intel™

PRODUCT RELEASE NOTES

Release 2.4

iPSC®/2 System Software

Intel Scientific Computers

Beaverton, Oregon

P/N: 311648

CONTENTS

Introduction

The iPSC/2 Product Release Notes explain the features, known limitations, and workarounds for Release 2.4 of the iPSC/2 system software. They provide procedures for installation of the software for the iPSC/2 system. The release notes also include an explanation of how to run iPSC/1 programs on the iPSC/2.

Features, Limitations, Restrictions and Workarounds

Release 2.4 of the iPSC/2 system software runs on all iPSC/2 system configurations. Some features and known limitations of this release are documented below:

A. 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.

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

3. ***Dup* call**
   Calling *dup* in a node process may not return the smallest file descriptor that is available.

4. **Opening files**
   A node process can open only twenty files at a time.

5. **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.

6. ***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.

7. **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.

8. **Print statement after an *irecv* may hang**
   If a node process posts an *irecv* using a type of -1, the next print statement may cause the process to hang. To recover, execute a *killcube* command.

9. **Node programs that call *killcube***
   If a node program kills itself by calling *killcube*, the next program executed on the node will receive the system response message for that *killcube* if the new process posts a receive with a type of -1.

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

B. Concurrent Workbench

1. ***Load* command**
   The *load* command does not report how much memory is used by a node process as it did for the iPSC/1.

2. ***Waitcube* and *killcube***
   The *waitcube* and *killcube* commands occasionally fail to return. To recover, press <DEL> to kill the command and then do a *bootcube*.

3. **Nodes marked unusable**
   If you get a message saying, "nn node(s) not responding, marked as unusable", where "nn" is the number of nodes, do a *bootcube* to reset the nodes. This message will only appear on the system console. If problems persist, use *cdp* to check for bad node boards. *Cdp* is described in Chapter 7 of the "iPSC/2 System Administrator's Guide."

4. **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.

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

6. **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*.

   Example 1:

