October, 1986
Order Number: 310104-003

iPSC HYPERCUBE SIMULATOR

intel Corporation

REV.	REVISION HISTORY	DATE
-001	Original issue	12-09-85
-002	Revision	06-01-86
-003	Revision	10-15-86

CONTENTS

CHAPTER 1

OVERVIEW

INTRODUCTION

The hypercube simulator is a software program which simulates the actions of the Intel iPSC. It allows you to develop your iPSC programs in a controlled environment prior to execution on the actual hypercube. Use of the simulator significantly speeds up the program development cycle and provides facilities for locating program errors. However, large cube programs will run much more slowly on the simulator than on the actual hypercube.

DESCRIPTION

The simulator package consists of a simulator program and a set of libraries. The simulator program runs on any of the following systems:

Hardware	Software
Intel System 310	XENIX Operating System
PC/AT	XENIX Operating System
DEC VAX 11/780	UNIX 4.2 BSD Operating System
Pyramid, Sun, etc.	UNIX 4.2 BSD Operating System

The simulator executes hypercube programs written in either the C language or Ryan McFarland FORTRAN. The simulator libraries simulate the actions of the iPSC Node Executive (NX) routines which are invoked by your programs.

Hypercube programs need not be modified to run on the simulator, with the exception of Ryan McFarland FORTRAN (INTEGER*2 should be set as default; refer to the next section). However, the simulator supports a somewhat smaller configuration than does the actual hypercube (refer to next section).

Cube manager programs are started within the simulator and not from the XENIX shell, as they would be with the actual cube manager.

The simulator provides an interactive interface which simulates the iPSC's cube manager commands. You can start a simulation by issuing these commands to the simulator. In addition, cube manager commands can be placed in a script file and directed to the simulator's standard input on the command line.

The simulator performs interrupt handling. You can interrupt your simulated hypercube's process (or processes), check its status, and then restart the simulator from where you left off...without ever exiting the simulator.

LIMITATIONS

- **Process Creation** — XENIX and UNIX limit the number of processes that the simulator can create and thus limit the size of a simulation. XENIX allows 14 processes to be created, and UNIX 4.2BSD allows 23 processes to be created.

- **Cube Dimension** — The simulator creates one UNIX process for every simulated node or host process. You must limit the cube's dimension and/or the number of processes per node to conform to these limitations.

- **NX Handler** — The NX ***handler*** command is not supported by the simulator.

- **Clock Calls & 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 ***very different*** from those obtained on the actual hypercube.

- **Cube Manager Programs** — Cube manager programs are started within the simulator. Arguments cannot be supplied to cube manager programs and their standard input and output cannot be redirected on the command line.

- **Integers** — Ryan McFarland FORTRAN integers default to INTEGER*4; the hypercube simulator interface requires INTEGER*2.

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

- **Signals** — Signal number 28 on UNIX 4.2 BSD and signal number 16 on XENIX are reserved for the simulator.

- **Huge Model** — The simulator does not support huge model.

- **Interchanging Calls** — Unlike the real iPSC system, host and node type calls are valid in both host and node programs. Therefore, it is possible to write programs for the simulator which will not run on a real system.

CHAPTER 2

INSTALLING THE SIMULATOR

If you receive the simulator on magnetic tape or diskette, then you must install the simulator prior to using it. This section explains how to do this.

When the simulator is supplied on tape or diskette separate from the complete hypercube system, the simulator comes in source form. The tape or diskette contains the following files:

bflib.c	implementation for UNIX FORTRAN NX interface
chost.def	definitions for C language host programs
cnode.def	definitions for C language node programs
rmfhost.def	definitions for FORTRAN language host programs
rmfnode.def	definitions for FORTRAN language node programs
gendef.h	specification for general definitions
hslib.h	specification for "C" version of NX library
hslib.c	implementation for NX library
pipemgt.h	specification for pipe management
pipemgt.c	implementation for pipe management
pcsmgt.h	specification for process management
pcsmgt.c	implementation for process management
qmgt.h	specification for queue management
qmgt.c	implementation for queue management
reqmgt.h	specification for request management
reqmgt.c	implementation for request management
rmfort_c.f	implementation for RM/FORTRAN NX interface on XENIX
sim.c	implementation for simulator
xrmflib.c	implementation for "C" part of RM/FORTRAN NX interface on XENIX
Makefile	make file for the simulator

The simulator library and program are created by compiling and linking the source files. A makefile, called ***Makefile***, is provided to assist in compiling and linking the simulator using the UNIX/XENIX ***make*** program.

