Contents

1 Introduction

The 16-node Intel iPSC/860 is a prototype scalable parallel supercomputer, with 128 million bytes (megabytes or Mbytes) of distributed memory and a theoretical peak speed of 0.96 billion floating point operations per second (gigaflops or GFLOPS), using 64-bit arithmetic. The iPSC/860 at the Department of Computer Science of the University of Vienna (hereafter referred to as UniVie) is a research tool which has been purchased by the Austrian Center for Parallel Computer for experimental research in the field of support tools for parallel programming. Hence access to the iPSC/860 is strictly limited. See section 2.1 for details on obtaining an account on the iPSC/860 and guidelines for utilizing it.

The iPSC/860 is managed by the System Administrator at the Department of Computer Science.

1.1 About This Manual

Much of the contents of this *INTEL iPSC/860 User's Guide* are based upon a preliminary manual prepared by the *Central Scientific Computing Complex* at NASA Langley Research Center. We gratefully acknowledge their kind permission to reprint the relevant sections.

This manual attempts to provide a concise description of relevant information that is necessary for an individual to access and effectively utilize the iPSC/860. Since this is a brief introduction, many topics are not covered in great detail. However, there is a reference to more detailed information in most sections. Users of the iPSC/860 are expected to have experience with UNIX, consequently only significant differences are discussed.

This document is also being maintained on **pixy** in the directory **/usr/info** and in **/global/doc/ipsc** on the SUN Workstations.

We welcome corrections and comments from users. Descriptions of commands are necessarily brief; for a full description, please refer to the Intel vendor manuals. Section 9 of this manual contains a complete listing of Intel manuals that may be ordered directly from Intel. Copies of these manuals are also available at UniVie.

1.2 Characteristics of the Intel IPSC/860

The Intel iPSC/860 is a multiple-instruction, multiple-data (MIMD) parallel computer. Applications for the Intel iPSC/860 should be designed such that the major computational kernels of the program are executed simultaneously by each assigned processor. Processors (and their associated memory) are allocated in groups by a power of two. Not all programs may demonstrate improved performance from the iPSC/860's novel architecture. Most applications are not completely parallelizable and may require the user to consider load balancing, internodal communication strategy, as well as other parallel optimization techniques before achieving improved performance. The following sections give a brief overview of the iPSC/860's hardware, software, performance and style of programming.

1.2.1 Hardware

The iPSC/860 at UniVie consists of 16 Intel 80860 (i860) RISC chips that run at 40 MHz. They are connected via a hypercube communications network. Each of the i860 single chip microprocessors contain approximately one million transistors, 8 Mbytes of local memory, a vector 64-bit floating point unit, and an on-chip 3-D graphics processor. In addition, each i860 has a 4 thousand byte (kilobyte or Kbyte) code cache and an 8 Kbyte data cache, which is large enough to handle loops with up to 1000 instructions. The i860 has 3 floating point vector pipelines for integer math, floating point addition, and floating point multiplication. These vector pipes may operate concurrently. Chaining is possible between the floating point adder and multiplier.

Processors communicate with each other by passing messages, circuit switched through intermediate nodes. Each computational node has a separate communications coprocessor to coordinate message passing. Message passing is necessary because each node is an independent processor with its own memory. Since the nodes do not share memory, they must communicate with each other to pass the program's data, which is done via synchronous or asynchronous message passing calls. All nodes are fully connected with a hypercube interconnect.

The i860-based processing nodes on Intel hypercubes are generally referred to as RX nodes. We adhere to this convention in the manual.

The UniVie system also includes two I/O nodes, based on the Intel 386 chip. These provide the computational nodes with access to the Concurrent File System (CFS) which has 4 disks with a total capacity of approximately 2.3 Gbytes. A single CIO Ethernet interface and its associated service node provide a back-end network connection for CFS file transfer and iPSC/860-based networking applications.

1.3 Software

Each of the 16 Intel iPSC/860 computational nodes run a specialized operating system called NX, which implements a subset of UNIX. NX provides message passing capability, memory management and process management. The message passing calls range from a simple, effective set of synchronous calls to advanced asynchronous calls that allow message passing and process overlapping as well as interrupt-driven message handling. A node has access to the host file system and the CFS. Management of the CFS takes place outside of the NX kernel, and is handled by the I/O nodes which run additional processes for this purpose.

1.3.1 Performance

The 40 MHz i860 chip has a theoretical peak 64-bit floating point performance of 60 MFLOPS when chaining is used. Thus the peak theoretical performance of a 16-node system is 0.96 GFLOPS. In practice it is difficult to even approach this rate, even using assembly code. A more realistic performance figure is the 17 MFLOPS per computational node quoted by Intel for running an optimized LINPACK suite. However, the single node performance rate for all FORTRAN application codes on the Intel system is usually not more than 3-5 MFLOPS.

1.3.2 Programming Paradigm

In order to make effective use of the iPSC/860, an application has to be designed to execute in a parallel fashion. An application may consist of a host and node program or just a node program. A host program runs on a Sun workstation, such as **edwin**, or the System Resource Manager (SRM), **pixy**. The node program runs on the iPSC/860.

Each node of the cube executes its own program. Usually each node executes the same program on different sets of data, but sometimes there is conditional code that causes special actions to be performed on one or more nodes. This corresponds roughly to the **S**ingle **P**rogramming **M**ultiple **D**ata or **SPMD** style of programming.

Normally, the user interface is separated from the computational portion of the code and is handled by the host program or a designated node via conditionally executed code. Each node program can determine if it is running on the designated node by testing its node number. The designated node number is usually node zero.

2 The Computing Environment

Users do not have direct login access to the Intel iPSC/860. The current operating environment requires a separate host machine as front-end for the iPSC/860. Users of the iPSC/860 at UniVie may have an account on the special front-end computer attached to the iPSC/860, **pixy**, or on both a SUN SPARCstation in the local network and the special front-end computer. **pixy**, an Intel SYP301, is also referred to as the System Resource Manager (**SRM**). When we discuss usage of the Intel iPSC/860 in the sections below, we will refer to the computers where a user may have an account by the names **pixy** and **edwin**. **edwin** is to be understood as a generic name for the SPARCstations, and should be implicitly replaced by the name of the SUN workstation in the local network on which you are working. A SUN SPARCstation may be used as a remote host for the iPSC/860.

pixy and **edwin** both run derivatives of the UNIX operating system. They are used for functions such as compiling, loading and editing. The iPSC/860 is used solely as a computational engine. Note that storage space for files is very limited on **pixy**.

Because the iPSC/860 is a research machine, access to the machine is limited. The next section discusses account validation and guidelines for using it to insure an equitable distribution of resources.

2.1 Getting Started on the Intel iPSC/860

The iPSC/860 is not intended for the novice programmer. Users are expected to be well-versed in UNIX, FORTRAN and/or C. Many of the utilities and tools are in early stages of development, so the programming environment is not as robust as that found on mainframe computers. Intel provides considerable user documentation, as described in Section 9. The machine is not a freely available computing center resource; each application for an account will be decided upon at the discretion of the Department of Computer Science, UniVie. Members of the ACPC will be given preference, as will research projects involving experiments with the system. This system will not, in general,

be used to run large applications. You must justify your request for an account, giving an outline of your proposed research area, as well as follow some basic guidelines for system etiquette.

