestions on using your Concurrent File System and keeping it healthy.

DEALING WITH DCHK AND FCHK FAILURES

When **bootcube** is run, the CFS directories and files are checked for consistency before it is started up. The CFS file fixer tries to fix any problems that are reported by the file checker.

LOADING EXECUTABLES

Node executables can be stored in and loaded from the Concurrent File System. Not only is loading from CFS faster, but it also allows you to save space on your host file system. To load node executables, use the **load /cfs/bin/nodeprog** command or the **load("/cfs/bin/nodeprog", ...)** call, from either the node or the host.

REMOTE HOST

To use the node shell (**nsh**) from a remote host workstation, you must copy all of the commands to CFS:

1. Login to the SRM.

2. Perform the following sequence to copy the commands to CFS:

```
getcube
nsh
cd /usr/ipsc/bin
mkdir /cfs/bin
cp /usr/ipsc/bin/[a-j]* /cfs/bin
cp /usr/ipsc/bin/[k-z]* /cfs/bin
exit
relcube
```

Once you have copied the commands, you must add the following to the *.cshrc.ipsc* file in your home directory on the remote host:

```
set path=(/cfs/bin)
set shell=/cfs/bin/csh
```

CFS ROUTINE SYNCHRONIZATION

Some of the library routines used to access the CFS occasionally perform some synchronization (message passing). The node processes participating in the synchronization must have the same PID, and all nodes in the cube must participate. The following table lists the routines and shows when (or if) they do such synchronization.

Routine	Conditions Causing the Routine to Synchronize
close()	Mode 1, 2, or 3
setiomode()	Always
lseek()	Mode 2
iseof()	Mode 2
cread()	Mode 2
cwrite()	Mode 2
iread()	Mode 2
iwrite()	Mode 2

Mode 3 is a synchronized mode. Routines **lseek()**, **iseof()**, **cread()**, **cwrite()**, **iread()**, and **iwrite()** do not perform any synchronization in Mode 3. The high performance provided by Mode 3 is attributable to this fact. Further information about I/O modes is in the *iPSC®/2 User's Guide*, with an example of writing to a file in each I/O mode.

CFS MEMORY REQUIREMENTS

All nodes involved in CFS must have at least 4 megabytes of memory. This includes Node 0 of the compute cube. For sites that currently have 1-megabyte node boards and want to add CFS to their system, a 4-megabyte replacement board for Node 0 will be included in the CFS upgrade.

USER INFORMATION 4

NODE MEMORY USAGE

The following table 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.

Node Type	Number of Message Buffers	Free Memory		NX/2 Code		NX/2 Data		Message Buffer Size		Total Node Memory
1M	512	630,784	+	118,148	+	197,244	+	102,400	=	1,048,576
4M	634	3,743,744	+	118,148	+	205,436	+	126,976	=	4,194,304
8M	1,146	7,827,456	+	118,148	+	213,628	+	229,376	=	8,388,608

To calculate the size of an 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: 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 node.

SUGGESTIONS FOR OPTIMIZING MESSAGE PASSING PERFORMANCE

There are several techniques that can be used in an application to reduce message-passing overhead. Some of these techniques are good programming practice for the iPSC/2 System and should be used in all applications. Other techniques are useful only when it is necessary to get the very best message passing performance (for example, when optimizing a critical section of an application).

The most important programming practice for the iPSC/2 System is to ensure that send and receive buffers are properly aligned and sized whenever possible. Although the message-passing system calls will work with any size or alignment of buffers, the hardware works only with well-aligned buffers. As a result, the software must copy messages which are in misaligned buffers, decreasing performance.

Send buffers can be any size, but should be aligned on a 4-byte boundary if the message is longer than 100 bytes. Messages smaller than 100 bytes may also benefit slightly from 4-byte buffer alignment. Receive buffers should always be aligned on a 4-byte boundary and should be a multiple of 4 bytes. In addition, receive buffers should be no more than 4096 bytes larger than the message that will be received in them. Note that the size of a buffer refers to the length parameter in a send or receive call. The buffer may be declared to be larger than the length if desired.

While these rules may seem strict, they are easy to follow because the compilers tend to align data on appropriate boundaries.

- In Fortran, avoid using **CHARACTER** or **LOGICAL*1** buffers if **INTEGER**, **REAL**, or **DOUBLE PRECISION** will do. If you have performance-sensitive **CHARACTER** or **LOGICAL*1** buffers, equivalence them to an **INTEGER** and try to make them a multiple of 4 bytes.

- In C, declare buffers as **int** or **long**, rather than **char** or **short**. If it is more convenient to declare the buffer **char** or **short**, make sure the size is a multiple of 4 bytes, and that other **char** or **short** arrays are also multiples of 4 bytes. Buffers allocated with **malloc()** or its derivatives will be correctly aligned.

Another useful technique is to arrange the application so that the receive call for a critical message is posted before the message comes in. In some cases, this will require the use of **irecv()** to post a receive before proceeding with a lengthy computation. The computation can then overlap with message reception.

In cases where it is difficult to overlap computation with message passing, use the simpler **csend()** and **crecv()** calls, instead of the more complex **isend()** and **irecv()**. **csend()** and **crecv()** are not only easier to use, but they also have slightly less overhead because their data structures in the system are preallocated, and they require fewer system calls to operate.

The techniques described above have the most impact on message-passing performance and are generally good practice. Below are additional tricks that can be used to gain slight additional speed, but at a cost that may not justify their use. Some of these techniques apply to C only.

- The regular system calls (such as **csend()**) check for errors by calling the underscore version (such as **_csend()**) and then inspecting the return value. You can save a little time by calling the underscore version directly (it takes the same parameters) and ignoring any errors. Of course, this technique should only be used on fully debugged programs.

- Avoid repeated use of calls like **mynode()** and **mypid()** in send and receive calls. Instead, declare variables for *node* and *pid*, and set those variables to the values returned by **mynode()** and **mypid()** at the beginning of the application. Use the variables in send and receive calls.

- You may obtain a small performance increase for large messages by making the buffers static and avoiding a call to **malloc()**.
- There is a tiny overhead associated with running the green LED on the node board. This overhead can be eliminated by calling **led()** once at the beginning of the program to transfer control of the LED from the system to the application.

MANUAL UPDATES A

This appendix contains reference manual descriptions for the C and Fortran versions of the **gcolx()** system call. Please remove these pages and add them to Chapters 2 and 3 of your *iPSC®/2 Programmer's Reference Manual*.

GCOLX() GCOLX()

Global concatenation operation for contributions of known length.

Synopsis

gcolx(*x, xlens, y*)

Parameter Declaration

char *x[];*
long *xlens[];*
char *y[];*

Return Values

None

Example

This example assumes that a vector of length *n* is distributed among the 16 nodes of the cube, and *x(m)* is a piece of the vector on a given node. *m* need not have the same value on all nodes (but for this example, we assume that it does). After computing the dot product and summing the global contributions, the vector must be normalized and then the entire vector must be available to all nodes, using **gcolx()** to collect the vector. The contents of the *xlens* array used by **gcolx()** is easy to calculate, since we assume *x* to be the same length on every node. The array *y* in each node then contains all of the vector *x*. The number of bytes returned in *y* is simply the sum of the elements in *xlens*. The order of *x*'s in *y* is by increasing node number. This example is simplified, and the way in which the norm is computed in this example can lead to problems with numerical underflow or overflow. Safer and more complicated implementations are possible and ought to be used in practice.

GCOLX() *(cont.)*

GCOLX() *(cont.)*

```
int i;
double x[m], y[n], dot, norm, dummy;
int dpsize = 8;
int num_nodes = 16;
long xlens[num_nodes],xlen;
                                    /* Assume that x, m, and n have been initialized */

                                          /* Compute the local DOT product */
dot = 0;
for (i = 0; i<m; i++)
    dot = dot + x[i]*x[i];

                                                /* Sum global contributions */
gdsum (&dot, 1, &dummy);
                                                /* Compute global NORM */
                                    /* Assume there exists a subroutine called dsqrt */
norm = dsqrt(dot);
                                                /* Do the local normalization */
for (i=0; i<m; i++)
    x[i] = x[i]/norm;

xlen = m*dpsize;
for (i=0; i < num_nodes; i++)
    xlens[i] = xlen;            /* get the length of all contributions into xlens */

gcolx(x, xlens, y);                                     /* Collect vector */
```

Environment

Node

Languages

C, Fortran

GCOLX() *(cont.)*

GCOLX() *(cont.)*

Description of Parameters

x	refers to the input buffer to be used in the operation. *x* may be of any type.
xlens	an array containing the length (in bytes) of the input buffer *x* expected on each node.
y	refers to the output buffer to be used in the operation (*y* contains the desired result). *y* must be of the same data type as *x*.