The makefile will generate either the UNIX 4.2BSD or the XENIX version of the simulator and libraries. The source code generates the XENIX version when a definition for XENIX is provided. This is normally accomplished as an option to the compiler. Otherwise, the source code generates the UNIX 4.2BSD version.

The makefile must be edited according to the version to be made. This is accomplished as follows. Two definitions of the make variables CCFLAGS and LDFLAGS are provided. Comment out the definitions for the version ***not*** to be made, as indicated within the makefile. Note that the XENIX version of CCFLAGS provides a definition for XENIX. In either version, you can also supply definitions for CASSERT and DEBUG. Their effects are described in the "*Hypercube Simulator Internal Product Description.*"

The versions are selected respectively using the make targets *bsd* and *xenix*. To make the UNIX 4.2BSD version, move to the directory containing the simulator's source files and type:

```
make bsd
```

To make the XENIX version, type:

```
make xenix
```

Note that both the "C" and "FORTRAN" libraries are generated for each version. For more information, refer to the "*Hypercube Simulator Internal Product Description.*"

In summary, installing the simulator is a 3-step process:

1. Copy the source files into the desired directory.
2. Edit the "Makefile" in order to select the compiler and loader flags appropriate for your target system.
3. Enter the proper ***make*** command.

CHAPTER 3

PROGRAM PREPARATION

INTRODUCTION

This chapter describes how to prepare programs for execution on the simulator. Both C and FORTRAN programs should be written exactly as they would be for execution on the actual hypercube (refer to the *"iPSC Program Development Guide"* and the *"iPSC Programmer's Reference Manual"*).

Definitions for the node executive commands are provided in Appendix A of this manual. These definitions follow those described in the *"iPSC Programmer's Reference Manual"* except that the FORTRAN type INTEGER is used instead of INTEGER*2. This difference affects only UNIX 4.2 BSD simulations.

It is important that you observe that an integer is 16 bits long on the Intel System 310 and 32 bits long on the VAX. Programs which are executed on the VAX based simulator (or other 32-bit systems such as the Pyramid) may behave differently than on the XENIX based simulator and hypercube where this distinction is significant. For example, when integers are passed in a message, it is important to know the size of an integer in order to correctly compute the message size. This distinction especially affects FORTRAN programmers because 32-bit integers are used in the VAX environment whereas the user must use 16-bit integers in hypercube library calls on the 310 using Ryan McFarland FORTRAN. RM FORTRAN integers default to INTEGER*4 but should be forced to INTEGER*2 on the System 310 for the simulated node executive (NX) routines.

NOTE

The simulator expects integer parameters to hypercube library calls to be of type INTEGER*2. If, by some chance, they are of type INTEGER*4 and the magnitude of the value being passed requires over two bytes of storage, the value received by the simulator is automatically set to zero.

In order to avoid conflicts with the simulator, adhere to the following programming conventions:

- Do not use the XENIX/UNIX file descriptors 9, 10, 11, and 12 because these descriptors are used by the simulator library.
- Do not use signal number 28 on UNIX 4.2 BSD or signal number 16 on XENIX. These are used by the simulator.

COMPILING AND LINKING PROGRAMS

This section describes how to compile and link programs for execution on the simulator. The System 310/XENIX and VAX/UNIX 4.2BSD environments are discussed separately.

A set of definition files are provided as an aid to you. These files contain the external declarations for the NX calls used in hypercube programs. These files are contained in directory /usr/ipsc/lib/sim and closely match the actual hypercube definition files. If you wish to use these definition files. include the appropriate file in the source code module. Make sure that you use these definition files rather than the real ones. The files are described in the next paragraph.

NOTE

In these examples, and throughout this manual, it is assumed that you have installed the simulator in the directory: /usr/ipsc/lib/sim (as suggested in the installation instructions in the Internal Product Description). However, remember that this may vary from site to site.