2.1.1 Accounting and Validation

At this time, no formal accounting is done for iPSC/860 usage. Usage is tracked, but there is no charge-back mechanism. Also there are no quotas on CPU use or permanent file storage. As utilization of the machine increases, it may become necessary to implement one or the other of these quotas. Currently, the mechanism for controlling usage is peer pressure.

Users must be validated to use the iPSC/860. This validation provides an account on **pixy**; users may sometimes be granted an additional account on **edwin**. External accounts other than those for the ACPC member groups are disabled at the beginning of each academic year. To continue using the system, all users must be revalidated.

To become validated or revalidated for the iPSC/860, obtain the *iPSC Account Application* form from the system administrator at **martin@par.univie.ac.at**. Complete this and return it to the address listed on the the form. ACPC members performing experimental system work are given accounts automatically on request. Other ACPC members and those interested in doing research into parallel processing may be given accounts, based on their description of anticipated use of the system. Users local to the Department of Computer Science, UniVie, are expected to have a valid account in the local network.

2.1.2 Guidelines for System Use

The guidelines for using the system are very basic: be a good neighbor, protect your password and don't share accounts. Rules for selecting a password are given in Section 2.2.3. One common error that new users make frequently involves interactive use of the iPSC/860. Interactively, you must explicitly release your processors or other users are blocked from accessing the system. Section 3.1.1 describes the **getcube** and **relcube** commands.

It is requested that you inform the system administrator well in advance if you require the entire 16 nodes for a large block of time. Disk space is limited for the iPSC/860 system. Local users are encouraged to retain their programs on **edwin**. External users with large storage requirements should indicate this in their request, and should apply for an account on **edwin** as well.

2.2 Using The Intel System

The remote host and SRM are accessible only via the Internet. There are no directly connected dial-in lines.

pixy has the Internet number **131.130.70.33**. The **iphost** is called **cube** and has the Internet number **131.130.70.34**.

2.2.1 Logging into the Local Network

When you get connected to **edwin** the system responds with the login window. Type your login name followed by a carriage return. Then enter your password at the next prompt.

If you type either your login name or password incorrectly, the system prompts you again. If you hit the backspace in an attempt to correct an error, your login attempt fails. If you are not able to login, send e-mail to **martin@par.univie.ac.at**.

2.2.2 Logging into the System Resource Manager (SRM)

When you get connected to **pixy**, the system responds with

System V.3.2 UNIX (pixy)

and then prompts for your login name with

login:

Type your login name followed by a carriage return. The carriage return is followed by a prompt for your password:

password:

If you type either your login name or password incorrectly, the system prompts you again. If you hit the backspace in an attempt to correct an error, your login attempt fails. If you are not able to login, send e-mail to **martin@par.univie.ac.at**.

2.2.3 Changing Your Password

If you haven't been given a password by the System Administrator, you have to enter one as your first action. Use the **passwd** command to do this.

Otherwise you must change your initial password when you first login to **pixy** or **edwin** (or both, if you have home directories on both machines). You should also change your password on both machines at least once a year. Your new password must meet the following security requirements:

- It must be at least six characters long.
- It must have at least two alphabetic and one numeric or special character.
- It must not be any permutation of your login name.
- It must differ from your old password by at least three characters.

Also, words found in a dictionary with only a single digit appended to the end or added at the beginning to form the password are highly susceptible to being compromised and

should not be used. The command to change your password is **passwd**. It prompts you for your old password and twice for your new password. If the two entries for your new password don't match, the system does not change your password and prompts you again to enter your new password. If you forget your password, contact the System Administrator for assistance.

2.2.4 Your .login and .cshrc Files

This section has sample **.login** and **.cshrc** files that are similar but not identical to the skeleton files that the system administrator gives you initially. If you modify the default files or bring your **.login** and **.cshrc** files from another machine and discover that something isn't working, then you can copy the default **.login** and **.cshrc** files from the directory **/etc** to your home directory.

The **.login** file is a file of commands and environment variables that is automatically executed every time that you invoke the C-shell.

The default **.login** file can be found on **pixy** in the **/etc** directory. The System Administrator keeps these files up to date, and announces any changes made. However, users are still encouraged to check them periodically and make any necessary changes to their setup files.

An example of a typical **.login** file in the Intel host environment is given below. Everything to the right of the pound signs (#) is a comment to explain the various entries.

```
#
# Commands here are executed for a login shell only.
#
setenv SHELL /bin/sh                    # Use sh to run scripts
setenv TTY `tty`
setenv HZ 100
setenv TERMCAP /etc/termcap             # terminal data base
setenv TZ `grep TZ= /etc/TIMEZONE | sed s/TZ=/' '/`
                                        # needed for csh to know TIMEZONE
if ( `tty` == "/dev/console" ) then
        setenv TERM at386-m
else
        setenv TERM vt100
endif
```

The **.cshrc** file is similar to **.login** but it is automatically executed every time that you log into or spawn a new C-shell.

The default **.cshrc** file can be found on **pixy** in the **/etc** directory. The system administrator keeps these files up to date. However, users are still encouraged to check them periodically and make any necessary changes to their setup files.

An example of a typical **.cshrc** file in the iPSC/860 host environment is given below. Everything to the right of the pound signs (#) is a comment to explain the various entries.

```
# BMC 11/91
# Default .cshrc file for C-Shell accounts.
# Commands here are executed each time csh starts up.
#
set prompt="‘hostname‘[‘logname‘]: "
set ignoreeof                           # don’t let control-d logout
set LOGNAME=‘logname‘                   # get login name for mail boxes
set history=50                          # save last 50 commands

setenv IPSC_XDEV /usr/ipsc/XDEV         # for man pages on SRM
setenv IPSC_UTIL /usr/superb/ipsc_util
setenv VFCS_UTIL /usr/superb/vfcs_util
setenv MANPATH "$IPSC_XDEV/i860/man"

set path=(. /bin /usr/bin $home/bin /usr/i860/bin  $IPSC_XDEV/i860/bin \
$VFCS_UTIL/new_bin  )    # search path
set cdpath=". .. ~"

# alias declarations
alias x      chmod +x
alias +w     chmod go+w
alias -w     chmod go-w
alias l      ls -lsai

set mail=(30 /usr/mail/$LOGNAME)        # mailbox location for csh
setenv MAIL /usr/mail/$LOGNAME  # mailbox location for environment
```

2.3 User Files

User files are NFS mounted between **pixy** and **edwin**. Users' home directories may reside on either **pixy** or or **edwin** or both of these. Users may also read/write CFS files; they may be accessed by node programs. Node programs read/write standard FORTRAN files from/to **pixy** or **edwin**.

2.3.1 UNIX File System

pixy runs UNIX System V, Release 3.2 operating system and contains the standard file system associated with this particular version of UNIX. **edwin** runs the latest SUN OS (currently 4.1.3) and includes the standard file system associated with this particular version of UNIX.