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

   Example 2 (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);
```

7. **Releasing a cube**
   It is recommended that *killcube* be used prior to releasing the cube. *Relcube* alone does not always clean up properly, causing subsequent programs to fail. Using *killcube* avoids this problem.

8. **Number of nodes in subcube**
Getcube always allocates a number of nodes that is a power of two. If you specify a number of nodes that is not a power of two, *getcube* attempts to allocate additional nodes to make the number of nodes a power of two. If the cube with the next largest dimension is unavailable, no nodes are allocated. *Numnodes* reports the number of nodes actually allocated, not necessarily the number of nodes you requested.

9. **Node memory**
When the memory size is specified in *getcube*, the only guarantee is that nodes containing enough memory to satisfy the request will be allocated. Even when nodes containing the exact amount of memory specified are available, they might not be allocated by *getcube*.

   Example:

   Given that 1M, 4M, 8M, and 16M node boards are resident and available in the cube,

   getcube -td0m1 may allocate either the 1M, 4M, 8M, or 16M node.
   getcube -td0m4 may allocate either a 4M, 8M, or 16M node.

10. ***Killsyslog***
Calling *killsyslog* causes an extra blank line to appear in the host program output.

11. ***Setsyslog***
Calling *setsyslog* when the host program is already piping output through *syslog* causes the host program to hang.

12. ***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.

13. ***Load* and *killcube***
Calling *killcube* immediately after calling *load* may cause the *killcube* to hang. To recover, press <DEL> to kill the host process and then do a *bootcube*.

14. **"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.

15. ***Killcube* fails if *waitcube* is in the background**
If a *waitcube* is executing in the background, a *killcube* cannot be executed until the *waitcube* completes. Attempts to run both at the same time fail with the message "(host) setpid: Pid already in use."

16. ***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*.

17. ***Getcube* may allocate one or more SX nodes**
When you have a mixture of SX and 80387 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 80387 code on an SX node, or vice versa.

18. ***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*.

19. ***Relcube -F* switch**
There is now an *-F* switch for the *relcube* command. It can be executed by root only, and is used to force the release of all cubes. On the SRM, it releases all cubes allocated on that SRM. On the remote host, it releases all cubes allocated from that workstation.

20. **Shutting off cube occasionally causes SRM panic**
Shutting off the cube may cause the SRM to panic. To recover, power cycle the SRM.

21. **Outstanding *crecv* or *msgwait* will block any *hrecv***
An outstanding *crecv* or *msgwait* blocks any *hrecv* from being serviced.

22. **Cube allocated by *bootcube***
When *bootcube* is executed on an SRM, a 0 node cube named "iocube" is allocated. This is necessary for systems running the Concurrent File System™. This reduces the number of cubes that users can allocate from 10 to 9.

23. **Maximum of 10 outstanding *irecv*'s may be posted on an SRM**
A maximum of 10 outstanding *irecv*'s per subcube 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.

24. ***Cubeconf* not updated to show suspect boards**
When *bootcube* displays an error message indicating that a board is suspected of being bad, the *cubeconf* slot information should be changed to "FAILED", but is not. You should run *cdp* to determine what is wrong.

25. **Multiple *getcube* calls**
If you call *getcube* more than once in a host program, it will attach you to all cubes allocated in that host program, rather than just the last one allocated.

26. **SX and VX *cubetypes***
You may not mix upper and lower case letters when you specify SX and VX *cubetypes* in a *getcube* call or command. If you do, the SX or VX part of the *cubetype* descriptor is ignored.

C. Concurrent Workbench Assembler and Loader

1. **Directive incorrectly used**
The *.bss* assembler directive is incorrectly used by the assembler.
Workaround: Use the *.data* directive instead.

2. **Unsupported flags**
The **-R, -m, and -Y** flags are not supported in the assembler even though they are documented in the UNIX manuals..
3. **The assembler is undocumented**
As is only documented as *as(1)* in the UNIX manuals. The actual assembly instructions are undocumented

4. **Misaligned double precision variables**
The link editor occasionally misaligns double precision variables. This may cause unpredictable results when using the vector board.

D. 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 on the SRM is built with the AT&T *pcc* compiler rather than the new Green Hills compiler. The support libraries for the Green Hills compiler have been remade with the AT&T include files for compatibility with UNIX. 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. As part of the installation procedure, *gcc* will be linked to *cc* in the directory */bin*. The current version of *pcc* is in the same directory.

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

2. **Execution profiling support**
   Use the **-p** switch to generate calls for execution profiling on the SRM. The **-pg** switch is not supported by UNIX System V. It is a Berkeley UNIX feature.

3. **Range checking**
   Range checking is available with the **-C** switch.

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

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

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

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

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

6. **Type double variable alignment by the linker**
   The **-vx** switch does a better job of aligning variables of the type double on 8-byte boundaries than the linker does.

7. **C compiler -o flag format**
   A space is required following the **-o** compiler flag.

8. **Compiler -p switch**
   The compiler **-p** profiling switch causes an incorrect value to be placed in *argc*.

9. **Documentation Errors**

A. In the C Programmer's Reference Manual, the documentation for the *getcube* command incorrectly states that "*nohup getcube*" allows you to logout without releasing your cube. This should be "*nohup getcube &*".