Files

After the simulator has been installed, it includes the following files. These files are typically in the directory: /usr/ipsc/lib/sim.

chost.def	definitions for C language host programs
cnode.def	definitions for C language node programs
rmfhost.def	definitions for FORTRAN host programs
rmfnode.def	definitions for FORTRAN node programs
rmfort_c.o	XENIX simulator RM FORTRAN interface module
xsimlib.a	XENIX simulator library for C and RM FORTRAN programs
bsimlib.a	UNIX 4.2BSD simulator library for C and FORTRAN programs
xsim	the XENIX simulator program
bsim	the UNIX 4.2BSD simulator program
LOGFILE	the default log file

NOTE

Of the above files, only the definition files are supplied on the media. The others are created when you "make" and run the simulator (refer to Chapter 2, "*Installing The Simulator*").

Preparing System 310/XENIX Programs

XENIX based C programs should use the large memory model, which is specified with the option -Ml. The following commands should be used when preparing C host and node programs for the simulator. It is assumed in these examples that there is a single source file named xxx.c. If there are several modules in your application, simply add them in where the xxx appears.

```
cc -Ml -c xxx.c
cc -Ml -o xxx xxx.o /usr/ipsc/lib/sim/xsimlib.a -lx
```

If you are using math functions, then add the suffix "-lm". Thus,

```
cc -Ml -o xxx xxx.o /usr/ipsc/lib/sim/xsimlib.a -lx -lm
```

The following commands should be used when preparing Ryan McFarland FORTRAN host and node programs for the simulator. It is assumed in this example that there is a single source file named xxx.f. If there are several modules in your application, simply add them in where the xxx appears.

```
rmfort xxx.f
cc -Ml -o xxx xxx.o  /usr/ipsc/lib/sim/rmfort.o
                     /user/ipsc/lib/sim/xsimlib.a -lx -lf
```

Preparing VAX/UNIX 4.2 BSD Programs

UNIX 4.2 BSD based C and FORTRAN programs are prepared in a straightforward manner. The following commands should be used when preparing C host and node programs for the simulator. It is assumed in these examples that there is a single source file named xxx.c. If there are several modules in your application, simply add them in where the xxx appears.

```
cc -c xxx.c
cc -o xxx xxx.o /usr/ipsc/lib/sim/bsimlib.a
```

The following commands should be used when preparing FORTRAN host and node programs for the simulator. It is assumed in these examples that there is a single source file named xxx.f. If there are several modules in your application, simply add them in where the xxx appears.

```
f77 -c xxx.f
f77 -o xxx xxx.o /usr/ipsc/lib/sim/bsimlib.a
```

CHAPTER 4

USING THE SIMULATOR

INTRODUCTION

This chapter describes how to use the hypercube simulator. Once you start the simulator, you have twelve different commands available to you.

Start the simulator by typing:

```
xsim   (on XENIX systems)
```

or

```
bsim   (on the VAX)
```

Once started, the simulator will give you the following prompt:

```
iPSC sim>
```

At this point, you can use the simulator commands.

COMMANDS

Most of the commands at your disposal simulate the commands which you type in at the cube manager's console to control the cube, while a few commands are unique to the simulator.

When using a command, you can either enter the name or an abbreviation (the simulator expects both to be in lower case). On the following pages, the command ***name*** is in bold type on the left side of the page....the ***abbreviation*** is in bold type on the right side of the page. Options to commands are indicated with brackets ([ ]) and alternatives are indicated with a vertical bar (|).

These commands are briefly mentioned below and described in detail on subsequent pages:

Command	Abbreviation	Description
cubelog	log	manages the simulator's log file
cubeman	m	loads programs into simulated cube manager
help	h or ?	prints summary of simulator commands and their use
load	ld	simulates dynamic loader's "LOAD" command
loadkill	lk	simulates dynamic loader's "LOADKILL" command
loadstart	ls	simulates dynamic loader's "LOADSTART" command
loadwait	lw	simulates dynamic loader's "LOADWAIT" command
quit	q or exit	terminates simulation and exits the simulator
start	s	starts the simulation after cube and host processes are loaded
status	st	checks the status of an application. It has the following switches: -a = all (message, process, request, and traffic status) -m = message status -p = process status -r = request status -t = message traffic
system	!	passes commands through the simulator to the shell
trace	t	enables or disables tracing of node operating system (NX) calls

cubelog

log

This command simulates the cube manager's ***cubelog*** command and is used to manage the simulator's log file. It has the following syntax:

cubelog [-n] [-l{ *log__filename, stdout}*]

Options:

[-n]	an option that specifies that the system log file is not to be used any longer. Events can be logged again to the default log file by typing ***cubelog*** without any arguments.
[-l {*log__filename, stdout}*]	an option that specifies that the simulator is to maintain a log in the specified log file. If no log file name is supplied, then the default log file, called LOGFILE, is used. The simulator supports a special log file named stdout which causes all logging output to be directed to the standard output, which is, by default, sent to the user's terminal. Simulated processes use the log file name that is in effect at the time they are loaded.

cubeman

m

This command loads programs into the simulated cube manager. It has the following syntax:

cubeman [-H] *object__file*

Options:

[-H]	wait for a ***start*** command to start the process; do not start process immediately. In either case, with or without "-H", the actual hypercube simulation will not run until the "start" command is entered. If the "-H" option is not used, the process will block at the first call to a hypercube library routine. The "-H" option is very useful for interactive host programs. By delaying the execution of the host (via "-H") until the simulation is started, you avoid the race condition of both the simulator interface and host program competing for input from the terminal (stdin).
object__file	pathname of the executable program to be loaded. Arguments cannot be supplied to ***object__file*** within the ***cubeman*** command as they would be from the shell in the actual cube manager.

This command can be executed multiple times to load more than one program into the simulated cube manager. With or without the "-H" option, simulation will start only after a "start" command has been entered. Until then, no hypercube library routines can be executed by the cube manager process.

help — h or ?

This command prints a summary of the simulator's commands and their use in order to give you quick assistance. Its syntax is:

help

load — ld

This command simulates the dynamic loader's ***load*** command and is used to load programs into the simulated hypercube. It has the following syntax:

load [-c dim] [-p pid] [-H] [node...] file

Options:

[-c dim]	cold start switch is ignored ("dim" is the dimension of your simulated hypercube). The dimension of the simulated hypercube is initialized to 3 when the simulator is invoked (via bsim or xsim). Unlike the real dynamic loader, the simulator's load command does not supply a default value for "dim." If you use the "-c" option, you must also specify a "dim."
[-p pid]	"pid" is the process id to be assigned to the new process being loaded. If "-p" is not used, pid will be zero.
[-H]	wait for a ***loadstart*** or ***start*** command to start the process; do not start process immediately. In either case, with or without "-H", the actual hypercube simulation will not run until the ***start*** or ***loadstart*** is entered. If the "-H" option is not used,the process will block at the first call to NX.
[node...]	the numbers of the node(s) to be loaded. If no node number is specified, all nodes in the simulated hypercube of "dim" dimension are loaded.
file	the pathname of the file to be loaded into the simulated hypercube.

This command may be executed multiple times to load more than one program into the simulated hypercube. The hypercube simulator will accept the ***load*** command's other arguments (-S, -F, -T,- R, -b, -M, -Q, -K) but will ignore them. Note that processes loaded with the ***load*** command must have unique process id's in every node. Be aware also that with the simulator there is no need to "load" the dynamic loader. Therefore, "load" commands entered without a filename result in error messages.

loadkill lk

This command simulates the dynamic loader's ***loadkill*** command and lets you kill processes "running" in your simulated hypercube. It has the following syntax:

loadkill [-p pid] [node...]

Options:

[-p pid]	"pid" is the process id of the process you want killed. If "-p" is not used, all processes will be killed.
[node...]	the number(s) of the node(s) in which processes are to be killed. If no node number is specified, all "pid" processes on all nodes are killed.

If no parameters are supplied, then ***loadkill*** kills all simulated node and cube manager processes and cleans out all messages awaiting delivery.

loadstart ls

This command simulates the dynamic loader's ***loadstart*** command and is used to start processes which have been loaded via "***load***" with the "***-H***" flag and starts the simulation running. Note that ***loadstart*** does not start processes loaded with ***cubeman***. The command has the following syntax:

loadstart [-p pid] [node...]