2.3.2 The Concurrent File System

The Concurrent File System (CFS) provides fast simultaneous access to secondary storage for the nodes. Large data files can be written to or read from the CFS. The sytem installed

at UniVie includes two I/O nodes to access these. It also includes a tape drive which may be mounted onto the CFS to enable high speed transfer of data between the tapes and the computational nodes via the I/O nodes. The following example illustrates the use of the CFS.

```
       PROGRAM CFSTEST
C
C Program to illustrate the use of the CFS
C
       CHARACTER*16 MSG
       CHARACTER*20 MSGBUFFER
       PARAMETER (MSG='Hello from node ')
       MODE = 1
       IUNIT = 10
C
C ME is the node number
C
       ME = MYNODE()
C
C Open unit 10 on CFS
C
       OPEN(UNIT=IUNIT, FILE='/cfs/yourname/cfs0', STATUS='UNKNOWN',
1       FORM='UNFORMATTED')
       CALL SETIOMODE(IUNIT, MODE)
C
C Construct the message (CHAR(10) is Line Feed)
C
       WRITE(MSGBUFFER, 5) MSG, ME, CHAR(10)
5       FORMAT(A16, I3, A1)
       MSGLEN = LEN(MSGBUFFER)
       DO 20 I=1,3
          CALL CWRITE(IUNIT, MSGBUFFER, MSGLEN)
20     CONTINUE
       CLOSE(IUNIT)
       STOP
       END
```

The first thing to notice in the example is the file name for the CFS file. The file name begins with **/cfs** to distinguish it from ordinary files which reside on the SRM. The second feature from this example is the use of the routine SETIOMODE to set the I/O mode. The parameters for SETIOMODE are unit number and mode. CFS provides four I/O modes.

Mode 0: Each node has an individual file pointer, and each may write to any part of a file at any time. If two nodes write to the same place in the file, the data from the second node overwrites the data from the first.

Mode 1: There is a common file pointer for all nodes, and all file operations are performed on a first-come, first-serve basis. This file may not have a consistent order from run to run.

Mode 2: There is a common file pointer with file operations done in order by node number. Buffers may be of variable length.

Mode 3: There is a common file pointer with file operations done in order by node number. Unlike mode 2, which allows variable length buffers, the mode 3 buffers are of fixed length. For fixed length buffers, all nodes can read and write at once.

The last feature included in the example is the use of the CWRITE routine. CWRITE is used for high speed synchronous output to a CFS file. Since the output is synchronous, the program waits until the write operation is complete. Other routines for reading and writing to a CFS file include:

```
CALL CREAD(UNIT, BUF, LEN)

ID = IREAD(UNIT, BUF, LEN)
ID = IWRITE(UNIT, BUF, LEN)
```

The CREAD routine is the companion to CWRITE for synchronous writing, and the IREAD and IWRITE functions are the read and write routines for asynchronous I/O. For all these routines, UNIT is the unit number, BUF is the buffer where the data read or written is stored, and LEN is the size of the buffer in bytes. The return value for the asynchronous operations is an id used by the routines IOWAIT and IODONE. The IOWAIT routine is used to wait for completion of an asynchronous operation with the syntax:

```
CALL IOWAIT(ID)
```

The IODONE routine returns the status of an asynchronous I/O operation. The syntax of IODONE is:

```
IR = IODONE(ID)
```

where IR is zero if the read or write operation is not finished. The return value is 1 if the operation is complete.

Output from the example might look like this in mode 0, corresponding to the case where node 1 happened to finish first, with the output from node 0 overwriting the node 1 output:

```
Hello from node 0
Hello from node 0
Hello from node 0
```

In mode 1, there is no synchronization between the nodes while performing the writes and the output might look like this:

```
Hello from node 1
Hello from node 0
Hello from node 1
Hello from node 1
Hello from node 0
Hello from node 0
```

The output from modes 2 and 3 look alike, but mode 3 offers greater performance since in mode 2 each node must wait for the other node to complete its operation:

```
Hello from node 0
Hello from node 1
Hello from node 0
Hello from node 1
Hello from node 0
Hello from node 1
```

The Concurrent File System is also available for C programs and is described in the *iPSC/2 and iPSC/860 User's Guide*.

2.4 X Window System on the Intel

A set of X Window System client libraries (Version 11 Release 4.0) is available for the iPSC/860 system. The X Window System is also referred to as X or X11. Your workstation must have the X server software to be able to use the client libraries. The name or Internet address for which you want to give access to your workstation is the name or Internet address of the **iphost**, not the SRM (see Section 2.2). Applications using X must be written in C. In order to compile an X application, you must include certain switches on the **cc** command line.

-i860	compiles the code to run on RX nodes
-I/usr/include/ipsc	specifies the correct path for X11 include files
-node	compiles the code to be executed on iPSC nodes

To link the X applications, you must link in the client libraries of your choice with one or more link switches as shown in the *iPSC/2 and iPSC/860 User's Guide*. The preferred method of defining resources is to use the command **xrdb** to download resources to the X server. This method is documented in the X server documentation for your workstation. If not downloaded, X primarily looks for resource information in the following directory:

/usr/ipsc/lib/X11/app-defaults

User specific resource definitions should be located in the file:

$HOME/.Xdefaults

Only the node calling **XOpenDisplay** has a connection to the server. Since there is no default X Window System server running on the iPSC/860 system, ensure that the environment variable **DISPLAY** is set on your workstation via

setenv DISPLAY edwin:0.0

where **edwin** stands for the name of your workstation.

3 The Runtime Environment

The runtime environment consists of a set of system commands issued from the operating system and a set of system calls available to host and node programs. This environment provides utilities for running a concurrent application on the Intel: allocating a cube, loading one or more programs on the nodes, running the node processes, and then deallocating the cube. This section introduces both the system commands and system calls and provides examples using both sets.

3.1 System Commands

The iPSC/860 system commands provide cube allocation, manage node processes, redirect node output, and log host output. The iPSC system commands belong to the iPSC system UNIX extensions and are issued at the UNIX prompt. These commands may also be issued from within user programs; this topic is discussion in Section 3.2.

3.1.1 Allocating and Releasing Cubes

Before you can load programs onto the nodes, you must allocate a cube. Use the **getcube** command to allocate a new cube, name it, and make it the current cube. The basic syntax of the **getcube** command and some of the most useful options are listed below:

getcube [**-t cubetype**] [**-c cubename**]

where **-t cubetype** allows you to specify the size and type of your cube. Specify size first followed by type in a contiguous string. The only valid cube type on our iPSC system is **rx**. Without this option, you get the largest available cube.

The option **-c cubename** allows you to allocate multiple cubes by assigning a unique name to each cube as you get it. Valid names are any ASCII character string of 15 characters or less. Without this option, the cube is automatically assigned the name **defaultname**.

After executing a program, you must release the cube. The **relcube** command allows you to release one or all cubes previously allocated with the **getcube** command. The basic syntax of the **relcube** command and some of the most useful options are listed below:

relcube [**-c cubename** | **-a**]

The option **-c cubename** allows you to release a specific named cube. Names are assigned with **getcube**. For a list of cube names, use the **cubeinfo** command. If the specified cube does not exist, an error is returned. Without this argument, the command releases the currently attached cube. If you have not allocated a cube, an error is returned.

The **-a** option allows you to release all of the cubes that you own on the system from which you invoke **relcube**.

To display the list of allocated cubes use the **cubeinfo** command. With no arguments, the **cubeinfo** command returns information about the current attached cube. The other options provide information about other allocated cubes as described. The basic syntax of the **cubeinfo** command and some of the most useful options are listed below:

cubeinfo [**-a** | **-s**]

The **-a** option returns information about all the cubes that you own on the system from which you invoked the command. The **-s** option gets information on all the cubes on the system from which the command was executed. If executed on an SRM, it returns information on all cubes allocated from that SRM either directly or via remote workstations. If executed on a remote development machine, it returns information on all cubes that have been allocated from that workstation.

You should always release **all** your cubes at logout; this is not done automatically. To ensure this, construct a *.logout* file which includes the command **relcube -a**.

3.1.2 Input/Output with the Host

The redirecting input and output system commands provide a method of altering the standard output and standard error of the host and nodes. Two commands are available. The **syslog** command sends the output of the host process to the file server handling I/O from the nodes. The **newserver** command starts a new file server for the specified cube. A simple alternative which will often suffice is given in 3.1.4 below.

3.1.3 Loading and Executing a Program

After you've allocated a cube, the next step is to load one or more processes onto the cube to run. To load a node process, use the **load** command. This command puts the specified file onto every node of the current cube and assigns an NX process ID (pid) of 0 to the node process. Each node process starts running as soon as it is loaded.

For example, if you have an executable called **hello**, you can allocate a cube and load **hello** on each node of that cube as follows.

```
pixy[user]: getcube -t4
getcube successful: cube type 4m8rxn4 allocated
pixy[user]: cubeinfo
CUBENAME       USER    SRM     HOST    TYPE     TTYS
defaultname    user    pixy    pixy    4m8rxn4  ttyp02
```

```
pixy[user]: load hello
hello world from node 0 process id 0.0000000
hello world from node 1 process id 0.0000000
hello world from node 2 process id 0.0000000
hello world from node 3 process id 0.0000000
pixy[user]: relcube
relcube released 1 cube
pixy[user]: cubeinfo
(host) cubeinfo: There is no attached cube
```

On **pixy**, see the directory **/usr/ipsc/examples/f/hello** for the FORTRAN source code for the hello example (and **/pixy/ipsc/examples/f/hello** on **edwin**). The directory **/usr/ipsc/examples/c/hello** contains the C source code for the **hello** example.

A host program is a typical UNIX executable which may run on either **pixy** or **edwin** and may perform computations, serve as a user interface, optionally perform cube allocation/deallocation as well as node program loading and total execution management. For example, if you have an executable node program called **pi** and a host program called **host**, you can allocate a cube and execute the host program. This example loads the **pi** program on each of the nodes from the host program and prompts the user for input.

```
pixy[user]: getcube -t4
getcube successful: cube type 4m8rxn4 allocated
pixy[user]: cubeinfo
CUBENAME       USER    SRM     HOST    TYPE     TTYS
defaultname    user    pixy    pixy    4m8rxn4  ttyp02
pixy[user]: host
... {Results from the program execution}
pixy[user]: relcube
relcube released 1 cube
pixy[user]: cubeinfo
(host) cubeinfo: There is no attached cube
```

On **pixy**, see the directory **/usr/ipsc/examples/f/pi** for the FORTRAN source code for the **pi** example, and the directory **/usr/ipsc/examples/c/pi** for C source code for the **pi** example. On **edwin**, the directories begin with **/pixy/ipsc...**.

The **killcube** command kills the specified processes and flushes messages related to those processes. It is not an error to use **killcube** to kill a nonexistent process. It is recommended that you use the **killcube** command to kill any remaining cube processes before you release the cube.

3.1.4 Using A Script Shell

One of the simplest ways of controlling execution of your program, in particular if you do not have a host program running on **pixy** or **edwin**, is to write a simple script shell.

The file **script1** below, for example, will allocate a cube with 16 nodes and load the (compiled) node program *nodep* onto each of them. The nodes will read their input from

the file *infile* on the host machine executing the script. The cube is released when the program terminates. The file **script2**, on the other hand, allocates 8 nodes on the system and redirects *all* output from the program to the file *outfile* on the machine where the script is executed (either **pixy** or **edwin**).

```
pixy[user]: more script1
getcube -t16 ; load nodep; waitcube <infile; relcube

pixy[user]: more script2
getcube -t8 > outfile; load nodep; waitcube <infile; relcube
```

3.1.5 NX Commands on the Nodes

The NX operating system running on the computational nodes of the Intel hypercube may be accessed directly by the user. However, before a command may be issued, a login script must be executed. If you wish to do this, you should create a login and logout script, and a shell script for using NX as follows:

```
pixy[user]: more .login.ipsc
# Sample .login.ipsc
set prompt="node%"
echo 'Entering node shell...'
echo 'Commands in Concurrent Programming Vol. 2'

pixy[user]: more .logout.ipsc
# Sample .logout.ipsc
echo 'Terminating node shell...'

pixy[user]: more .cshrc.ipsc
# Sample .cshrc.ipsc
set shell = /usr/ipsc/bin/csh
set path = (/usr/ipsc/bin /usr/ipsc/XDEV/i860/bin)
```

Examples for the most recent login files are kept in **/usr/info** on **pixy**. You can just copy them to your home directory.

A list of the NX commands, which is very restricted, is to be found in the *iPSC/860 Concurrent Programming Vol. 2*. We do not discuss them further here.

3.1.6 Troubleshooting the Remote Host Software

The remote host software is used to work with the iPSC/860 from your SUN Workstation. All the above-mentioned commands work in the same way from **edwin** as if your are logged into **pixy**. There is a daemon running on all the SUNs that is talking to a similar daemon on **pixy**. However, this daemon is not very stable on the SUNs and often hangs. You realise this situation if the only answer you get to your requests is **Commser not responding**. In this case, there is a simple command that restarts the daemon, called

rebootcube. So if you have trouble getting a connection to **pixy**, you should enter the following.

Example

```
tanja[martin]: cubeinfo
(host) cubeinfo: Commser not responding
tanja[martin]: rebootcube
1.  Execute takedown run file: /usr3/ipsc/lib/rc0
2.  Execute startup run file: /usr3/ipsc/lib/rc1
tanja[martin]: cubeinfo
(host) cubeinfo: There is no attached cube
tanja[martin]:
```

3.2 System Calls

The iPSC system calls allow the iPSC system commands to be performed from within host and node programs and provide additional functions. The iPSC system calls are divided functionally into four groups: cube control, message passing, byte swapping, and global operations. Some system calls can be issued by either host or node programs. Others are available only to host or only to node programs. When these calls are listed below, the environment in which they can be invoked is identified.

3.2.1 Cube Control

The iPSC provides four types of system calls for cube control:

- Allocating, Loading, and Releasing a Cube
- Redirecting Input and Output
- Controlling Processes
- Handling Errors and Exceptions

This section provides a brief description of each category. A more detailed description is available in the *iPSC/2 and iPSC/860 User's Guide*.

The first type of cube control system calls deal with allocating, loading, and releasing a cube. When allocating a cube, you must assign a unique name or accept the name **defaultname**. The last cube allocated is referred to as the current cube. When you issue a **cubeinfo**, **load**, or **relcube** system call without specifying a cube name, the call refers to the current cubes are allocated, the current cube can be changed with **attachcube**.

System Call	Environment	Description
attachcube	host	Attach to a cube and make it the current cube.
cubeinfo	host	Obtain information about allocated cubes.
getcube	host	Allocate a cube.
load	host/node	Load a node process.
myhost	host/node	Obtain node ID of host machine.
mynode	host/node	Obtain node ID of calling process.
mypid	host/node	Obtain NX process ID (always 0) of call process.
relcube	host	Release specified cube.
setpid	host	Set NX process ID for host program.

The system calls for redirecting input and output provide a method of altering the standard output and standard error of the host and nodes. These calls are only available from the host program.

