1991

Porting Ncube Packages to iPSC/860 and iPSC/2 Hypercubes

Jingwen Wang

Mo Mu

Report Number:
91-049

PORTING NCUBE PACKAGES TO iPS/860 AND iPSC/2 HYPERCUBES

Jingwen Wang
Mo Mu

CSD-TR-91-049
June 1991

Porting Ncube Packages to iPSC/860 and iPSC/2 Hypercubes

Jingwen Wang and Mo Mu
Computer Sciences Department
Purdue University
Technical Report CSD-TR-91-049
CAPO Report CER-91-21
June 1991

Abstract

This report collects the experiences in porting the Sparse Matrix Solver of the Pellpack from Ncube/2 to Intel hypercubes. The differences between these machines are discussed and some important machine dependent issues are raised and solved. A Ncube/2 library has been installed on the Intel iPSC/2 and iPSC/860. Steps to run Ncube/2 code on the Intel hypercubes are listed for both the specific Sparse Matrix Solver package and general Ncube/2 programs.

1. Machine Features

We are not going to give a detailed description of these machines as they could be found in many documents listed at the end of this report. We just want to compare them and show the similarities and differences between them.

Host System

Both iPSC/2 and iPSC/860 have the PC/386 as their host machine. This host is too slow compared with the Sun/4 host of the Ncube/2. Users have been complaining about the speed and software environment of this host. First, it is slow in compiling code, loading the nodes and controlling input and output of hypercube. Second, the modest Unix System V support on this machine makes it difficult to develop sophisticated software environment. There is even no on-line manual on this host. Also, no news and X-window is now supported on this host. One solution to this is to use the remote host support of Intel hypercubes. Sun work-stations can be used as the remote host of the system so that software development can be done partly on the remote host and the user does not need to log into the PC/386 machine to use the hypercube. However, this facility is currently not supported by our iPSC/860 system. The other way is to mount the PC/386 host's file system into the Network File System (NFS). In this way, files on the 386 host can be accessed transparently by any Sun in the cluster as if they were on the same machine. The Caltech iPSC/860 system has been using this approach.

Node Processors

The Ncube uses its own custom VLSI chips as node processors. The iPSC/2 uses 80386/80387 pair as a node processor, called CX node. A fast scalar processor Weitek 1167 can be included optionally on each node to form a so-called SX node. Another

option on iPSC/2 is to include a vector board on each node to form a VX node. The iPSC/860 uses the i860 processor as a node processor. This processor is much more powerful than other processors. Its peak performance is 18Mflops. But current compilers can only dig out 6-9 Mflops. Handcoded assembly routine can have higher performance. Even the current 6-9Mflops speed is too fast for its communication hardware.

The Compiler

The current Ncube/2 uses ncc to compile node code. It is more convenient than the original xucc. However, the xucc is still working in this machine, and most of the makefiles on the Pellpack are using the xucc. The Ncube/2 ncc compile command has an option to specify the maximum communication buffers allocated from the node memory, this should be remembered should there be a problem of insufficient communication buffer. This option allows a simpler and efficient communication protocol. The Intel hypercube, however, does not have this option. Users don't need to worry about the space of the buffer, the system does it automatically, at the cost of more complicated communication protocols and lower efficiency. The host program of the Ncube/2 is compiled by the standard *f77* compiler, with the inclusion of the host libraries for controlling and management of the cube. The iPSC/2 uses *f77* for compiling both the host and node programs because both node and host processors are 80386/387. The options "-host" and "*-node*" tells the compiler whether to link with the host or node libraries. The iPSC/860 uses *f77* for host code and *pgf77* for node code. An additional option "*-i860*" is used with the *pgf77* to tell the complier the node code is for i860 processor and the i860 libraries should be linked. The preprocessors of the Fortran compiler for the Ncube/2 and the Intel machines use different conventions, the former uses "$" to start a preprocessor statement (e.g. *include*), while the latter write the preprocessor statement in the same format as the ordinary fortran statements.

Host OS

The Sun OS of the Ncube/2 support the Unix with BSD4.3 features. Almost all popular tools in the Unix environment can be installed on it. The AT&T System V Unix OS on the PC/386 of the Intel machines, however, provides only bear Unix shell. Also, some commands are incompatible between these machines. For example, the "*mv dir1 dir2*" command on the Sun OS can move directory *dir1* under directory *dir2*. This command cannot run on the PC/386 host. The "*getrusage()*" function on the Sun OS which gives time information of the host in microseconds is not supported on the PC/386 host.