Options:

[-p pid]	"pid" is the process to be started. If "-p" is not used, all processes will be started.
[node...]	the number(s) of the node(s) in which processes are to be started.

If too many processes have been previously loaded by using ***load*** or ***cubeman*** , you will be notified at this time. (Refer to "Limitations" in Chapter 1 of this manual.)

loadwait lw

This command simulates the dynamic loader's ***loadwait*** command. It starts the simulated node processes and waits for a process to complete. If you use ***loadwait*** with no arguments, it will wait for a process to complete anywhere in the simulated cube. After the process completes, ***loadwait*** prints the following message:

Node xx PID yy completed, code xx (no meaning)

Unlike the real system, the return code for ***loadwait*** is always zero and has no significance.

The ***loadwait*** command has the following syntax:

loadwait [-p pid] [node...]

Options:

[-p pid]	"pid" is the process id of the specific process to wait for. If "-p" is not used, ***loadwait*** waits for any process.
[node...]	the number(s) of the node(s) in which processes are to be found. If no node number is specified, ***loadwait*** waits for the specified process in any node.

quit q or exit

This command lets you terminate the simulation and exit the simulator. The simulator removes all cube and host processes before exiting. The command's syntax is:

quit

start s

This command is used to start the simulation after all cube and host processes have been loaded via ***cubeman*** or ***load***. It has the following syntax:

start

The actual XENIX/UNIX processes which were previously loaded by ***cubeman*** are created only after ***start*** is invoked if the "-H" option is used. If too many processes have been previously loaded using ***cubeman*** or ***load***, you will be notified at this time. (Refer to Chapter 1 for "Limitations.")

status

st

This command lets you check the status of your application. A common usage is to load the simulated hypercube and start your process running, then press the interrupt key and enter the ***status*** command. At this point, you can peruse the various status tables that are available. The ***status*** command has the following syntax:

status [-a] [-m] [-p] [-r] [-t]

Options:

[-a]	all status tables are displayed
[-m]	display only message status
[-p]	display only process status
[-r]	display only request status
[-t]	display only message traffic status

If no option is supplied, then only the process status is displayed.

system

!

This command lets you pass commands through the simulator to the shell, the same way you can use ":!cmd" in vi. It has the following syntax:

system cmd parameters

Options:

cmd parameters	can be any command allowed in the UNIX/XENIX shell.

Note that the the abbreviated command "!" can be used with or without a space between the "!" and the command.

trace

t

This command enables or disables the tracing of hypercube library calls. When enabled, the simulator puts a message in the log file whenever you execute a NX command. This helps you trace the progress of the simulation and locate bugs. Trace messages are listed at the end of this chapter. This command has the following syntax:

trace [on | off]

Options:

[on | off]	an option that allows you to turn tracing on or off. Tracing is initially off. If neither on nor off is supplied, then the simulator reports whether tracing is currently enabled or disabled.

OTHER FACILITIES

In addition to the above commands, you can hit the interrupt key during a simulation. This causes the simulation to pause. Simulation is resumed by re-entering the ***start*** (***start*** or ***loadstart***) command.

The simulator's commands can be placed into a script file which is supplied to the simulator's standard input. For example, the following commands could be placed into a script file:

```
load nodeprog
cubeman hostprog
cubelog -l stdout
trace on
start
```

where nodeprog and hostprog are executable programs. This file would then be used to invoke the simulator, as follows:

```
/usr/ipsc/lib/sim/xsim <script >simout
```

Note that the log file's output, including tracing output, is saved in the file simout in this example.

ERROR MESSAGES

The following error messages are generated by the simulator's user interface. They are directed to the standard error, which is by default sent to the user's terminal. The symbol <string> indicates that a string is placed in the message at this location.