The system calls for controlling processes provide a means of synchronization and termination of node processes. For example, one set of calls enables the user to terminate a specific node process or a set of processes; while another set allows the user to wait for a node or set of nodes.

The system calls for handling errors and exceptions provide a method to invoke a user supplied routine for a hardware interrupt. The default action when a node process experiences a hardware exception is to print an error message and kill the process.

3.2.2 Message Passing

The iPSC/860 is a multicomputer consisting of independent processor/memory pairs which do not share physical memory. Each processor has it own memory and process cooperation occurs through message passing.

Messages can be either synchronous or asynchronous and are characterized by a length, a type, and an id. The message length is described in bytes, and the message sending routines will send exactly the specified length. If the receive buffer is not large enough to hold the message, an error will occur. Different mechanisms are employed for sending long messages (> 128 bytes). The type is an identifier, determined by the programmer, allowing control and validation of messages by type. The id is an identifier used to check for the completion of asynchronous messages.

The iPSC provides system calls for:

- Synchronous and asynchronous message passing
- Pending messages
- Getting information about pending or received messages
- Flushing and canceling messages
- Treating a message as an interrupt

This section provides a description of each category. A more detailed description is available in the *iPSC/2 and iPSC/860 User's Guide*.

A synchronous send indicates that the submitting program waits until the send is complete. The completion of the send is not a verification that the message was received, but only means that the message left the sending process. A synchronous receive means that the receiving program waits until the message arrives in the specified buffer. The synchronous message calls are:

System Call	Environment	Description
crecv	host/node	Receive a message, wait for completion.
csend	host/node	Send a message, wait for completion.
csendrecv	host/node	Simultaneously, send a message, and post a receive. Wait for completion of the receive.

An asynchronous operation, either a send or receive, does not cause the submitting program to wait until the operation is complete, but returns a unique message id which can be tested for completion. The iPSC has a limited number of message ids, and care must be taken to release unneeded ids.

The asynchronous message calls are:

System Call	Environment	Description
irecv	host/node	Receive a message, don't wait for completion.
isend	host/node	Send a message, don't wait for completion.
isendrecv	host/node	Simultaneously, send a message, and post a receive. Don't wait for completion.

A pending message is a message that is available for receipt, but has not yet been received. That is, a message type has arrived, for which a receive has not been issued, and is held in a system buffer until a receive is issued. If a receive has already been issued, the message goes directly into the application's buffer and bypasses the system buffer.

The iPSC provides system calls to return information about received or pending messages. These calls, often referred to as info calls, return the size of the message, its type, and the node number and NX pid of the sending process. Note that the NX pid of the sending process on an **rx** node always returns as 0.

The iPSC provides system calls to flush a pending message by clearing pending messages from the system buffer. The **flushmsg()** call only flushes messages pending to be received, not those pending to be sent. The **msgcancel()** call cancels an asynchronous send or receive operation.

System calls are provided to treat a message as an interrupt and attach a handler to the message's receipt. That handler is then invoked when the message of that type is either sent or received. For sections of critical code which should not be interrupted, a system call is provided to mask all handlers.

3.2.3 Byte Swapping

The use of byte-swapping calls are needed when sending messages between the host and the cube on which the remote workstation does not use Intel's byte ordering convention. The Intel convention specifies that the least significant byte of an integer is stored at the lowest memory address. A Sun workstation such as **edwin** uses a different byte ordering convention. Host programs on **edwin** which pass messages to the Intel will require byte-swapping. The SRM (**pixy**) does not require byte-swapping as it matches the iPSC/860's byte ordering convention. Byte-swapping is required on either the remote host or the node - but not both.

The names of the byte-swapping routines denote the base type of message being sent and the direction between machines. For example, **HTOCL()** means that a Host byte ordering is converted TO a Cube byte ordering and the data consists of **Long** integers.

The byte-swapping calls require two arguments. The first is the address of the data to be swapped; the second is the size of the message. The size of the message is in terms of the number of base type elements. That is, for an integer array of 100 elements, the size of the message is 100.

The following partial code segment demonstrates how to swap bytes between the host and node with an array of 5 elements of integers requiring 4 bytes. Note that the byte-swapping routine for long integers is used for integers represented with 4 bytes.
Example:

```
integer*4 msg_send(5)
integer*4 msg_rec(5)
...
...
HTOCL(msg_send, 5) # Host TO Cube Long
csend(NODE_TYPE, msg_send, 20, -1, NODE_PID) # 20 denotes message length
... # in terms of bytes
CTOHL(msg_send, 5) # Swap the message back
... # in case it is needed
...
crecv(NODE_TYPE, msg_rec, 20) # Receive a message
CTOHL(msg_send, 5) # Swap the received message
...
```

NODE TYPE - message type. NODE PID - process ID of destination node.

There are some special calls for byte-swapping for C language constructs.

3.2.4 Global Operations

A global operation is a iPSC system call providing a high level construct for communication among node processes. Global operations optimize communication by using the "e-cube routing algorithm" which operate on nearest neighbors over a DCM channel.

Examples of global operations provided include:

System Call	Environment	Description
gdhigh	node	Global vector double precision MAX operation
gdlow	node	Global vector double precision MIN operation
gdprod	node	Global vector double precision MULTIPLY operation
gdsum	node	Global vector double precision SUM operation
giand	node	Global vector integer bitwise AND operation
gior	node	Global vector integer bitwise OR operation
gixor	node	Global vector integer bitwise exclusive OR operation
gsendx	node	Send a vector to a list of nodes
gsync	node	Global synchronization operation

There are Online Manual pages available on both **pixy** and **edwin** for all of the above commands.

4 Using Fortran and C

Applications development for the Intel system is supported by various software components. Both FORTRAN and C language compilers for the local and remote hosts may be used for developing a host program designed to interface with any allocated node programs. Cross-compilers enable iPSC/860 code to be generated. A library of system-callable routines gives access to operating system and machine as well as cube specific functions (see Section 3).

There are several FORTRAN and C compilers for the various platforms of the iPSC/860 system. You can compile host programs either on **pixy** or **edwin**.

FORTRAN		**C**	
Compiler	Target Machine	Compiler	Target Machine
f77	pixy	cc	pixy
f77	edwin	cc	edwin
if77	iPSC/860	icc	iPSC/860

The compilers **f77** and **cc** are the standard FORTRAN and C compilers from Sun Microsystems. The **if77** and **icc** compilers are cross-compilers for the iPSC/860 from The Portland Group. The **f77** and **cc** compilers on pixy are compilers written by Greenhills for the Intel 386 chip and are the native **f77** and **cc** compilers for the SRM. To use any of these compilers, you must first set the environment variable **IPSC_XDEV**. This is to insure that the compiler(s) can locate the necessary Intel libraries and include files. The syntax to set the environment variable is:

setenv IPSC_XDEV /usr/ipsc/XDEV

The above directory only applies to **pixy**, for **edwin** the directory is called **/usr/tools/ssd**. The *iPSC/2 and iPSC/860 User's Guide* refers to the rf77 and rcc compilers. These are

for older Intel systems with CX nodes and should not be used on **UniVie**'s iPSC/860 which has RX nodes.