B. In the C Programmer's Reference Manual, the documentation for the *relcube* command syntax is incorrect. It should read "*relcube [-c cubename | -a]*". Only one option may be specified.

C. In the C Programmer's Reference Manual, the messages displayed for system exceptions and faults now match those displayed by the *waitcube* command and *waitone* routine. They are as follows:

-256 Integer divide by zero
-255 Single-step interrupt
-254 Non-maskable interrupt
-253 Breakpoint interrupt
-252 Integer overflow exception
-251 Array bounds exception
-250 Invalid opcode
-249 Numeric processor not present
-248 Double fault
-247 Numeric processor overrun
-246 Invalid TSS fault
-245 Segment not present fault
-244 Stack overflow
-243 General protection violation
-242 Memory fault
-241 Vector processor error
-240 Floating point exception

E. 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.

Example:

INCLUDE '/usr/include/fcube.h

2. **GRAY, GINV, and CUBEINFO return integers**
The iPSC functions GRAY, GINV and CUBEINFO 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 arguments > 2**63

5. **-I2 compiler flag**
   The **-I2** compiler flag (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** flag be on both the compile and link commands. Modules compiled with the **-sx** compiler flag may not be linked with those compiled without the **-sx** flag and vice versa.

7. **-X87 compiler flag**
   The compiler flag **-X87** is now obsolete. Refer to the "Green Hills Fortran Language Reference Manual" for replacement switches.

8. **Invalid invocation line switches**
   Invalid switches presented to the compiler do not result in error messages, but are ignored.

9. **Execution profiling support**
   Use the **-p** switch to generate calls for execution profiling on the SRM. The **-pg** switch is not supported by UNIX System V, it is a Berkeley UNIX feature.

10. **Range checking**
    Range checking is available with the **-C** switch.

11. **VAX/VMS extensions**
    VAX/VMS Fortran extensions are available during both compilation and linking, using the **-vms** switch. These extensions include %loc, %val,WHILE, ENCODE/DECODE, and VAX/VMS carriage control. See the VAX Fortran User's Guide and Reference Manual for further information.

12. **Additions to the Fortran library**
    The following functions have been added to the Fortran library: *abort, rand, irand, srand,* and *signal*. They are documented in the Fortran Library support section of the AT&T UNIX System V Programmer's Reference Manual.

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

    The **-B** flag sets the following flags: **-g, -ga, -X9, -X18, -X39, -X168, -X211, -X219, -X230, -X307, -X356.**
    If a program works correctly with these flags, but incorrectly in more optimized versions, you should report a compiler bug to iSC.

14. **Misaligned data causes parity error with vector board**
    If you use the vector board, you may get a parity error under certain circumstances. This is due to a hardware problem with the 80386 node's iLBX-II interface and the Vortex iLBX-II interface.

To avoid this problem, you must ensure that all variables start on 32-bit boundaries. The Fortran compiler does this for all variables except character strings.

For example, 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 a 32-bit word.

15. **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.

16. **Running out of node board memory**
If you run out of node board memory while running a Fortran application, you may receive the message, "Fortran runtime error on external file "" (106): Buffer too large".

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

18. **-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.

19. **Compiler -p switch**
The compiler **-p** profiling switch causes an incorrect value to be returned by function IARGC.

20. **Linking with switches *-vec -vx* or *-vec***
Occasionally when linking with switches ***-vec -vx*** or ***-vec***, routines from *libf* (like *pow_ri*) will not be linked in.
Workaround: Use the switches ***-lvec -vec -vx*** or ***-lvec -vec*** instead.

21. **Documentation Errors**
    A. In the Fortran Programmer's Reference Manual, the documentation for the *getcube* command incorrectly states that "***nohup getcube***" allows you to logout without releasing your cube. This should be "***nohup getcube &***".

    B. In the Fortran Programmer's Reference Manual, the documentation for the *relcube* command syntax is incorrect. It should read "***relcube [-c cubename | -a]***". Only one option may be specified.

    C. In the Fortran Programmer's Reference Manual, there are several errors in the *gopf* routine example that make it unexecutable. These will be corrected in a future revision of the manual.

    D. In the Fortran Programmer's Reference Manual, the *gdlow* routine documentation makes several references to "***gdhigh***". These should all be references to "***gdlow***".

    E. In the Fortran Programmer's Reference Manual, the messages displayed for system exceptions and faults now match those displayed by the *waitcube* command and *waitone* routine. They are as follows:

        -256 Integer divide by zero
        -255 Single-step interrupt

-254 Non-maskable interrupt
-253 Breakpoint interrupt
-252 Integer overflow exception
-251 Array bounds exception
-250 Invalid opcode
-249 Numeric processor not present
-248 Double fault
-247 Numeric processor overrun
-246 Invalid TSS fault
-245 Segment not present fault
-244 Stack overflow
-243 General protection violation
-242 Memory fault
-241 Vector processor error
-240 Floating point exception

F. 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. **Underscore byte swapping calls**
   There are no byte-swapping routines in the remote host library with names prepended by an underscore.

3. **Maximum of seven cube partitions per remote host**
   The remote host software supports a maximum of seven cube partitions per remote host. The C and Fortran Programmer's Reference Manuals say that ten cubes are allowed.

4. **First *getcube* may fail**
   The first *getcube* after a *bootcube* may fail with a "(host) getcube: No SRM that matched your request was found" message. Subsequent *getcubes* succeed.

5. **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 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
   ```