"sim: error in opening log file <string>"
"sim: unrecognized command: <string>"
"sim: missing parameters"
"sim: missing node number after -n"
"sim: illegal value for node: <string>"
"sim: illegal dimension: <string>"
"sim: node specified not compatible with dimension"
"sim: missing file name"
"sim: file <string> not found"
"sim: too many processes; start failed."
"sim: missing filename after -l"
"sim: unknown option: <string>"
"sim: illegal argument: <string>; use 'on' or 'off'."
"sim: missing process id after -p"
"sim: illegal value for process id: <string>"
"sim: missing dimension after -c"
"sim: missing argument after -S"
"sim: missing argument after -F"
"sim: missing argument after -T"
"sim: missing argument after -M"
"sim: missing argument after -b"
"sim: missing argument after -Q"
"sim: missing argument after -K"
"sim: illegal load option <string>"
"sim: no processes to start"
"sim: process to be started not found"
"sim: too many processes; load failed."
"sim: process in use; load failed"
"sim: PID <string> not loaded"
"sim: unknown user command"
"sim: user error occurred"
"sim: internal error 1"
"sim: internal error 2"
"sim: internal error 3"
"sim: internal error 4"
"sim: internal error 5"
"sim: internal error 6"
"sim: internal error 7"
"sim:internal error 8"
"sim:internal error 9"

If you receive an "internal error" message, then call ISC Customer Support for assistance.

The following error messages are generated when errors are detected in the use of the host and NX routines. These messages are sent to the log file. The format for the messages is:

mm/dd/yy: hh:mm NODE: nn PID: pp UPID: uu : <string>

where:

mm/dd/yy	the current date
hh:mm	the current time
nn	the node which generated the error message
pp	the process which generated the error message, as specified when you open a channel
uu	the UNIX process which generated the error message
<string>	one of the message strings shown on the following page. In the message strings, an asterisk (*) indicates that the error applies to all commands with the given prefix. For example, send* refers to the send, sendw, and sendmsg commands.

The messages themselves are listed on the following page.

The error messages are:

"send*: illegal channel"
"send*: channel not open"
"send*: not process's channel"
"send*: illegal type"
"send*: illegal msg length"
"send*: illegal target node"
"send*: illegal process id"
"send*: no buffer space available"

"recv*: illegal channel"
"recv*: channel not open"
"recv*: not process's channel"
"recv*: illegal type"
"recv*: illegal buf length"

"status: illegal channel"
"status: channel not open"
"status: not process's channel"

"probe: illegal channel"
"probe: channel not open"
"probe: not process's channel"
"probe: illegal type"
"probemsg: illegal process id"

"copen: illegal process id"
"copen: no channel available"

"cclose: illegal channel"
"cclose: channel not open"
"cclose: not process's channel"

"load: illegal process id"
"load: illegal target node"
"load: path name is too long"
"load: file name not found"
"load: process id is in use"

"lkill: illegal process id"
"lkill: illegal target node"

"lwait/all: illegal process id"
"lwait/all: illegal target node"
"lwait: selected process not found"
"lwait/all: no buffer space available for request"

TRACE MESSAGES

Trace messages are sent to the log file when tracing is enabled. The format for these messages is:

mm/dd/yy: hh:mm NODE: nn UPID: uu : <string>

where:

mm/dd/yy	the current date
hh:mm	the current time
nn	the node which generated the trace message
uu	the UNIX/XENIX process which generated the trace message
<string>	one of the trace messages, as shown below

In the trace messages, asterisk (*) indicates that the message applies to all commands with the given prefix. For example, send* refers to the send, sendw, and sendmsg commands. The symbol <int> indicates that an integer is placed in the message at this location.

The trace messages are:

"send*: for chan = <int> type = <int> tnode = <int> tpid = <int>"
"send* (cont'd): deliver msg to waiting receiver"
"send* (cont'd): receiver not waiting; msg enqueued"

"recv*: for chan = <int> type = <int>"
"recv* (cont'd): msg not found; holding req"

"status: for channel <int>"

"cubedim"

"mynode"

"probe: for chan = <int> type = <int>"
"probe: msg of length <int> found for chan = <int> type = <int>"

"probemsg: for pid = <int>"
"probemsg: msg of length <int> found for pid = <int>"

"copen: for process <int>"
"cclose: for channel <int>"

"clock"

"load: for process <int>, node <int>"
"load (cont'd): path name is <string>"
"lkill: for node <int>, process <int>"
"lwait/all: for node <int>, process <int>"

APPENDIX A

NODE EXECUTIVE ROUTINES

Routines for the cube manager (host) interface library for C and FORTRAN are listed in Tables A-1 and A-2, respectively.