4.1 FORTRAN on the Intel System

An Intel application program may consist of code running on several platforms; that is, code running only on the Intel nodes (i.e., node only program); or code running on the host and code running on the nodes (i.e., Host/Node program). Thus, compilation may consist of generating code for different architectures. Additionally, the compilation of a program can also be performed on various platforms. That is, when compiling the node program, you can work on the remote host or use the SRM. This section describes how to compile host and node FORTRAN programs for the Intel on the remote host and the SRM.

4.1.1 FORTRAN Compilation on the Remote Host

All home directories are located on **edwin** and NFS-mounted on **pixy**. The iPSC/860 cross-compiler, **if77**, is installed on **edwin**, making it possible to compile and run node programs with little need to login directly to pixy. The use of the remote host is encouraged to reduce the load on the SRM, which is severely underpowered as a front-end for the iPSC/860. CPU intensive applications or host programs should not be run on the SRM.

For programs which run on the RX (i860) nodes, the **if77** command invokes the iPSC/860 FORTRAN compiler, assembler, and linker. Set the environment variable **IPSC_XDEV** as shown in Section 4.

More detailed information on **if77** may be obtained by invoking the **man** command, or from the *iPSC/860 FORTRAN Compiler User's Guide* and the *iPSC/2 FORTRAN Language Reference Manual*.

The basic syntax for the **if77** compiler is:

if77 [switches] sourcefile...

A partial list of switches follows:

-node Creates an executable program for RX nodes.

-Kflag Requests special compilation semantics from the compiler. The permitted flag values are:

ieee - Performs REAL and DOUBLE PRECISION divides in conformance with the IEEE 754 standard (default).

noieee - Performs REAL and DOUBLE PRECISION divides using a faster inline divide algorithm, which produces results that differ from

the IEEE result by no more than three units in the last place.

-g Generates symbolic debug information at optimization level 0, unless a **-O** switch is present on the command line after the **-g**.

-O[**level**] Set the optimization level:

0 - A basic block is generated for each FORTRAN statement.

1 - Scheduling within extended basic blocks and some register allocation is performed.

2 - Level 1 optimizations plus scalar optimizations such as induction and loop invariant motion by the global optimizer.

3 - Level 2 optimizations plus software pipelining.

4 - Level 3 optimizations with aggressive register allocation for software pipelined loops.

If a level is not supplied with **-O**, the optimization level is set to 2. If **-O** is not specified, then the level is set to 0 if **-g** is specified, and set to 1 if **-g** is not specified.

-c Skips the link step; compiles and assembles only.

-oname Uses name for the executable program. The default is **a.out**.

-Idir Adds directory to the compiler's search path for include files. For include files surrounded by angle brackets (<..>), each directory is searched followed by the default location. For include files surrounded by double quotes (".."), the directory containing the file containing the include statement is searched, followed by the **-I** directories, followed by the default location.

-llib Loads **lib.a** from the standard library directory. The library name is constructed and the full library path is passed to the linker. See also the **-L** switch.

-Ldir Changes the default directory in which the linker searches for libraries to directory. The linker searches directory first (i.e., before the default path and before any previously specified **-L paths**).

The **if77** command bases its processing on the suffixes of the files it is passed. Files

generally have names ending with **.F**, **.f**, **.s**, **.o** and **.a**. The meaning of each of these extensions is:

.F FORTRAN source code to be preprocessed, compiled and assembled.
.f FORTRAN source code to be compiled and assembled.
.s i860 assembly language files to be assembled.
.o object files to be passed directly to the linker.
.a libraries, which must be linked.

All other files are taken as object files and passed to the linker (if linking is requested) with a warning message. If a single FORTRAN program is compiled and linked with the **if77** command, the intermediate object and assembly files are deleted.

Example:

The following command compiles the node program in the file **f1.f** on the remote host, using the **-o** option to indicate the executable as **f1**, and the **-node** option to link in the appropriate libraries.

if77 -o f1 f1.f -node -i860

For host programs which run on **edwin**, the **/usr/local/bin/f77** command invokes the Sun FORTRAN compiler and can be used with the appropriate libraries to interface with node programs running on the RX (i860) nodes. This command requires access to the appropriate include and library files.

More detailed information may be obtained by using the **man** command and the *Sun FORTRAN Compiler User's Guide*.

The basic syntax for the **f77** compiler is:

f77 [**switches**] **sourcefile...**

A partial list of Intel related switches follows:

-Idir Adds directory to the compiler's search path for include files. The switch **-I/usr/ipsc/include** adds the directory for Intel include files.

-Ldir Changes the default directory in which the linker searches for libraries to **dir**.

-llib Loads liblib.a from the standard library directory. The switch **-lhost** loads the Intel **libhost.a** library.

Example:

The following command compiles a host program that runs on **edwin**, using the **-I** option defining the prefix for include files, the **-o** option to indicate the executable as **prog**, and the **-lhost** option to link in the appropriate libraries.

f77 -I/usr/ipsc/include -o prog prog.f -lhost

4.1.2 FORTRAN Compilation on the SRM

Programs which run on the RX (i860) nodes are compiled the same as on the remote host. Set the environment variable **IPSC_XDEV**, as shown in Section 4.

The basic syntax for the **if77** compiler is:

if77 [switches] sourcefiles -node

See Section 4.1.1 for details on the switches.

Host programs which run on the SRM (discouraged, but sometimes useful) are compiled slightly differently than on the remote host. The include and library files for the Sun are not needed during compilation on the SRM, thus the FORTRAN compilation command is simplified.

The basic syntax for the Greenhills **f77** compiler script is:

f77 [switches] sourcefiles -host

A partial list of options for **f77** follows:

-host	Creates an executable program for the SRM.
-o name	Uses **name** for the executable program. The default is **a.out**.
-c	Skips the link step; compiles and assembles only.

Example:

f77 -o prog prog.f -host

4.2 C on the Intel system

An Intel application program may consist of code running on several platforms; that is, code running only on the Intel nodes (i.e., node only program); or code running on the

host and code running on the nodes (i.e., Host/Node program). Thus, compilation may consist of generating code for different architectures. Additionally, the compilation of a program can also be performed on various platforms. That is, when compiling the node program, you can work on a remote host or use the SRM. This section describes how to compile host and node C programs for the Intel on the remote host and the SRM.

4.2.1 C Compilation on a Remote Host

All home directories are located on **edwin** and NFS-mounted on **pixy**. The iPSC/860 cross-compiler, **icc**, has been installed on **edwin**, making it possible to compile and run node programs there with little need to login directly to **pixy**. The use of the remote host is encouraged to reduce the load on the SRM, which is severely underpowered as a front-end for the iPSC/860. CPU intensive applications or host programs should not be run on the SRM.

For programs which run on the RX (i860) nodes, the **icc** command invokes the iPSC/860 C compiler, assembler, and linker with switches derived from the driver's command line switches. Set the environment variable **IPSC_XDEV**, as shown in Section 4.

More detailed information may be obtained by using **man icc**, the *iPSC/2 and iPSC/860 User's Guide* or the *iPSC/2 and iPSC/860 C Language Reference Manual*.

The basic syntax for the **icc** compiler is:

icc [switches] sourcefile...

A partial list of switches follows:

-node	Creates an executable program for RX nodes.
-Kflag	Requests special compilation semantics from the compiler. The permitted flag values are **ieee** and **noieee**. See Section 4.1.1 for description of these flags.
-g	Generates symbolic debug information at optimization level 0, unless the **-O** switch **-g** switch.
-O[level]	Set the optimization level to **0**, **1**, **2** or **3**. See Section 4.1.1 for a description of these levels.
-c	Skips the link step; compiles and assembles only.
-o name	Uses **name** for the executable program. The default is **a.out**.
-Idir	Adds **dir** to the compiler's search path for include files. See Section 4.1.1 for details.