Communications

Ncube/2 and Intel hypercubes (iPSC/2 and iPSC/860) are very similar in communication mechanisms. They have similar communication hardware and similar communication libraries. However, the Intel hypercubes uses different protocols in communications as the Ncube/2.

2. Approaches to Porting Ncube/2 Code to Intel Machines

The key to port Ncube/2 programs to Intel machines is to map the Ncube/2 communication calls to Intel machine's communication calls. Several possible ways are:

1) Rewrite the communication statements in the Ncube/2 code in terms of Intel machine's calls. This method is advantageous in minimize the overhead associated with such transformations. The disadvantage is that every program has to be modified on a one by one basis.

2) Write a library that automatically interprets Ncube/2 communication calls in terms of Intel machine's communication statements. This method provides a generic way of porting programs on different machines. Once the library is built, any program on the Ncube/2 can be run directly on the Intel machines without any modifications. However, you have to switch to the new software environment on the host. The potential drawback of this approach is the lower efficiency than the first method. However, since the two hypercube families have very similar syntaxes in communication statements, there is only a negligible overhead in doing so. For example, the "*nread()*" call on the Ncube/2 can run on the iPSC/860 machine with this library with an additional overhead of only a dozen of microseconds, which is only a very small fraction of the communication startup time.

We used the second approach since it is a more practical way to port large packages of routines. We have designed such a library for the Intel machines. This library will run on both the iPSC/2 and iPSC/860. It should be included as a library when compiling the code of Ncube/2 on the Intel machines.

3. Machine Dependent Issues

In addition to the design of a Ncube/2 library on the Intel machine, some machine specific modifications have to be done in order for the program to run correctly. These machine dependent modifications are:

1) The byte-swapping operations needed for the Ncube/2 are no longer needed on the PC/386 host of the Intel machines. This is because the Intel machines use the same address convention for both host and node memories. Thus the byte swapping routines of the Ncube/2 code are modified to skip the swap operation if the host is PC/386 (or SRM).

2) Some of the utility programs on the Ncube/2 Pellpack were written in C (e.g. i9swap.c). These routines occasionally call the Ncube/2 communication routines in the C format, which are not implemented in the current Ncube/2 libraries of the iPSC hypercubes. It is necessary to change these C routines so that they will call these communication routines in Fortran format (by adding a suffix underscore to these primitive names and changing all parameters passed to the nread and nwrite calls into pointer data type). These utility

routines are the system support of the Pellpack shared by all application programs generated by the preprocessor. So once they are modified, all applications can run with them without modifications. Currently all utility routines called by the Sparse Matrix Solver have been changed to use Fortran format nread and nwrite as they were ported to iPSC hypercubes.

3) The Fortran on the PC/386 host of the Intel machines does not assume the implied file fort.n for device number n. Thus, if your program has a statement like "write(2,1234) ....", and you have not opened a file as device number 2, then the program will dump core during execution. The Pellpack programs have depended upon the machine to open the file fort.2 for device number 2 in a I/O statement. On Intel machines, you must explicitly open every file you use. These opens are currently implemented in the Ncube/2 library of the iPSC hypercubes, thus no modifications to Pellpack programs are necessary.

4) The PC/386 host of the Intel hypercube does not have the Unix kernel call "getrusage" to measure the execution times of the host program. The Ncube/2 Pellpack has been using this call on the Sun/4 to measure the time spent in executing the host program in microseconds, including the system software overhead on this process. We used the Intel library call *mclock()* for this purpose on the Intel hypercube. *mclock()* measures the same time as the *getrusage()*, but it does so in milliseconds, not microseconds. Since the host system software overhead varies from one execution to another on largely the milliseconds range, this accuracy can really make sense.

5) In order to measure the performance of the parallel programs, it is necessary to make all processors synchronized before the computation begins. To make this measurement accurately, some details about the clock timers on these machines must be clarified. The Ncube/2 uses a single crystal oscillator to drive all the processors and their associated clocks and timers. Thus all processors have exactly the same time counts at any instant. The Ncube/2 provides no direct instruction to synchronize all processors. The Sparse Matrix Solver of the Pellpack uses a broadcast routine to synchronize all nodes. It is shown by measurement that this routine can synchronize all nodes quite accurately. The iPSC/860 machine, however, uses a different approach. Each processor has its own crystal oscillator and clock. These clocks are reset at booting time simultaneously. If the time is long enough, there will be some offsets between the clocks of the the individual nodes. The Intel machines provide a call to synchronize all the nodes in the allocated subcube. Therefore, we can use a software timer on each node and reinitialize the timers on all nodes simultaneously after this synchronization call. The *q1time()* routine of the Pellpack has been modified to effect this function. When a minus parameter is passed to this subroutine, the software timer of the node will be reset to zero. This is done by subtracting from any subsequent readings of the timer an initial value, which is the time the reset was done. The iPSC/2 machine has bigger clock offsets among the processors. In addition to the

separate oscillators on the processors, the clocks of the processors are maintained by software. So the timers are actually initialized after the node operating system is loaded. On a 64-processor iPSC/2 hypercube, several seconds can pass before the last processor's operating system is loaded and the time reset. Thus, the iPSC/2 clocks will have offsets of several seconds as soon as they are started. Fortunately, the synchronization call and the software timer we used can solve the time difference problem by reseting the timers after the synchronization call. The IPSC/860 and the iPSC/2 hypercubes are software compatible. The code written on one machine can run on the other directly except that the statements concerning the allocation of cubes may need to change. There are some other differences between these two machines, apart from the above ones. The timers on the iPSC/860 nodes provide an timing resolution of 1μs, while those on the iPSC/2 can only provide 1ms resolution. You will get nonsense data if you are trying to measure times of very small granularity on the iPSC/2.

6) There is a Pellpack preprocessor error that must be noted if the code is to run on the Intel hypercubes. There are 2 main programs on each Pellpack application. One is for host, the other is for nodes. By careful examination, you will find that there are many arrays in the programs which have the same name in both the host and the node program. These identical names in the host and nodes are often the communication counterparts between the host and the nodes. They are supposed to have the same dimension sizes in the host and nodes. However, current Pellpack preprocessor produces the host and node programs that are inconsistent in data array sizes. These arrays always have a dimension size of 1 while their counterparts in the host have sizes of several hundreds or thousands. This error has survived on the Ncube/2 processor, but generated error messages on the Intel machines. It is therefore required to check the source code produced by the preprocessor so that all data arrays on the nodes have same sizes as the host. Currently we are correcting this by hand easily for the Sparse Matrix Solver. This problem should not be confused with the inconsistent data size among the common blocks of the main program and subroutines, which is allowed on most systems and produces only warning messages.

7) There are also differences in generating libraries on the PC/386 host. No "*ranlib*" command is provided on this AT&T Unix system V. You only need to use "*ar*" command to generate host and node libraries for the iPSC/2 machine and to generate iPSC/860 host libraries. The iPSC/860 node libraries are generated with "*ar860*" command. You never need to use "*ranlib*" command as in the Ncube/2 host.

8) One of the major differences between the Ncube/2 and Intel hypercubes in software is the different parameters used in communication calls. The Ncube/2 *nread()* call requires the user to provide information about the source of the message as a parameter. The Intel hypercubes, however, do not

have this information in their *crecv()* or *irecv()* calls. Since the programs designed for the Ncube/2 are containing this information in communications, we must somehow provide this on the Intel hypercubes in order that the program can run on them without any modification. On Ncube/2, message types are only 16 bits, while on Intel hypercubes, message types are 32 bits. Thus we can use the higher 16 bits of the message type of the Intel machine to record message source because these bits will not be used by any programs designed for Ncube/2. In the Ncube/2 library of the Intel machine, we have used this method to merge the message source into message type automatically in the *nwrite()* call, and extract this information in the *nread()* call. This is done transparently to the user, so user programs can still use the original form of the Ncube/2 *nread()* and *nwrite()* calls.

4. Current Software Development

We have built a Ncube/2 library for the Intel hypercube so that every Ncube/2 program can be moved to the iPSC/2 and iPSC/860 without concerning the difference in communication statements. The user can run the program as if it was still on the Ncube/2, but with a different host and different compiler. The only thing a user has to learn about is the new host environment and new compiling procedure. Thus the user does not need to change his program, but he need to rewrite his makefile. We shall give a list of the steps to follow when porting an application from Ncube/2 to Intel machines at a later section.

The current status of the work is only the first step in porting the Pellpack from Ncube/2 to Intel hypercubes. We have successfully moved the Sparse Matrix Solver to both the iPSC/2 and iPSC/860 machine. The iPSC/2 and iPSC/860 are compatible in software. Our ported Sparse Matrix Solver on the iPSC/860 is moved to iPSC/2 as is without any problem. Although we have only moved one package of the Pellpack to Intel machines, we believe that most of the problems in porting a generic Ncube/2 program to the Intel machines have been solved and tested. The issues summarized above will be common to other Pellpack programs. To port other packages to the Intel machines, the same Ncube/2 library can be linked and few modifications to the original code are needed. The only expected modifications should be mostly the host machine dependent problems that are not met before, such as those code related with the Pellpack tool, visual display etc.

In addition to porting the original Sparse Matrix Solver, we have expended the Solver with additional features to improve performance and to aid in performance visualization. In summary, we have currently 4 parts of programs developed or ported on the Intel hypercubes. There are makefiles in these directories to create the corresponding Pellpack libraries.

1) Ncube/2 Library

The Ncube/2 library consists of two parts: *nhost.f* and *ncube.f*. They are the Ncube host and node libraries on the Intel machine. The compiled object code was created into two library modules to be linked with Ncube/2 host and node programs respectively.

These library routines are currently installed in */usr/cube/u/wangj/ncube/lib* and the source code is in */usr/cube/wangj/ncube*. There is a makefile in the source directory to create the libraries. In the */usr/cube/u/wangj/ncube* directory, we have designed several Ncube/2 example programs to test the Ncube/2 library. These examples are in subdirectory example and example1. These routines were used to measure the software overhead associated with the library as well as to study the various global communication calls provided by the iPSC/860. By running these programs, we can have estimates the performance of broadcast communications, global collection communications. Although this library supports all Ncube/2 system calls used by the Pellpack, it is not a complete implementation of all the Ncube/2 calls. In case that an unimplemented call is used by your program, the compiler will tell you that the routine is not defined. It will be helpful if you have a look at the source code of the Ncube/2 library before using it.

2) Sparse Solver

The original Ncube/2 Sparse Solver is currently installed under the directory */usr/cube/u/wangj/pellpack*. This directory contains both source code and library code. It contains 4 parts: 1) *lib*: this directory contains the routines of the Sparse Solver. 2) *ellplib*: This directory contains the common Pellpack routines that are part of the Ellpack. They are shared by all Pellpack libraries, be it Sparse Solver or other Solvers. 3) *pellplib*: This directory contains the common Pellpack routines shared by all Pellpack packages. The pellplib and ellplib form part of the utility routines of the Pellpack they provide services such as error report, terminal message printing, timing and communication utilities etc. 4) *mislib*: This directory contains the miscellaneous routines pertaining to the Sparse Solver.

3) Optimized Sparse Solver

The optimizations include block wrapping and message merging technique. The only code changed are in the solution module of the node and host code. The optimized code is under the directory of */usr/cube/u/wangj/pellopt* in parallel with the pellpack directory. The optimized package can still run in its original method as a special case (block size=1). Thus the original package can in be completely replaced by the new code.

4) Optimized Sparse Solver with performance visualization aids

This tool is added onto the optimized Sparse Solver and it only affects the solution code. This code is under the directory of both the host and node code of the solution module. It appears as a subdirectory "see" in both the node and host directory of the solution module (*/usr/cube/u/wangj/pellopt/lib/src/5ig*).

5) Node and Host main programs

The above directories contain the Pellpack library routines of the Sparse Solver. With these library routines, we can solve different sparse problems by producing the source main programs for the node and host with the Pellpack preprocessor. The Pellpack tool can automatically generate the source code according to user's specifications of the problem and the choice of solution methods. The resultant source

code can be placed in any directory the user likes to. Currently, there are many sample problem directories in the directory */usr/cube/u/wangj/tmpxx*, where "*xx*" varies from 21 to 101, each representing a problem size (e.g. 21×21). There is two makefiles in each of these directories, one for sequential Sparse Solver (*makes*), the other for Parallel Sparse Solver (*makefile*). The parallel source programs are named *hmain.f* and *nmain.f* for Parallel Sparse Solver, *hs.f* and *ns.f* for Sequential Sparse Solver. The makefiles in different directories are the same.

5. Running a Ncube/2 code on the Intel hypercubes

1) Environment setting

To use the i860 cube, the following directories need to be included in the search path:
/bin /usr/bin /usr/local/bin /usr/lib/etc /usr/local/pgi/i860/bin
/usr/i860/ipsc/bin /lib

You also need to define an environment variable *PGI as /usr/local/pgi*. You can simply copy the *.login* file in */usr/cube/u/wangj* to your home and make modifications as you may like to.

2) Steps to run the Sparse Solver on the Intel machines

The Sparse Solver Libraries have been ported to the iPSC/860 and iPSC/2 machines. In order to run a program generated by the Pellpack Preprocessor for the Ncube machine, following steps are required:

- Create the desired Sparse Solver libraries by running the makefiles in the appropriate directories. For example, if you need to run the original point wrapping Sparse Solver, run all makefiles in the */usr/cube/u/wangj/Pellpack* directory. If you need to run the improved block wrapping method, run the makefiles in the */usr/cube/u/wangj/pellopt/lib/src/5ig* directory after running the makefiles in the Pellpack directory. The only difference between the Pellpack and Pellopt is in the solution routines (directory *5ig*). You can even discard the Pellpack directory since the improved block wrapping program can also run in point wrapping. Before running the block wrapping makefiles, you need to do an egrep mbsize * to see what the current block size is and make changes if the size is not what you wanted. If you need to run the improved block wrapping method with performance visualization aids, run the makefiles in sub-directory *5ig/host/see* and *5ig/node/see*.
- Create a working directory on the cube machine. Transfer the source code generated by the Pellpack Preprocessor from ifestos to the working directory in the Intel machine. You can use ftp to transfer your files quickly. See a later section for an easy way to transfer your files with *ftp* script quickly.
- Copy the *makefile* from directory */cube/usr/u/wangj/tmpxx* to your working directory. Edit the *makefile* and change the filenames in accordance with your host and node source file names respectively.

- Enter *make,* and compile the code.
- Enter *host* (suppose *"host"* is the executable file name of your host program), the program will begin execution.

3) Steps to run an arbitrary Ncube/2 program on the Intel hypercubes

With the Ncube/2 library in directory */usr/cube/u/wangj/ncube/lib* linked to to your code, in principle you should be able to run any code written for the Ncube/2 on the Intel hypercubes. However, you have to solve any problems that are host dependent. Fortunately, there are only a few such problems and we believe that most of which have been addressed in section 3. As we have noted at the beginning of section 4, we can run the Ncube/2 code on the Intel machine with the library as if it were still running on the Ncube/2, except that a different host environment and a different compiler is used.

The steps to run a general Ncube/2 code on the Intel hypercubes are as follows:

- Move your Ncube/2 source code to the Intel host.
- Create a makefile that contains the following lines:

```
NCUBE=/usr/cube/u/wangj/ncube/lib/ncube.a
NHOST=/usr/cube/u/wangj/ncube/lib/nhost.a

all:    host node
node:nmain.f
        pgf77 -o node nmain.f $(NCUBE) -node -i860

host:hmain.f
        f77 -o host hmain.f $(NHOST) -host
```

- Enter *make* to compile the host and node programs.
- Enter *host* to run the program.

If you have any user libraries in your program, like the Pellpack case, you need first to move this library source routines and compile them in the new host environment. See the current Sparse Solver libraries on the iPSC/860 for an example of how to compile the code for library routines and how to generate libraries. Note the differences in the commands used to generate libraries, which we have pointed out at the end of section 3.

Once the libraries have been generated, you only need to included them in the compile command line of the makefile in the same way *"NHOST"* and *"NCUBE"* libraries are used in the above makefile example.

6. Using the Sun -- i860 Cross-Compiler

The compiler provided by intel running on the 386 host of the i860 machine is rather slow. The Sun - i860 compiler enables the user to compile node programs several times faster on the Sun-4 stations. However, the cross compiler does not compile the

host program because the host program depends on what host machine (SRM or Sun) you use. Currently, host programs have to be compiled on the PC/386 host. The cross-compiler is currently installed in the ifestos.

To use the cross-compiler on the Sun, add two lines in your *.login* file of Sun:
setenv PGI /usr/local/pgi/../..
set path = ($path /usr/local/pgi)

This should enable you to use the cross-compiler. To Cross-compile your file, use the command:
pgf77 -o node node.f -i860 -node

If you need to use the preprocessor, the preprocessor statements (such as include) should be of the format: *include 'filename'*. No "#" or "$" sign is needed. The statements mustn't be put in the label area. They should appear just as other fortran statements do. Also note that no -B option is accepted by this cross-compiler. You may also need to copy the *fcube.h* include file from the 386 host to your Sun.

7. File Transfer Using *ftp*

To use the i860 from the 386 host, you often need to *ftp* the files between your home host and the PC/386 host of the Intel hypercube. To simplify this tiresome process, following is a convenient way to do this automatically (assuming you need to frequently transfer files to and from the PC/386 host when you are on the Sun).

- Login to the 386 host machine.
- create a file named "ft" on one of your directory that is included in the search path. The *ft* file contains only one line:
  ftp yourhost.cs.purdue.edu
- Change the mode of file *ft* by using *chmod +x*.
- Assuming that:
  The name of your Sun machine is: *ifestos;*
  your login name on your Sun machine is *johnson;*
  your password on the Sun is *mypasswd;*

  Create a *.netrc* file in your home directory. It contains the following:

  machine ifestos.cs.purdue.edu
  login johnson
  password mypasswd
  macdef init
  first ftp command you wish to be executed automatically (e.g. cd mydir)
  *2nd ftp command you wish to be executed automatically (e.g. mget *.f)*
  .
  .
  .
  last ftp commands (e.g. quit)
  (This is a blank line -- you must leave a blank line)

- *chmod 600 .netrc*

  This command protect your password from being read by others.

- Now, you just need to press *ft*, the machine will *ftp* into your Sun very quickly and finishes the desired operations. Be sure to cd to the right directory before you start the transfer.

If you are not familiar with *ftp*, just press "?" after pressing *ft*. You will get a help menu. The same .netrc file can be created on the Sun host if you are going to transfer files when you are currently on the PC/386 host.

8. Tape archive files

The current files in the iPSC/2 and iPSC/860 have been placed into Tape archive files (Tar files). By doing so we can put an entire library into one file and can be saved for later use. These Tar files are saved in directory */u20/wangjw/Tar*. There is a *README* file in it describing how to extract these files in Intel hypercubes.

There are altogether 7 Tar files in this directory. The *i860* sub-directory contains the 4 Tar files for the iPSC/860 hypercube. These 4 Tar files are for Ncube library (*Ncube.860tar*), Sparse Matrix Solver library (*pellpack.860tar*), block wrapping Sparse Matrix Solver library (*pelopt.860tar*) and a sample application directory (*tmp21.860tar*). The block wrapping library in fact contains a sub-directory for the performance visualization aids in the solution sub-directory (*5ig*). See section 4 for the contents of these directories. The sample application directory is included because the makefiles in this directory is important. The order in which the libraries are linked with the host and node main programs is significant. The ipsc2 sub-directory contains 3 Tar files. They are similar to the i860 case except that the Sparse Matrix Solver library (point wrapping) is not included because it is the special case of block wrapping. There is also a tar file for a sample application directory on the iPSC/2. Details can be found in the *README* file of the */u20/wangjw/Tar* directory.

These Tar files can be extracted into their original directory structure on any machine with the command "*tar -xf tar_filename*". To move the package onto an Intel hypercube, we only need to move these tar files to the home directory and extract them and compile them. So these tar files are very convenient in keeping these packages in a very neat form.

It should be noted that these tar files are created on a specific user's home directory (Jingwen Wang), so they will work only if they are extracted in his home directory. If you are going to move it to your own directory, then after extract the tar files, you need to change the path names in all the makefiles to your home directory environment before you can compile them correctly.

Acknowledgements

This work was supported in part by the National Science Foundation under grant CCR-86 19817, the Air Force Office of Scientific Research grant 88-0234, and the Strategic Defense Initiative through ARO grant DAAGL-03-90-0107.

References

1. iPSC/2 User's Guide, Intel Corporation, March 1988.
2. iPSC/2 Fortran Programmer's Reference Manual, Intel Corporation, March 1988. IP 3. iPSC/860 User's Guide, Intel Corporation, Intel Corporation, June 1990.
4. iPSC/860 Programmer's Reference Manual, Intel Corporation, June 1990.
5. Ncube2 Programmer's Reference Manual, Ncube Corporation, Dec. 1990.
6. Ncube2 Programmer's Guide, Ncube Corporation, Dec. 1990.
7. Mo Mu, J.R.Rice and J. Wang, Performance Experiments and Optimizations of PDE Sparse Solvers on Hypercubes, Technical Report CSD-TR-91-10045, June 1991.
8. Mo Mu and J.R.Rice, The Structure of Parallel Sparse Matrix Algorithms for Solving Elliptic Partial Differential Equations on Hypercubes, Tech. Report CSD-TR-976, Sept. 1990.
9. T.H.Dunigan, Hypercube Clock Synchronization, ORNL/TM-11744, March 1991.
10. T.H.Dunigan, Performance of the Intel iPSC/860 Hypercube, ORNL/TM-11491, June 1990.