Routines for the node interface library for C and FORTRAN are listed in Tables A-3 and A-4, respectively.

Refer to either the iPSC Program Development Guide or the iPSC Programmer's Reference Manual for a more detailed description of these commands.

Table A-1
Cube Manager Interface Library for C
Calling Summary

Procedure	Calling Sequence
CCLOSE	`cclose(ci);`
COPEN	`ci = copen(pid);`
CUBEDIM	`dim = cubedim();`
LOAD	`load("filename", node, pid);`
LKILL	`lkill(node, pid);`
LWAIT	`lwait(node, pid, &cnode, &cpid, &ccode);`
LWAITALL	`lwaitall(node, pid);`
MYPID	`pid=mypid( );`
PROBEMSG	`status=probemsg(pid);`
RECVMSG	`recvmsg(ci, &type, buf, len, &cnt, &node, &pid);`
SENDMSG	`sendmsg(ci, type, buf, len, node, pid);`
SYSLOG	`syslog(pid, "desired message");`

Table A-2
Cube Manager Interface Library for FORTRAN
Calling Summary

Procedure	Calling Sequence
CCLOSE	CALL CCLOSE(CI)
COPEN	CI = COPEN(PID)
CUBEDIM	DIM = CUBEDIM()
LOAD	CALL LOAD('FILENAME', NODE, PID)
LKILL	CALL LKILL(NODE, PID)
LWAIT	CALL LWAIT(NODE, PID, CNODE, CPID, CCODE)
LWAITALL	CALL LWAITALL(NODE, PID)
MYPID	PID=MYPID( )
PROBEMSG	STATUS=PROBEMSG(PID)
RECVMSG	CALL RECVMSG(CI, TYPE, BUF, LEN, CNT, NODE, PID)
SENDMSG	CALL SENDMSG(CI, TYPE, BUF, LEN, NODE, PID)
SYSLOG	CALL SYSLOG(PID, 'DESIRED MESSAGE')

Table A-3
Node System Interface Library for C
Calling Summary

Procedure	Calling Sequence
CCLOSE	`cclose(ci);`
CLOCK	`millsecs = clock();`
COPEN	`ci = copen(pid);`
CUBEDIM	`dim = cubedim();`
FLICK	`flick();`
GREENLED	`greenled(state);`
MYNODE	`node = mynode();`
MYPID	`pid = mypid();`
PROBE	`result = probe(ci, type);`
RECV	`recv(ci, type, buf, len, &cnt, &node, &pid);`
RECVW	`recvw(ci, type, buf, len, &cnt, &node, &pid);`
REDLED	`redled(state);`
SEND	`send(ci, type, buf, len, node, pid);`
SENDW	`sendw(ci, type, buf, len, node, pid);`
STATUS	`result = status(ci);`
SYSLOG	`syslog(pid, "desired message");`

Table A-4
Node System Interface Library for FORTRAN
Calling Summary

Procedure	Calling Sequence
CCLOSE	CALL CCLOSE(CI)
CLOCK	MILLSECS = CLOCK()
COPEN	CI = COPEN(PID)
CUBEDIM	DIM = CUBEDIM()
FLICK	CALL FLICK()
GREENLED	CALL GREENLED(ISTATE)
MYNODE	NODE = MYNODE()
MYPID	PID = MYPID()
PROBE	RESULT = PROBE (CI, TYPE)
RECV	CALL RECV(CI, TYPE, BUF, LEN, CNT, NODE, PID)
RECVW	CALL RECVW(CI, TYPE, BUF, LEN, CNT, NODE, PID)
REDLED	CALL REDLED(ISTATE)
SEND	CALL SEND(CI, TYPE, BUF, LEN, NODE, PID)
SENDW	CALL SENDW(CI, TYPE, BUF, LEN, NODE, PID)
STATUS	RESULT = STATUS(CI)
SYSLOG	CALL SYSLOG(PID, 'DESIRED MESSAGE')