-llib Loads liblib.a from the standard library directory. See Section 4.1.1 for details.

-Ldir Changes the default directory in which the linker searches for libraries to directory. See Section 4.1.1 for details.

The **icc** command bases its processing on the suffixes of the files it is passed. Files specified generally have names ending with **.c**, **.s**, **.o** and **.a**.

The meaning of each of these extensions is:

- **.c** C source code to be compiled and assembled.
- **.s** i860 assembly language files to be assembled.
- **.o** object files to be passed directly to the linker.
- **.a** libraries, which must be linked.

All other files are taken as object files and passed to the linker (if linking is requested) with a warning message. If a C program is compiled and linked with the **icc** command, the intermediate object and assembly files are deleted.

Example:

The following command compiles the node program in the file **node.c** on the remote host, using the **-o** option to indicate the executable as **node**, and the **-node** option to link in the appropriate libraries.

icc -o node node.c -node -i860

For host programs which run on **edwin**, the **/usr/ucb/cc** command invokes the Sun C compiler and can be used with the appropriate libraries to interface with node programs running on the RX (i860) nodes. This command requires access to the appropriate include and library files.

More detailed information may be obtained by using the **man** command and the *Sun C Compiler User's Guide*.

The basic syntax for the **cc** compiler is:

cc [switches] sourcefile...

A partial list of Intel related switches follows:

-Idir Adds **dir** to the compiler's search path for include files. The switch **-I/usr/ipsc/include** adds the directory for Intel include files.

-Ldir	Changes the default directory in which the linker searches for libraries to **dir**.
-llib	Loads **liblib.a** from the standard library directory. The switch **-lhost** loads the Intel **libhost.a** library.

Example:

The following command compiles a host program to run on **edwin** using the **-I** option defining the prefix for include files, the **-o** option to indicate the executable as **host**, the **-L** option to define the prefix for library files, and the **-lhost** option to link in the appropriate libraries.

cc -I/usr/ipsc/include -o host host.c -lhost ...

4.2.2 C Compilation on the SRM

Programs which run on the RX (i860) nodes are compiled the same as on the remote host. Set the environment variable **IPSC_XDEV**, as shown in Section 4.

The basic syntax for the **icc** compiler is:

icc [switches] sourcefile... -node ...

See Section 4.2.1 for more details on the switches.

Host programs which run on the SRM (discouraged, but sometimes useful) are compiled slightly differently than on the remote host. The include and library files for the Sun are not needed during compilation on the SRM, thus the C compilation command is simplified.

The basic syntax for the Greenhills **cc** compiler is:

cc [switches] sourcefiles -host

Example:

cc -o prog prog.c -host

5 Mathematical Libraries

There are two mathematical libraries available on the iPSC/860, the Basic Math Library and Veclib, the vector library. This section provides an overview of their content, char-

acteristics and use. More detailed information is available in the documents *iPSC/860 Basic Math Library User's Guide* and *iPSC/2 and IPSC/860 Math Libraries Reference Manual* . These libraries are not available to host programs. Reference appropriate documentation for the SRM or your remote host to determine availability and use of provided mathematical software.

5.1 iPSC Basic Math Library

The iPSC Basic Math Library contains highly optimized implementations of many of the operations most often found to dominate large scale scientific computing. Library content includes routines comprising the the Basic Linear Algebra Subprograms (BLAS) collection supplemented with onedimensional Fast Fourier Transform (FFT) routines. The iPSC Basic Math Library is available only to FORTRAN node programs. Access to this library is accomplished by specifying the **-lkmath** load option on the **if77** command line as follows:

if77 -node -o node node.f -lkmath

5.1.1 Content and Characteristics

All three levels of BLAS routines are included. Both single and double precision real and complex arguments are supported.

- BLAS Level 1 routines input one or two vectors and output either a vector or a scalar. Specific examples are dot product, vector scaling and Givens plane rotation.
- BLAS Level 2 routines perform calculations involving both vectors and matrices. Specific examples are the product of a vector and a matrix, the solution of a linear system and certain low rank matrix update operations.
- BLAS Level 3 routines have matrices as both their input and output arguments. Specific examples are the product of two matrices and certain high rank matrix update operations.

Three one-dimensional FFT routines are included in both single and double precision versions. Transforms are done in place and are limited to power-of-two lengths. Forced precomputation of necessary coefficients provides highly efficient computation of multiple transforms of common length. Specific routines are:

- Forward or inverse FFT, complex to complex
- Forward FFT, real to complex
- Inverse FFT, complex (forward FFT of a real vector) to real

5.1.2 Arguments

Vector arguments are passed in one-dimensional arrays and include length and stride attributes. The stride (or increment) can be positive, negative or zero, respectively specifying vector element selection from the input array in a forward, reverse or broadcast fashion, respectively.

Matrix arguments are passed in two-dimensional arrays and include leading dimension, number of rows and number of columns as attributes. In addition, a character transposition parameter is often passed which indicates whether the matrix argument is to be used in normal, transposed or (for a complex matrix) conjugate transpose form.

5.1.3 Performance Hints

For optimal performance, follow these guidelines for array arguments whenever possible:

- Use a vector stride of one.
- Begin double precision complex arrays on a 16 byte memory address boundary.
- Begin array arguments to FFT routines on a 16 byte memory address boundary.
- Place vector arguments to BLAS 1 routines in the data cache. Place the first of two vector arguments in data cache when both will not fit.

5.2 iPSC Vector Library

The iPSC Vector Library VecLib routines provide a basic set of vector operations that can be used to replace either FORTRAN DO loops or C for loops. VecLib is available to both FORTRAN and C node programs. Access to this library is accomplished by specifying the **-lvec** load option on the **if77** or **icc** command line as follows:

if77 -node -o node node.f -lvec

icc -node -o node node.f -lvec

5.2.1 Content and Characteristics

VecLib routines include, but are not limited to, BLAS Level 1 routines, along with routines for direct and inverse FFT of single or double precision complex data. All VecLib routine names include a one character prefix to a base functional name which designates the vector argument type they accept. FFT routines do not transform in place. BLAS Level 2 and Level 3 routines are not available. This library is recommended only as a supplement to the iPSC Basic Math Library.

6 System Software Tools

Several third party software packages are available to analyze and/or evaluate application programs being developed for the iPSC/860. Additionally, Intel provides a Performance Analysis Tool (PAT) to gather statistics on the expenditure of execution-time resources.

6.1 Portable Instrumented Communication Library (PICL)

PICL is a set of communication routines developed primarily at Oakridge National Laboratories with the purpose of providing both a portable communication layer as well as communication routines with an inbuilt mechanism for generating trace file output.

The library is made up of three distinct sets of routines.

- low-level communication and system primitives
- high-level communication routines
- and routines to control the tracing facility

PICL consists of two libraries, **hostlib.a** for host programs and **nodelib.a** for node programs. On **pixy** they can be found in

hostlib: /usr/lib/libhostlib.a and

nodelib: /usr/ipsc/XDEV/i860/lib-coff/libnodelib.a

On **edwin** the filenames are

hostlib: /usr/local/lib/libhostlib.a and

nodelib: /usr/tools/ssd/i860/lib-coff/libnodelib.a

More information about PICL and the available functions can be found in the directory **/global/doc/picl**.

