Chapter 3

Compiling and Linking a CMMD Program

This chapter describes compiling and linking for CMMD programs:

- compiling hostless C, C++, C* programs
- compiling hostless Fortran 77 and CM Fortran programs
- compiling host/node programs
- linking with `cmmd-ld`

3.1 Compiling Hostless C, C++, and C* Programs

In many respects, these are the most "natural" languages for message-passing programming, since they are designed to work within a UNIX environment and they share UNIX notions of I/O.

3.1.1 Compilers

C and C++ Programs

You compile a hostless program just as you would an ordinary C or C++ program. You use whatever compiler (Sun or GNU) you would ordinarily use, and you supply your own makefile. (The only difference is that linking

Copyright © 1995 Thinking Machines Corporation

must be done with `cmmd-ld`; see Section 3.4, below.) The following compiler versions are supported:

- All Sun bundled C compilers are supported. Versions of `acc` (unbundled Sun C) prior to Version 2.0 are also supported.
- All versions of GNU C up to and including Version 2.3.3 are supported.
- The Sun CFront (C++) compiler Version 1.0 is supported. The GNU G++ (C++) compiler Version 2.3.3 and earlier is supported.

For information on C and C++ compilers, see the SunOS documentation.

C* Programs

As a convenience, it is possible to compile *and link* a hostless CMMD program written in C*, using just the C* compiler (`cs`). To do this, you specify the `-node` switch, which indicates hostless C*-on-a-node execution. (When compiling on a CM with vector units, you must also supply the `-vu` switch to ensure proper linking of the program.)

NOTE: In a hostless C* program, the node program's main routine must be written and compiled in C* so that C*-specific initializations are included.

For full functionality in programs that use parallel compilers, use C* Version 7.2 (Beta or final) or later.

For information on compiling and linking a global/local C* program (a closely related program model), see the *C* Programming Guide*, Version 7.2.

For information on the parallel compiler `cs`, see the *CM-5 C* User's Guide*, Version 7.2.

3.1.2 Execution Model for Vector Units

C and C++ cannot access the VUs directly. They must call routines written in DPEAC in order to do so. (For information on DPEAC, see the *VU Programmer's Handbook*.)

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

C* is designed to use the VUs. C* applications can be written in either of two ways:

- *C*-on-a-node* works entirely locally, with one copy of the application running independently on each node. All arrays are local as well; each node spreads its parallel arrays across its four VUs.
- *Global/local C** is based on a global C* program, which allocates its parallel arrays across all VUs in the partition. Routines within the global program may call local C* routines whenever they wish. During the execution of these local routines, all control is local. Each node acts independently, and each acts on its own subgrid of a global parallel array. During the local routines, nodes can also allocate their own local parallel arrays; however, these are accessible only within the scope of the local routines.

3.2 Compiling Hostless Fortran 77 and CM Fortran Programs

3.2.1 Compilers

Fortran 77 Programs

You compile a hostless F77 program just as you would an ordinary Fortran 77 program. You use the compiler you would ordinarily use, and you supply your own makefile. (The only difference is that you must link it using `cmmd-ld`; see Section 3.4 , below.)

- Sun F77 versions 1.* are supported.
- Version 2.0 will compile and run, but is not supported by the Prism debugger.

For information on the `f77` compiler, see the SunOS documentation.

Using the −Nx Option to Increase Symbol Table Limit. Fortran 77 programmers may find that the default Fortran 77 symbol table size is not large

Copyright © 1995 Thinking Machines Corporation

enough to contain the symbols used in a CMMD program. You can tell this is the case if you see the following error when you try to compile a program:

```
f77 program.pn.F ...
Compiler error: Too many external symbols.
```

A simple workaround is to specify the `-Nx` option to increase the size of the symbol table:

```
f77 program.pn.F ... -Nx500
```

Using the −f Option in Fortran 77. It is possible for a Fortran 77 program to contain double-precision data that is not doubleword aligned in memory (the default setting in Fortran 77 is not to force double alignment). The Fortran 77 versions of certain CMMD routines will get a bus error and crash the program if they are called on such data. The reason for this is that CMMD routines always attempt double loads when dereferencing double pointers. To avoid this problem, compile with the `-f` or `-dalign` option.

CM Fortran Programs

As a convenience, it is possible to compile *and link* a hostless CMMD program written in CM Fortran (that is, to use CM Fortran "-on-a-node"), using just the CM Fortran compiler (`cmf`). To do this, specify the `-node` switch, which indicates hostless CM Fortran-on-a-node execution. (When compiling on a CM with vector units, you must also supply the `-vu` switch to ensure proper linking of the program.)

For full functionality in programs that use CM Fortran, use CM Fortran Version 2.2.

For information on compiling and linking a global/local CM Fortran program, see the *CM Fortran Libraries Reference Manual*, Version 2.2.

For information on the `cmf` compiler, see the *CM Fortran User's Guide*, Version 2.2.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

3.2.2 Execution Model for Vector Units

Fortran 77 applications cannot access the VUs directly. They must call routines written in DPEAC or in CM Fortran in order to do so. (For information on DPEAC, see the *VU Programmer's Handbook*.)