Discussion

Use **gcolx()** to globally collect and concatenate a contribution of specified length from each node in node number order. By providing the expected length of each contribution, the speed of this operation is greatly improved due to the reduced overhead of calculating where each contribution belongs in the output buffer. The position of each element in *xlens* corresponds to the node number from which the *x* parameter of the given length is expected. The *x* and *y* parameters can be of any data type as long as they are of the same data type. The result is returned in *y* to every node. All participating processes must have the same PID. This is a global operation, meaning that all nodes in the cube must issue this command before the execution of the process can continue.

If the lengths of the contributions from all the nodes is unknown, use the **gcol()** routine.

Errors

gcolx: Received message too long for buffer	Make sure buffer is long enough.
gcolx: Invalid buffer pointer	Specify a pointer which contains the address of a valid buffer.
gcolx: Invalid length	Use a non-negative number for *xlens*.
gcolx: Buffer length exceeds allocation	Make sure buffers are large enough and length parameters are not greater than buffer size.

GCOLX() *(cont.)*

GCOLX() *(cont.)*

See Also

gcol()

GCOLX() GCOLX()

Global concatenation operation for contributions of known length.

Synopsis

SUBROUTINE **GCOLX**(*x, xlens, y*)

Parameter Declaration7z

INTEGER *x(*)*
INTEGER *xlens(*)*
INTEGER *y(*)*

Return Values

None

Example

This example assumes that a vector of length *N* is distributed among the 16 nodes of the cube, and *X(M)* is a piece of the vector on a given node. *M* need not have the same value on all nodes (but for this example, we assume that it does). After computing the dot product and summing the global contributions, the vector must be normalized and then the entire vector must be available to all nodes, using **gcolx()** to collect the vector. The contents of the *XLENS* array used by **gcolx()** is easy to calculate, since we assume *X* to be the same length on every node. The array *Y* in each node then contains all of the vector *X*. The number of bytes returned in *Y* is simply the sum of the elements in *XLENS*. The order of *X*'s in *Y* is by increasing node number. This example is simplified, and the way in which the norm is computed in this example can lead to problems with numerical underflow or overflow. Safer and more complicated implementations are possible and ought to be used in practice.

GCOLX() *(cont.)*

GCOLX() *(cont.)*

```
      INTEGER I, DPSIZE, NUM_NODES
      DOUBLE PRECISION X(M), Y(N), DOT, NORM, DUMMY
      PARAMETER (DPSIZE = 8)
      PARAMETER (NUM_NODES = 16)
      INTEGER XLENS(NUM_NODES), XLEN
C     Assume that X, M, and N have been initialized
C     Compute the local dot product
      DOT = 0.0
      DO 1 I = 1, M
          DOT = DOT + X(I)*X(I)
1     CONTINUE
C     Sum global contributions
      CALL gdsum (DOT, 1, DUMMY)
C     Compute global NORM
      NORM = DSQRT(DOT)
C     Do the local normalization
      DO 2 I = 1, M
          X(I) = X(I)/NORM
2     CONTINUE
      XLEN = M*DPSIZE
C     Get the length of all contributions into XLENS.
      DO 3 I = 1, NUM_NODES
          XLENS(I) = XLEN
3     CONTINUE
C     Collect the vector
      CALL gcolx(X,XLENS,Y)
```

Environment

Node

Languages

C, Fortran

GCOLX() *(cont.)*

GCOLX() *(cont.)*

Description of Parameters

x	refers to the input buffer to be used in the operation. *X* may be of any type.
xlens	an array containing the length (in bytes) of the input buffer *X* expected from each node.
y	refers to the output buffer to be used in the operation (*Y* contains the desired result). *Y* must be of the same data type as *X*.

Discussion

Use **gcolx()** to globally collect and concatenate a contribution of specified length from each node in node number order. By providing the expected length of each contribution, **gcolx** improves the speed of this operation compared to **gcol()** due to the reduced overhead of calculating where each contribution belongs in the output buffer. The elements in *XLENS* are in increasing node number order. The *X* and *Y* parameters can be of any data type as long as they are of the same data type. The result is returned in *Y* to every node. All participating processes must have the same PID. This is a global operation, meaning that all nodes in the cube must issue this command before the execution of the process can continue.

If the lengths of the contributions from all the nodes is unknown, use the **gcol()** routine.

Errors

gcolx: Received message too long for buffer	Make sure buffer is long enough.
gcolx: Invalid length	Use a non-negative number for *XLENS*.
gcolx: Buffer length exceeds allocation	Make sure buffers are large enough and length parameters are not greater than buffer size.

GCOLX() *(cont.)*