6. **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>*

7. ***Getcube* displays name of SRM that allocated cube**
The remote host *getcube* command now displays the name of the SRM that allocated the cube.

8. ***rcc, rf77, rld, ras,* and *rar* no longer append ".*ipsc*" extensions to output filenames**
Remote development utilities *rcc, rf77, rld, ras,* and *rar* no longer append ".*ipsc*" extensions to their output filenames.

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

10. **Heavy message passing**
Heavy message passing between a remote host and the nodes for a sustained period (> 5 hours) may cause the remote *commser* process to die. To recover, execute a *bootcube* on the remote host as root and deallocate the cube on the SRM.

11. ***Calling getcube* and *newserver***
If you call *getcube* with a *keep* parameter = 0 in a host program then call *newserver*, you receive a "(host) file server: There is no attached cube" error message when the host program terminates. Ignore this message.

12. **SX and VX *cubetypes***
You may not mix upper and lower case letters when you specify SX and VX *cubetypes* in a *getcube* call or command. If you do, the SX or VX part of the *cubetype* descriptor is ignored.

13. ***Relcube -F* switch**
There is now an *-F* switch for the *relcube* command. It can be executed by root only, and is used to force the release of all cubes. On the SRM, it releases all cubes allocated on that SRM. On the remote host, it releases all cubes allocated from that workstation.

14. **Maximum of 12 outstanding *irecv*'s may be posted**
A maximum of 12 outstanding *irecv*'s per subcube may be posted on a workstation by a host program. If you attempt to post more than this, an error message displays and the process aborts. The maximum for the SRM is 10.

15. ***Setsyslog* call**
The *setsyslog* call fails and displays a "(host) setsyslog: Cannot start syslog process" error message.

G. DECON Concurrent Debugger

1. **-B compiler switch**
There is now a **-B** compiler switch for both Fortran and C which should be used when compiling programs for use with DECON. When using the **-B** flag, remove the **-O** and **-OLM** flags. Compiling object files for use with DECON may be done as follows:

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

or

cc -c -B host.c

2. **Loading node programs**
DECON must be in control when a node program is loaded so that it can generate symbol information. Therefore, you must comment out any *load* calls in your host program.

3. **Accessing Fortran variables declared in COMMON**
To access (i.e., *display* or *assign*) a variable declared in COMMON, you must indicate the name of the common block in which it is found using the following notation:

[common_name] . common_member

where *common_name* identifies the name of the common block. Omit *common_name* if it is a blank (un-named) common block. For example, given

COMMON /abc/ a(10)
COMMON // xyz (5)

you would use *abc.a(1)* to identify the first element in array *a* in named common *abc*, whereas, *.xyz(3)* would identify the third element in array xyz in blank common.

4. **Debugging a program on the SRM only**
If you are debugging a program only on the SRM, you should specify 0 nodes when doing a *getcube* before starting up DECON (e.g., *getcube* **-t0**). This will keep you from tying up nodes that you are not using.

5. **No remote DECON**
DECON only runs on the SRM. When using a remote work station, *rlogin* to the SRM to use DECON.

6. **Unsupported features**
Data breakpoints/tracepoints on host processes are not supported.
Label breakpoints/tracepoints on host and node processes are not supported.

7. **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.

8. ***Assign* command limitations**
An indirect assign to a subrange or entire Fortran character array does not work correctly. All elements of the array are concatenated and treated as a single unit.
Workaround: Use *assign* on each individual element of the array.

9. ***Break* command limitations**
When using the ***break when <condition>*** form of the *break* command, *<expr>* must be a simple integer value. Arithmetic expressions are unsupported.

10. ***Display* command limitations**
When displaying an entire Fortran character array, all of the elements are printed out in a single concatenated string.

11. ***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.

Step cannot be used to terminate a node program. If you attempt to step off the end of a node program, you will get the message "error: stopped at an unknown location xx."
Workaround: Use the *run* command to complete execution of a node program.

12. ***Type* command limitations**
The type of a Fortran *character*n* variable will just be 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*.

13. **Documentation errors in the DECON Concurrent Debugger Manual**
A. Page 3-8, remove " = " from the *assign* syntax.

The current line reads:

assign [<debug-context>] variable [= <expr>]

the line should read:

assign [<debug-context>] variable [<expr>]

B. Page 1-2, Item 2 is misleading. It should read:

The **-X18** compiler option turns off a portion of the compiler optimization phase.

14. **Interrupt while host process is doing a system call**
The system restarts a *csend*, *crecv*, *cprobe*, or *msgwait* system call if you resume a stopped host process that was previously interrupted while doing a system call.

15. **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.

16. **Input to a user program may be intercepted**
The input you are trying to send to your program might be intercepted by DECON if it is waiting for input at the same time. If this is the case, DECON will complain that an unknown command has been entered. Simply retype the input to your program.

17. ***Dimension* command returns number of nodes, not dimension of cube**
DECON's *dimension* command returns the number of nodes rather than the dimension of the allocated cube.

18. **Maximum pathname**
The maximum length of the source pathname is 32 characters.

19. ***Hload* documentation is incomplete**
The pid specified with the *hload* command must be the same as the *setpid* value used within the host program.

20. ***List* has -f switch**
The -f switch on the *list* command displays the name of the source file being listed.

21. ***File* command has been modified**
The command *file <fname>* sets the current source file to *<fname>*, which is then the file used by the *list* command. Either a *step* or a *run* command changes the current source file back to the one being executed.

22. **Interrupting and resuming host application**
Interrupting a host application and continuing its execution several times during a debugging session may hamper message passing. To prevent this, kill the host process occasionally. This clears all messages associated with the process from the system. If messages are not getting through at all, execute *bootcube* as root to clean things up.

23. **Erroneous "Process terminated" message**
After a node loads and terminates normally, if you load another program with the same pid, you receive the message "Process terminated". Ignore this message because it refers to the first process that was loaded.
Workaround: To prevent this message from appearing, execute a *killcube* between each load.

24. **Automatic execution of command file now available**
When you invoke DECON, it searches the current directory for the file name *.deconcf*, which it executes if found. This file should contain valid DECON commands that you want executed on entering DECON. The *.deconcf* file is very useful for setting up aliases for DECON commands.

H. 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 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 and/or the number of processes per node to conform to these limitations.

2. **NX handler call**
The NX *handler* call is not supported in the simulator.

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, the values from the clock are different from those obtained on the iPSC/2 system.

4. **Remote host and cube sharing**
Remote host and cube sharing commands are not supported by the simulator.

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**
UNIX/XENIX file descriptors 9, 10, 11, and 12 are reserved for the simulator.

7. **Signals**
Signal number 28 on UNIX 4.2 BSD 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/2 System does not. Therefore, it is possible to write programs for the simulator which will not run on the iPSC/2 System. Refer to Chapter 3 of the "iPSC/2 Programmer's Reference Manual" for more information.

9. **UNIX 5.2 ATT version**
To make and install UNIX 5.2 ATT version, type:

```
make att52
make instlatt
```

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

I. The Cube Diagnostic Program

1. **Invoking the Cube Diagnostic Program**
To invoke the Cube Diagnostic Program (*cdp*) at the SRM, log in as root and perform the following steps:

```
cd /usr/ipsc/diag
bootcube -D1-2
./cdp
```

2. **Invoking the Optional Board Cube Diagnostic Program**
To invoke the Optional Board Cube Diagnostic Program (*cdpo*) at the SRM, log in as root and perform the following steps:

```
cd /usr/ipsc/diag
bootcube
getcube -tdiag
./cdpo
```

3. **Aborting a test**
You can use the <DEL> key to abort *cdp* tests. *Cdp* ignores this key during diagnostic channel communications. Therefore, it may be necessary to hold the <DEL> key down until *cdp* recognizes it and the "Abort in progress" message appears on your terminal. If it becomes necessary to kill *cdp*, press <CTRL-\> .

J. UNIX

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

2. **Bad block mapper**
The bad block mapper allows entry of a maximum of 40 bad blocks. The bad block information must be typed in each time the disk is formatted.

3. **Cartridge Tape**
The cartridge tape occasionally hangs. To recover, the SRM must be turned off and turned back on.
Workaround: Avoid typing the kill character 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.

4. **Head command missing**
There is no *head* command on System V UNIX. There are several differences between Berkeley style UNIX and System V. The *less* command acts like the Berkeley *more* command and can be used to display the head of a text file.

5. **UNIX documentation error**
The procedure for recovering root's password in the "UNIX System Administrator's Guide" is incorrect. The correct usage is described in Chapter 5 of the "iPSC/2 System Administrator's Guide."

6. **Path for mail set incorrectly**
In the default *.cshrc* file, the path for mail is set incorrectly. The usual mail path is "*/usr/mail/$LOGNAME*" not "*/usr/spool/mail/$LOGNAME*".

7. ***Pcc* compiler *-p* switch**
The pcc compiler **-p** profiling switch causes an incorrect value to be placed in *argc*.

K. TCP/IP

1. **Ftp messages**
ftp> get <anyfile> issues unnecessary messages.

2. **Extraneous error message on boot**
Ignore the "xttyopen: no such device or address" message displayed during UNIX boot.

3. **Rlogin hangs**
Rlogin hangs if type ahead is used to enter a remote login name.

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	Free memory		NX/2 code		NX/2 data	Number of message buffers*		Message buffer size		Total node memory
1M	655,360	+	98,080	+	192,736	512	+	102,400	=	1,048,576
4M	3,768,320	+	98,080	+	200,928	634	+	126,976	=	4,194,304
8M	7,847,936	+	98,080	+	213,216	1,146	+	229,376	=	8,388,608

* Not part of node memory total

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 (rounded to 4,096)	overhead		
code (101,272):	102,400	+ 4,096	+	
data + bss (568,192):	569,344	+ 4,096	+	
stack and other:		12,288	=	692,224

This program would require a 4M node.