CM Fortran is designed to access the VUs. CM Fortran programs can be written in one of two styles:

- *CMF-on-a-node* works entirely locally, with one copy of the application running independently on each node. All arrays are local as well; each node spreads its parallel arrays across its four VUs.
- *Global/local CM Fortran* is based on a global CM Fortran program, which allocates its parallel arrays across all VUs in the partition. Routines within the global program may call local routines whenever they wish. (The local routines are usually written in Fortran 77.) During the execution of these local routines, all control is local. Each node acts independently, and each acts on its own subgrid of a global parallel array. During the local routines, nodes can also allocate their own local parallel arrays; however, these are accessible only within the scope of the local routines.

For further details, see the section on global/local programming in the *CM Fortran Libraries Reference Manual*, Version 2.2.

3.3 Compiling Host/Node Programs

Remember: In host/node programming, the host part cannot be written in C* or CM Fortran. It has to be written in C, C++, or F77.

You compile and link host/node programs in almost the same way as hostless programs. For host/node programs, however,

- You must write your own host program.
- When you compile the host program, you must specify `-DCP_CODE`. This allows the use of appropriate include files for the control processor, rather than those for the nodes.

Copyright © 1995 Thinking Machines Corporation

- When linking, you must specify the `-host` option as well as the `-node` option to either `cmf` or `cmmd-ld`. (The `-host` option specifies the host objects to be linked.)

3.3.1 Fortran 77–CM Fortran Host/Node Programs

For convenience, it is possible to compile a host/node program using the CM Fortran compiler (`cmf`) as a single tool that

- compiles and links the nodes, using any combination of files written in CM Fortran and F77.
- directs the host to be compiled and linked with an appropriate non-parallel compiler.

Another way of looking at this model is as an ordinary F77 host/node program in which the nodes happen to call CM Fortran routines.

To do this, use the parallel compiler `cmf` to

- compile and link only the node part of the program in CM Fortran, by preceding the node files with the `-node` switch and the host files with the `-host` switch.
- supply the `-comphost` switch to specify which non-parallel compiler is to be used to compile and link the host program. Remember, the argument to `-comphost` cannot be `cmf`; it must be one of `f77`, `gcc`, `acc`, `cc`, `g++`, etc.

3.3.2 C-C* Host/Node Programs

The same convenience does not exist for compiling C and C* host/node programs, that is, you cannot use `cs` in the manner just described for `cmf`. Instead, you must compile your C and/or C* node programs using `cs`, and your node program using whichever non-parallel compiler you prefer, and link them using `cmmd-ld` (specifying the `-host` and `-comphost` options).

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

3.3.3 Global/Local CM Fortran and Global/Local C*

Global/local programming (in CM Fortran or C*) should not be confused with host/node programming. One way to understand global/local is as "hostless," with a changing "viewpoint" (global or local). In other words, when the local part is running (the CMMD part), it *cannot* communicate with the host, except to do I/O in a hostless CMMD fashion. In host/node CMMD programs, however, the nodes may perform explicit message passing, reductions, or other operations with the host as a peer. For more details, see the *CM Fortran Libraries Reference Manual* and the *C* Programming Guide*.

3.4 Linking with cmmd−ld

After compiling your program, you must link it using the customized linker, `cmmd-ld`. Some options are

`-comp` *comp*
: Required. Specifies the compiler to be used for linking (`gcc`, `f77`, etc.).

`-comphost` *comp*
: Optional. Selects a different compiler to be used for host files in host/node linking (see Section 3.3.1, above).

`-vu`
: Required for applications written in CM Fortran or C*, or any applications that want to make use of the CM-5's VUs. Indicates that code should be linked with a VU-compatible library. Should be used only on CM-5s equipped with vector units.

`-node` *object_code.o library.a ...*
: Required. Specifies the object code and library files that you want linked for the node programs.

`-host` *object_code.o library.a ...*
: Optional. Specifies alternative object code and library files that you want linked for the host program.

Copyright © 1995 Thinking Machines Corporation

`-o` *`executable_name`*
: Optional. Specifies executable name (default `a.out`).

`-v`
: Optional. Requests verbose descriptions of the linking process.

`-l`, `-L`, etc.
: Optional. Pathnames of any required user libraries. (Same as corresponding argument of Sun `ld`.)

`-cmos_root` *`path`*
: Optional. Tells the linker where to find the CMOST system software, if not in `/usr/lib`.

`-cmmd_root` *`path`*
: Optional. Tells the linker where to find the CMMD library, if not in `/usr/lib`.

`cmmd-ld` first uses the specified compiler to link the program in the normal manner; it then customizes the resulting file for execution with CMMD, adding its own host program.

An on-line manual page is available for `cmmd-ld`.

Usage Note: If you attempt to compile a program with many source files or very long pathnames, you may run into `cmmd-ld`'s command-line length limit, which is currently 4K characters. A simple workaround for this limit is to collapse a number of the object files you are linking by using `ld`:

`ld -r -o` *`output`*`.o` *`list-of-input`*`.o's`

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation