Chapter 2

Developing a CMMD Program

This chapter discusses the mechanics of developing a message-passing program:

- basic components of a CMMD program
- hostless programs
- host/node programs
- hostless vs. host/node programming
- include files
- I/O modes
- sample programs

2.1 Basic Components of a CMMD Program

The source code for a CM-5 message-passing program is of two types:

- For a hostless program, there is a single set of source code files for the nodes. The host (partition manager) initiates execution of the node program, and thereafter acts as an I/O server for the nodes.
- For a host/node program, there are two sets of source code files, one for the host and one for the nodes. The host program must explicitly start and monitor the execution of the node programs.

Copyright © 1995 Thinking Machines Corporation

2.2 Hostless Programs

Hostless programming is the recommended model for writing a CMMD message-passing program. It supports C, C++, C*, Fortran 77, and CM Fortran. In hostless mode, the user writes a single node program, which runs on all the nodes. This program does all the computation and communication; it does not communicate explicitly with the host.

Each node executes its code asynchronously, fetching data and instructions from its local memory. It synchronizes with other nodes only when required to do so for message-passing purposes (for example, to send or receive a synchronous message or to participate in a global instruction, or to do parallel I/O).

In hostless programming, what runs on the host is an internal server program provided by the CMMD library itself. This program

- enables CMMD
- downloads the user code to the nodes, which then begin executing it
- goes into a polling loop as an I/O server, so that it can communicate with I/O devices on behalf of the nodes. This allows node programs to do I/O.

2.2.1 Computation and Communication

In hostless programming, the program that executes on each node performs all the usual tasks of an application program (initialization of variables, allocation of memory, computation, post-run cleanup, etc.). The computation section of the code is written as it is for a serial machine — that is, there are no special commands as there are in data parallel languages.

Communication among nodes uses CMMD function calls. Communication can be point-to-point, when one node sends a message to a second node; or it can be global, with all nodes contributing to the message (and, usually, with all nodes receiving the result). Global communications synchronize all the nodes; point-to-point communications can be either synchronous or asynchronous.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

2.2.2 Termination

To terminate a hostless program normally, all nodes should call `exit`. Execution of this call on all nodes releases the host from the I/O server and allows the host internal server program to exit normally.

Note that even if your program does not explicitly call the `exit` routine, `exit` is called automatically when program execution completes (that is, when the `main` routine returns).

The exit status as seen on the partition manager is the sum of all the exit statuses of the nodes.

2.3 Host/Node Programs

Host/node programs have a user-written program that runs on the host, that is, on the partition manager, and a second program that runs on the nodes. The host program may be written only in C, C++, or Fortran 77. The node program and the host program cooperate with each other.

The host program starts parallel operations by calling `CMMD_enable`. This function

- gathers `argv`, `argc`, environment variables, and process data in a process control block
- downloads node code, and thus allows the nodes to start running
- downloads the process control block
- starts up the CMMD I/O server, which goes into a polling loop, waiting for I/O requests from the nodes. The server stops only when it is explicitly disabled by the nodes, or when the program terminates abnormally. The host program can execute user code only when the I/O server has been disabled.

NOTE: If any other CM node activity has taken place, the user must ensure that the activity has finished before calling `CMMD_enable`.

The node program must start its operations by having all nodes call `CMMD_enable_host`. This function disables the host I/O server, and allows

Copyright © 1995 Thinking Machines Corporation

the host to return from the `CMMD_enable` call and execute the rest of the user-written host program.

In order to perform I/O, the node program must temporarily re-enable the I/O server. CMMD provides server functions for this purpose, as explained in the chapter on host/node programming in the *CMMD Reference Manual*.

Code that runs on the host (that is, on the partition manager) may contain anything ordinarily included in a program running on a Sun computer. This includes system calls, I/O calls, X11 routines, and calls to other specialized libraries.

2.3.1 Computation and Communication

Host programs, when used, generally perform computations, make CMMD calls to communicate with the nodes (perhaps to provide input or receive output from them), and make calls to other libraries or routines.

The CMMD library provides specific functions for communication between host and nodes; these are discussed in the chapter on host/node programming in the *CMMD Reference Manual.*

2.3.2 Termination

Host/node programs terminate only when the host program calls `exit`. Having the node program call `exit` merely puts the nodes into a busy-wait state in the OS dispatch loop.

The exit status of a host/node program is simply the exit status of the host program. The node exit statuses are not communicated to the host.

As we have stated, host/node CMMD programming is designed for use by serial languages on the host: C, C++, and Fortran 77. A form of hostless programming known as global/local programming provides many of the advantages of host/node programming, as well as other advantages, for programs written in the parallel languages CM Fortran and C*. Global/local programming for CM Fortran is described in Part I of the *CMMD Reference Manual* for Version 3.3, and in the *CM Fortran Libraries Reference Manual* for Version 2.2. Global/local programming in C* is discussed in the *C* Programming Guide* for Version 7.2.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

2.4 Hostless versus Host/Node Programming

The recommended method of programming in CMMD is hostless mode, for it is much simpler than host/node mode.

Hostless programming requires only one set of source files, those for the node program. A "canned" host executable, which acts as an I/O server, is linked in automatically. Hence, compiling and linking is simpler, and makefiles tend to be simpler and shorter. Also, in hostless mode, I/O from the nodes takes place naturally and transparently.

Host/node programs, in contrast, require writing a separate program to run on the host. The "canned" host executable provided by CMMD is not used. Compiling and linking is more complicated, and if the nodes must perform I/O, coordinating their actions with the host can be very tedious. Furthermore, since there are two completely separate components of the program, debugging is usually more difficult.

Most users therefore prefer hostless programming over host/node programming.

Notice, however, that some types of programs may not fit easily into the hostless model. For example, a "master/slave" application might have the need for a distinguished processor with special attributes that only the partition manager has. Such attributes might be, for example, a local disk, an Ethernet connection, or other special device not available to the nodes. Disregarding the issue of special devices, even though one of the nodes (node 0, say) might theoretically serve as the "master," it might nevertheless be very inconvenient to have it do so. Problems may arise from the fact that all nodes must participate in reductions, or for algorithmic reasons the number of "slaves" might need to be a power of two. Nevertheless, it is rare for these considerations to outweigh the sheer simplicity offered by the hostless model.

2.5 Include Files

The include files required by CMMD are the same for hostless and host/node programs. For C-based programs (programs written in C, C++, or C*), the include file to use is

```
<cm/cmmd.h>
```


Copyright © 1995 Thinking Machines Corporation

For Fortran-based programs (programs written in Fortran 77 and CM Fortran), the include file is

```
'cm/cmmd_fort.h'
```

Standard include files may also be needed. For example,

```
<stdio.h>
<fcntl.h>
<sys/types.h>
```

Specific include files may be required for particular libraries. For instance, a C* program that uses global/local inquiry functions must include the header file `<csgl.h>`.

2.6 I/O Modes

CMMD provides four modes of I/O:

- *Local independent* — Each node opens and accesses files directly. In Fortran, each node does direct access I/O to its own file, accessing records according to the record argument.
- *Global independent* — A single file is opened on all nodes simultaneously, with each node having its own, independently movable, file position indicator. In Fortran each node can access a record in that file via the record argument.
- *Global synchronous broadcast* — A global file is accessed by all nodes simultaneously, with each node reading (or writing) the same data. In Fortran, all processors access the same record.
- *Global synchronous sequential* — A global file is accessed by all nodes simultaneously, with each node reading (or writing) its own portion of the file. In Fortran, processors access sequentially numbered records, starting with the record accessed by processor 0.

In local independent mode, each node can potentially access a different file, and a separate file descriptor is needed on the host for each file opened by each node. In contrast, the global modes allow all nodes to open and access a file simultaneously, using only a single UNIX file descriptor (or Fortran unit number, which is mapped to the file descriptor). Because UNIX sets a

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

limit on the number of files that can be open at any one time, the global file modes provide a significant increase in flexibility, allowing the nodes to perform multiple file accesses simultaneously without the immediate threat of running out of file descriptors.

A file or stream opened in any one of the three global modes can be changed to one of the other global modes at any time; in fact, a single node can independently change its own view of the mode of a file to facilitate its own purposes, without affecting the mode of the file as seen by other nodes.

The usual UNIX streams — `stdin`, `stdout`, and `stderr` — are available to the nodes. Since a CMMD program is a single process, all nodes share the same streams. Therefore, they must take care not to overwrite each other's contributions to the streams. This is usually accomplished by putting the streams into append mode; see the *CMMD Reference Manual* for details.

2.6.1 Special Issues with Fortran I/O

CMMD I/O is based on UNIX I/O. UNIX expects file I/O to be unbuffered, and expects user applications to have a relatively high degree of control over the timing of opens, reads, writes, etc. Fortran I/O, on the other hand, is based on buffered I/O. This causes certain problems for synchronous I/O, as explained below.

CMMD I/O Mode Restrictions

Buffered I/O, by its nature, removes the user's control over when the underlying read or write occurs. For instance, if `stdio` is used to write to a file, the actual UNIX write call will occur when the total number of bytes written to the file reaches the buffer size (typically 8K), rather than at each call to a `stdio` library routine. This means that it is difficult or impossible for the programmer to guarantee synchronous calls across the partition to the underlying I/O routines. Particularly troublesome are situations in which contributions to a write from some nodes fit into the I/O buffer, requiring only one write, while contributions from other nodes overfill the buffer, thus requiring more than one write.

Synchronous I/O Modes. This restriction results in a difficulty for Fortran I/O under synchronous sequential and synchronous broadcast modes, as the

Copyright © 1995 Thinking Machines Corporation

only Fortran I/O mechanisms provided are built on top of a layer of buffering. (Synchronous broadcast mode may work correctly, as long as the number of bytes read or written, and any line buffering done, is identical on each node.)

Two workarounds are possible for I/O in synchronous modes:

- Fortran 77 programs may be able to modify their I/O buffer sizes so that an entire read or write for each node fits into the buffer. (Use the Sun `fileopt='BUFFER=`*`n`*`'` extension to the open statement, where *`n`* is the desired buffer size.)
- To allow Fortran programmers to access the underlying, unbuffered UNIX I/O calls, CMMD provides global routines that resolve to these read and write calls.

The global I/O CMMD routines are global read and write calls:

```
INTEGER FUNCTION
    CMMD_global_read(unit, buffer, length)
INTEGER FUNCTION
    CMMD_global_write(unit, buffer, length)
         character*(*)  buffer
         integer        unit, length
```

The *`unit`* argument is a standard Fortran file unit number; *`length`* is the number of bytes to be read or written. On success, these functions return the number of bytes written or read; they return −1 on error. A program can use standard Fortran I/O functions to write data into character buffers, and then call the global CMMD I/O functions with the character data buffers as arguments.

Programmers should note the following issues regarding these functions:

- These functions take unit numbers as arguments. Any flags (for example, `status=new`) that are checked at the time the units are opened are still checked. Any other flags that affect writing to the unit (such as record format, etc.) are ignored.
- The global I/O functions can be called only on file units in synchronous-sequential or synchronous-broadcast mode. They will return errors if called on file units that are in local or global independent mode.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

- You can access any given file unit either by Fortran read and write statements or by these CMMD functions, but not by both. (Using both *may* work, but this usage is not supported.)
- The files created by `CMMD_global_write` will be normal UNIX files, not standard Fortran files; in particular, they won't have record separators. (This may or may not be a problem for a given application.)

Global Independent I/O Mode. There can also be problems using global independent mode from Fortran 77 and CM Fortran.

In Fortran 77 you can handle any buffering problems by explicitly using the `fileopt='BUFFER=`*n*' option. (While this may not always be necessary, it is nevertheless good practice to set this flag explicity.)

In CM Fortran the proper method for avoiding potential problems is to state explicitly the number of bytes being read or written in the read and write commands. (In other words, don't use the "*" specification.)

Local I/O Mode. I/O in local mode is not affected by these buffering problems.

Don't Mix F77 I/O and CM Fortran I/O

If your main program is in Fortran 77, you should use Fortran 77 for all I/O; if your main program is in CM Fortran, you should use CM Fortran I/O throughout. Do not try to mix Fortran 77 I/O with CM Fortran I/O.

2.7 Sample Programs

Several sample CMMD programs are included with this release. They can be found in the directory

`/usr/examples/cmmd/{hostless | hostnode}/`*`language/example`*

If you do not find them there, ask your system administrator where they were installed at your site.

Copyright © 1995 Thinking Machines Corporation