Compiling and Using iPSC/1 Programs on the iPSC/2

Source programs created for the iPSC/1 are not directly compatible with the current iPSC/2 software. We recommend that you convert your iPSC/1 code to iPSC/2 code rather than using the compatibility libraries for optimum performance. However, you may compile, link, and run iPSC/1 programs on the iPSC/2 if the following procedures are used.

1. Use the following procedure to compile and link your iPSC/1 applications with iPSC/2 software:

   To compile and link your C host program, type

   cc -o host host.c /usr/ipsc/lib/chost.a -host

   To compile and link your C node program, type

   cc -o node node.c /usr/ipsc/lib/Llibcnode.a -node

   To compile and link your Fortran host program, type

   f77 -o host host.f /usr/ipsc/lib/fhost.a -host

   To compile and link your Fortran node program, type

   f77 -o node node.f /usr/ipsc/lib/Llibfnode.a -node

2. Because the Fortran compiler has changed from the iPSC/1 (Ryan McFarland) to the iPSC/2 (Green Hills), the definition files for your Fortran programs have changed from *rmfhost.def* and *rmfnode.def* to *fhost.def* and *fnode.def*, respectively.

3. To use the *ccommutl* and *fcommutl* libraries, be sure to link them into your program.

   Example:

   f77 -o host host.f /usr/ipsc/lib/fcommutl.a /usr/ipsc/lib/fhost.a -host

4. Host programs must specify a process id to the system before any other calls to the system are made. Therefore, when using the iPSC/1 library, you should call *copen* with a process ID before

making any other calls to the library. Subsequent *copen* calls should use the same process ID. (Other process IDs will be ignored.)

5. In the iPSC/2 environment, the size of an integer variable is 4 bytes. This may impact your message length, because integers in C are 2 bytes in the iPSC/1 environment.

6. The *cubelog* command is no longer supported. To send output from your node and host programs to the same file, use the following:

   ```
   getcube > logfile
   host | syslog
   ```

   For more information on redirecting output, refer to the iPSC/2 C or Fortran Programmer's Reference Manual.

7. Some iPSC/1 commands are no longer supported, and many have changed. Refer to the iPSC/2 C or Fortran Programmer's Reference Manual for a description of the iPSC/2 commands. Refer to the "iPSC/2 System Administrator's Guide" for a description of *bootcube*.

Command Summary

iPSC/1	iPSC/2	Description
cubeinit	bootcube	Reset a cube.
cubelog	syslog	Send output to a logfile.
getcube	getcube	Get a cube.
load	load	Load a process into a cube.
load -c	bootcube	Download a new NX/2.
loadkill	killcube	Kill a cube process.
loadstart	startcube	Start a cube process.
loadwait	waitcube	Wait for a cube process to finish.

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 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 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 only works 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*, or 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 which can be used to gain slight additional speed, but at a cost which 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 *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.

Booting Standalone UNIX

To boot UNIX from the boot floppy, insert the floppy in the floppy drive and reset the CPU. When prompted with "Do you want to do a complete (destructive) install?", press <CTRL-\>. This will give you a single user shell prompt, "#".

To shutdown from single user mode, type "sync; sync; sync; uadmin 2 0".

Software Installation Procedures

Use the following procedures to reinstall all software on the hard disk. You only need to do this if the software becomes damaged and is unusable. We recommend you call Intel Scientific Computers first, if you believe your UNIX system needs to be rebuilt.

A. Installing UNIX V/386 R3.0

Use the following procedure to install UNIX V/386 R3.0 on your system. If you are reformatting your hard disk or recovering from a major disk crash, start with step 4. If you do not need to reformat your hard disk or are reinstalling UNIX V/386 R3.0 on a system that is currently running UNIX V/386 R3.0, proceed as follows:

1. Login as root

2. In addition to user files, verify that the following files have been backed up so that they may be reinstalled when the UNIX installation is complete:

   /etc/passwd

   /etc/group

   /etc/fstab

   /etc/inittab

   /usr/lib/uucp/Systems

   /usr/lib/uucp/Permissions

   /usr/lib/uucp/Devices

   /usr/lib/uucp/Dialers

   /usr/lib/uucp/Dialcodes

   /usr/lib/uucp/Maxuuscheds

   /usr/lib/uucp/Maxuuxqts

   /usr/lib/uucp/Poll

   Mail files kept in /usr/mail

   User defined cron entries in /usr/spool/cron

   vi save files in /usr/preserve

   Queued uucp files in /usr/spool/uucp

   Datafiles in /usr/spool/uucppublic

   Local news entries in /usr/news

   Nonstandard devices added to /dev

   Non-standard user programs and/or directory systems.