6.2 Paragraph

Paragraph is a graphical package which displays the results of a program run using PICL communication routines. It may be invoked on a SUN workstation, using the command PG (**/usr/ipsc/bin/PG**).

Paragraph is currently being further developed by Intel. It will be marketed under the name **SSD Paragraph** in an enhanced version which includes self-defining data formats, thus enabling its use as a graphical display for program performance data from a range of trace file formats.

The documentation for Paragraph can be found in **/global/doc/paragraph**.

6.3 Performance Analysis Tool (PAT)

The iPSC/860 Parallel Performance Analysis Tools (PAT) is an Intel Supercomputer Systems Division version of a performance analysis product developed by ParaSoft Corporation. The PAT utilities provide tools for analyzing program execution, communication performance, and event traced performance of application programs on the iPSC/860 system.

The PAT utilities gather performance data at runtime and then output the data to disk when the application terminates. PAT provides a set of analysis tools that convert the performance data to graphical and tabular forms that can be analyzed interactively using X. The PAT utilities provide hardcopy output in PostScript form. The reference for PAT is the *iPSC/860 Parallel Performance Analysis Tools Manual*.

The PAT performance data is collected automatically by using compiler switches and/or an environment variables or selectively through the use of PAT C or FORTRAN calls that are compiled with the application program. The iPSC system uses three basic methods to collect PAT profiling data:

- Compiler switches to automatically profile/trace application performance.
- Environment variables that interactively turn on/off the gathering of performance data.
- A programmatic interface that allows you to manually insert PAT system calls at desired locations in the application code.

The preferred method of gathering profiling data uses switches with the C and FORTRAN compilers to turn one or more of the profilers on or off for the code being compiled. With this method, appropriate PAT procedures are automatically inserted at the start and exit of every procedure/subroutine.

The **-Mperf** switch is available with the iPSC/860 C and FORTRAN compiler drivers, and turns on PAT instrumentation code. The syntax for the use of **-Mperf** is:

-Mperf [= { prof | comm | event } [= { auto | manual }] [,...]

A short description of the parameters follows:

prof	Enables the PAT execution profiling.
comm	Enables the PAT communication tracing.
event	Enables the PAT event tracing.
auto	Specifies that performance monitoring code is to be invoked automatically on the application.
manual	Specifies that performance monitoring code is to be invoked manually using PAT programmatic procedure calls.

The environment variable, **EXPROF_SWITCHES** allows you to set or reset any or all the PAT profile/trace tools. The syntax for the **EXPROF_SWITCHES** environment variable follows:

setenv EXPROF_SWITCHES "[c][e][x]"

where the option **c** represents communication tracing, **e** represents event tracing, and **x** represents execution profiling.

The PAT utilities provide three tools to analyze and display the results from the PAT profiling on the iPSC system:

- The execution profiler tool, **xtool**, monitors time spent in individual routines.
- The communication profiler tool, **ctool**, assesses time spent in communication and I/O.
- The event profiler tool, **etool**, shows the interactions between processors and allows user-specified events to be monitored.

The syntax for the use of each tool is:

xtool [options] program-name

ctool [options] program-name

etool [options] program-name

A partial list of options follows:

-p Suppress graphical output. Tabular output to stdout.

-T Use an alternative graphical device for output (e.g. -TX).

See the **man** page for **pat** or for each tool for a more complete description.

7 Running VFCS Code on the Intel

Output from VFCS generated using the iPSC backend includes the Intel byte-swapping routines (see Section 3.2.3) required to execute the host from a SUN workstation. However, it may also be desirable to run it from the SRM.

A simple program to compile the output from VFCS for the iSPC/860 is available in **/usr/ipsc/vfcs_util/bin** on both systems. It is called **ipsccomp**. A command line switch is provided to generate optional trace data. Read the README file in this directory for further information.

Both systems make use of runtime libraries specially constructed to run with Vienna Fortran programs. These libraries are maintained by the system operator, and are stored

in the standard library directories on each machine, so there is no need to provide any path information when linking them. Host codes use routines in the library **vfcs-host.a** and node programs use routines from **vfcs.a** and **parti.a**.

8 Debugging

The Interactive Parallel Debugger is a full-featured symbolic debugger that is useful in debugging FORTRAN, C and assembly language programs. The ability to delineate and control the debugging environment when running parallel applications makes this tool useful to the applications developer.

8.1 Interactive Parallel Debugger (IPD)

The Interactive Parallel Debugger (IPD) is used to debug codes on the SRM. IPD provides source level debugging functionality similar to debuggers available on other computers including commands specific to debugging a parallel application. IPD may only be used to debug node programs on **pixy**. You may not debug a host program on either **pixy** or a remote host such as **edwin**. Before invoking IPD, you must compile your code with the -g flag, as shown:

if77 -node -i860 -g -o node node.f

Before ipd is invoked, you must specify how many nodes you require.

The prompt **ipd>** appears. Enter IPD commands at this prompt. The first command required is **load**. You must load the node code onto the iPSC/860. For example:

load (all:0) node

The prompt changes to the default context **(all:0)**. Typing **help** or **?** lists all the IPD commands. For more information about a particular command, type **help** followed by the command name. To run the program enter **run** followed by the **wait** command as follows:

(all:0) run; wait

The program runs until all processes are complete, a breakpoint is reached or the program is manually interrupted with a Control-c. If you manually interrupt the program, use the **stop** command to halt execution.

The **frame** command gives a traceback. The commands **msgqueue** and **recvqueue** allow you to look at the message send and receive queues. You can use the **context** command to define the set of node(s) that the debug command affects. To set and remove breakpoints, use the **break** and **remove** commands respectively. To terminate IPD, type **exit**.

If you used **getcube** to obtain a cube, you need to do a **relcube** to release the cube after exiting IPD.

For more information about IPD, see the *iPSC/2 and iPSC/860 Interactive Parallel Debugger Manual*.

9 Intel Documentation

All manuals listed below are available in the Computer Room. Only the copies of the manuals are allowed to be removed from this room. The original manuals may only be read there.

Online documentation like README files or Postscript manuals for the different software packages mentioned can be found on **edwin** in the directory **/global/doc**. These are frequently updated by the Systems Administrator.

9.1 The iPSC/2 and iPSC/860 User's Guide

This manual gives an introduction to the iPSC systems. Major features of the hardware and software are described and some example programs are given. This manual is highly recommended for the new user.

Intel manuals include:

- Intel Mini Manual (Preliminary)
- iPSC/2 and iPSC/860 User's Guide
- iPSC/860 Basic Math Library User's Guide
- iPSC/2 FORTRAN Language Reference Manual
- iPSC/860 FORTRAN Compiler User's Guide
- iPSC2 iPSC/860 Programmer's Reference Manual
- iPSC/860 C Compiler User's Guide
- iPSC/2 iPSC/860 C Language Reference Manual
- iPSC/2 iPSC/860 Interactive Parallel Debugger Manual
- iPSC/860 Parallel Performance Analysis Tools Manual
- iPSC/2 iPSC/860 Network Queueing System Manual
- i860 64-bit Microprocessor Assembler and Linker Reference Manual
- iPSC/2 iPSC/860 Technical Documentation Guide
- iPSC/2 and iPSC/860 Math Libraries Reference Manual