3. Remove the current Excelan software by typing

   /net/deinstall.exos

4. After confirming that all users are logged off, bring the system down by typing

   cd /
   shutdown

5. Insert the UNIX boot floppy into the floppy drive, and press <CTRL-ALT-DEL> to reboot the SRM.

6. If the system prompts for it, hit key F1 to continue reading the boot floppy.

7. When "boot:" appears on the screen, press return.

8. At the "Do you wish to do a complete (destructive) installation?" choose from the following:

   A. If you are reinstalling, type "n" (no). Next, you will be asked to confirm your disk partitioning. Confirm it and skip to step 16.

   B. If you want to reformat your hard disk, type "y" (yes) and continue.

   C. If you want to enter single user mode, press <CTRL-\>.

9. When asked:

   "Do you wish to proceed ...?", enter "y" (yes).

   "Do you wish to (re)format ...?", enter "y" (yes).

   "Do you wish to do a complete surface analysis...?", enter "n" (no).

   Ignore the message: "WARNING: invalid pdinfo on device 0."

10. You should have 15 heads and 917 cylinders configured for your system. When asked if this is correct, enter "y" (yes).

11. When asked, "Do you wish to allocate any of your disk to exclusive use by DOS?", enter "n" (no).

12. The disk allocation is UNIX starting at cylinder 1 for 916 cylinders. When asked if this is the partitioning that you want, answer "y" (yes).

13. At this point, as prompted, enter ALL of your bad track information. This may be obtained by calling iSC Customer Support at **1-800-421-CUBE**.

14. Choose a disk partition configuration for *swap*, *root*, and *usr*, specifying the following sizes for these partitions:

    swap 109 tracks
    root 121 tracks
    usr the remainder

    The *usr* partition is automatically assigned the remaining tracks.

15. At this point, the installation script will finish formatting and partitioning the drive. This takes 50 to 60 minutes. Ignore any mount warning messages.

16. The boot floppy installation script will finish copying a minimal UNIX system to the hard disk. When it is finished, it will print:

    "Reset the CPU to reboot"

17. Remove the boot floppy and press <CTRL-ALT-DEL> to reboot the SRM.

18. When "boot:" appears on the screen, press return.

19. When asked if you are installing from tape, enter "y" (yes).

20. Insert the UNIX V/386 R3.0 tape into the tape drive firmly so that the black hook on the left swings into view and engages the tape cartridge. Press return when the tape is ready.

21. The tape will be read onto the hard disk. This takes about 20 minutes. When it is finished, it will print:

    "Reset the CPU to re-boot"

22. Remove the tape by pushing it in, and then press <CTRL-ALT-DEL> to reboot the SRM.

23. When "boot:" appears on the screen, press return.

24. At the "Console login:" prompt, log in as *setup* and answer the questions regarding the system name and passwords.

25. At the next "Console login:" prompt, the SRM is ready for use.

26. Reinstall the files you backed up in Step 2.

27. Reinstall your iPSC and TCP/IP software.

B. Installing iPSC/2 Release 2.4

Before you install this software, verify that the following iPSC/2 processes are not running:

cdemon
commser
fserver
lifeline
loader
rcam
ripscd
tocube
tohost

If any of these processes is running, kill it as follows:

1. Login as root.

2. Type

   bootcube -D1-1

3. Check to see if any of the processes are still running, by typing:

   ps -ef

4. If any are still running, you must explicitly kill them by typing:

   kill -9 *pid*

   where *pid* is the process id obtained by executing the **ps** command (see step 3.).

5. Check again to see if any of the processes are still running, by typing:

   ps -ef

6. If any are still running, you must reboot your SRM, as follows:

   cd /
   shutdown
   <CTRL-ALT-DEL>

Use the following procedure to install iPSC/2 software on the hard disk. It takes approximately 25 minutes to complete this installation.

1. Login as root.

2. Type

   cd /

3. Insert the tape labeled "iPSC/2 System Software R2.4."

4. To extract the software from the tape, (this step takes approximately 5 minutes to complete), type

   star xv

5. Run the **install** script by typing

   ./install.ipsc

6. If you would like the iPSC/2 Simulator installed, answer "y" (yes) when asked.

7. You are asked if you would like to generate a new UNIX kernel. If you have not installed TCP/IP, answer "n" (no). Otherwise, answer "y" (yes).

9. Wait for the install script to install the software. It takes approximately 15 minutes to complete the installation.

10. If you wish to install TCP/IP, you should do so at this time.

11. If you generated a new UNIX kernel, you should reboot your SRM at this time.

NOTE: If users on your system will only be using either the SX node library (*/usr/lib/libsxnode.a*) or the standard node library (*/usr/lib/libnode.a*), move the unused library to another location or rename it so that users will not inadvertently link in the wrong library. Loading an SX application onto an non-SX node will cause strange behavior.

After you install the iPSC/2 software, edit cube configuration file */usr/ipsc/conf/cubeconf* to make the keyword values correct for your system configuration:

- The value of "cardcages" should be **1**, **2**, or **4**, depending on the number of chassis in your system.
- The value of "slots" should be **32**, **64**, or **128**, depending on the number of node and vector boards in your system.
- The value of "baud" should not be changed.
- The value of "backplane_rev" is only used by *cdp*. If it is set incorrectly, *cdp* fails with the message "DCM revision is incompatible with backplane revision". If this occurs, change the value of "backplane_rev" from **B** to **A**.

Slot configuration information is filled in when you execute *bootcube*. If a warning message displays indicating that file */tmp/cubeconf* was used when the cube was booted, move */tmp/cubeconf* to */usr/ipsc/conf/cubeconf*.

C. Installing TCP/IP

Use the following procedure to install the TCP/IP Ethernet software. Refer to the "Excelan User's Manual" for an explanation of the installation instructions. It takes approximately 35 minutes to complete the installation.

1. Login as root.

2. Type

   cd /
   shutdown -iS

3. After the system is in single-user mode, indicated by the message

   "INIT: SINGLE USER MODE"

   mount the */usr* disk partition by typing

   mount /dev/dsk/0s3 /usr

4. If Excelan has been previously installed (fully or partially), be sure to deinstall it first, by typing

   /net/deinstall.exos

5. Insert Excelan disk 1 of 2 and type

   cpio -iduvBc < /dev/fdhi

6. Insert disk 2 of 2 and type

   cpio -iduvBc < /dev/fdhi

7. Run the Excelan install script by typing

   /net/install.exos

8. Answer "n" (no) to "Do you wish to use the default settings (y/n)?"

9. Type

<ENTER>	to the first four defaults
<2><ENTER>	for the *init* state that networking will start

10. If you want the Excelan ***help*** files integrated into the UNIX ***help*** facility, answer "y" (yes) to install the on-line ***help*** for the networking commands.

11. Answer "y" (yes) to install the new UNIX kernel.

12. When the install script is finished, edit your */etc/hosts* file to describe your local network.

13. Type

    cd /
    shutdown

14. If the Excelan board is not installed, power down the computer and install the Excelan board.

15. Type <CTRL-ALT-DEL> to reboot the SRM.

D. Installing Remote Host

The remote access feature of the iPSC/2 extends the message-passing environment across the network to supported workstations. Certain Concurrent Workbench tools are also visible from the workstation. Release 2.4 supports only the Sun-3 workstation.

1. Create a directory on the Sun-3 workstation to contain the remote host source code. This directory can be located wherever you choose.

2. Copy the remote host source code from the directory */usr/ipsc/src/rhost* on the SRM into the directory created for the remote host source code on the Sun.

3. Edit the makefile in the top level remote host source directory on the Sun so that the installation directories are correct for your system.

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 */usr/ipsc/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.

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

INCDIR--Contains iPSC/2 include files *cube.h* and *fcube.h*. The default is */usr/ipsc/include* .

NOTE: All pathnames must be absolute pathnames (i.e., start with "/").

4. To build binaries on the Sun, *cd* to the top level remote host source directory and type

   make

5. To install the binaries in the prescribed directories, login to the console as *root* and *cd* to the directory containing the remote host sources. Then type

   make install

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

Running Remote Host Software

To use the cube from the Sun-3, you must start a remote *commser* on the Sun. To start a *commser*, you execute *bootcube* on the remote host. You must run this as *root*.

Once the remote *commser* is running, you can get cubes and run cube applications.

Compiling Remote Host Applications

To compile existing iPSC/2 SRM applications, the makefiles must be modified. The changes are as follows:

1. If **LIBDIR** is either */lib* or */usr/lib*, replace the **-host** switch with **-lhost.** Otherwise, the entire path of the host library must be specified.

   For example: If **LIBDIR** is defined to be */usr/myusername/lib*, you must link with */usr/myusername/lib/libhost.a*.

2. Change the node compilation rule for *cc* and *f77* to *rcc* and *rf77*. Also change *ld*, *ar* and *as* to *rld*, *rar* and *ras*.

NOTE: The Sun and the cube have a different internal representation of integer and floating point numbers. See the description of the CTOH utilities in the iPSC/2 C or Fortran Programmer's Reference Manual for information about converting messages to and from cube byte order.