Chapter 3

Execution Models

CM Fortran's *execution model* refers to the way a program makes use of the hardware. CM systems support the following basic execution models, plus some variations:

- *global* model, where a single program operates on arrays of data spread across all the parallel processors
- *nodal* model, where multiple copies of a parallel program operate independently on subsections of the data and communicate in message-passing style
- *global/local* model, a combination of the global and nodal models, for maximum flexibility in programming
- *simulator*, where a data parallel program executes on a single processor (for purposes of testing and debugging)

The compiler always generates two streams of code: serial and parallel. A user-supplied switch determines the actual system components that will serve as "the front end" and "the CM." These switches control the assembly-level language generated for parallel code blocks and determine the libraries used for linking.

Copyright © 1991–1994 Thinking Machines Corporation

NOTE

The execution models are not object-code-compatible. If you compile and link separately, be sure to specify the target model on both command lines.

3.1 Global Data Parallel Models

The global models describe a program executed by a single control processor and a set of system components that serve as parallel processors. The parallel processors may be either the nodes or the vector units.

3.1.1 CM-5 SPARC Nodes Model

In this model, the partition manager serves as the control processor, and the node microprocessors are the parallel processors. If the system has vector units, they serve merely as memory controllers and do not participate in the processing of parallel data. Code for parallel computations is generated in SPARC assembler.

To compile and link for the SPARC nodes model:

```
% cmf -cm5 -sparc myfile.fcm
```

3.1.2 CM-5 Vector-Units Model

In this model, the partition manager serves as the control processor, and the vector units are the parallel processors. The SPARC microprocessors on the nodes are invisible to the program, although they assist the vector units with OS services and communication. Code for parallel computations is generated in DPEAC assembler.

Version 2.2, October 1994
Copyright © 1991–1994 Thinking Machines Corporation

To compile and link for the vector-units model:

```
% cmf -cm5 -vu [-nopadding] myfile.fcm
```

3.2 Node-Level Message-Passing Models

The nodal models describe multiple copies of a CM Fortran program that execute independently on the nodes of a CM-5. For each program, the microprocessor in the node serves as the CM Fortran "front end," and the four vector units serve as its parallel processors. The programs use the CM message-passing library CMMD.

3.2.1 CM-5 Nodal Model (Hostless)

In this model, the node microprocessor serves as the control processor and its four vector units are the parallel processors of a CM Fortran program. Code for parallel computations is generated in DPEAC assembler. Communication between programs is explicit via CMMD calls.

To compile and link for nodal CM Fortran (hostless) with CMMD message passing:

```
% cmf -cm5 -vu [-nopadding] -node myfile.fcm
```

```
INCLUDE 'cmmd_fort.h'
```

CM Fortran does not support the combination of **-sparc -node**.

3.2.2 CM-5 Nodal Model (Host/Node)

This model is similar to the hostless model of CM Fortran: the node microprocessor serves as the control processor and its four vector units are the parallel processors. Code for parallel computations is generated in

Copyright © 1991–1994 Thinking Machines Corporation

DPEAC assembler. Communication between programs is explicit via CMMD calls.

The difference from the hostless model is that the programmer writes the host program instead of letting CMMD provide it. The host program cannot be in CM Fortran; it must be in a serial language such as Fortran 77 or C.

To compile and link for nodal CM Fortran (host/node) with CMMD message passing, you must specify on the **cmf** command line the files that contain the host code and the compiler to invoke for them. Identify each host file by preceding its name with the **-host** switch. Use **-comphost** *compiler* to request that the linker treat the host program as if it had been compiled by *compiler*. For example:

```
% cmf -cm5 -vu -node file1.fcm -comphost f77 -host file2.f
```

```
INCLUDE 'cmmd_fort.h'
```

The **cmf** compiler treats the **-host** source files as indicated by the filename suffixes. For example, a **.c** host file is compiled using **cc**; a **.o** host file is simply passed to the linker.

If the compile line is:

```
% cmf -cm5 -vu -node -comphost f77 -host foo.c -host bar.o
```

then the host file **foo.c** is compiled by the **cmf** driver using **cc**, resulting in **foo.o**. The object files **foo.o** and **bar.o** are passed to the linker along with **-comphost f77**. The linker links the host program object files using **f77**.

Note that the host/node model is available only on CM-5 systems with vector units, and the **-node** switch must be specified.

3.3 The Global/Local Model

The global/local model describes a CM Fortran program that executes in global data parallel fashion across the entire partition of a CM-5, but which

Version 2.2, October 1994
Copyright © 1991–1994 Thinking Machines Corporation

can call local subroutines that run independently on the nodes. A copy of the local routine runs on each microprocessor, using the four associated vector units as its parallel processors. The global program uses transparent compiler-generated communication; the local programs use the CM message-passing library CMMD.

The global program must be in CM Fortran; the local routines can be in either CM Fortran or C (to facilitate direct calls to C/DPEAC). Global/local programming is described more fully in the *CM Fortran Libraries Reference Manual* and the *CM Fortran Programming Guide*.

To compile and link for the global/local model, you must specify on the **`cmf`** command line the files that contain local code and supply a *prototype* file (described in the *CM Fortran Programming Guide* and the *CM Fortran Libraries Reference Manual*) that defines the interface between the global and local portions of the program.

Copyright © 1991–1994 Thinking Machines Corporation

For example, assuming a CM Fortran subprogram:

```
% cmf -cm5 -vu global.fcm -local local.fcm file.proto
```

in **local.fcm**:

```
  INCLUDE 'cmmd_fort.h'
```

in **global.fcm**:

```
INCLUDE 'cmgl.h'
```

Assuming a C subprogram:

```
% cmf -cm5 -vu global.fcm -local local.c file.proto
```

in **local.c**:

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

in **global.fcm**:

```
INCLUDE 'cmgl.h'
```

A program can also be compiled incrementally. For example,

```
% cmf -cm5 -vu global.fcm -local local.fcm file.proto
```

is equivalent to the following steps:

```
% cmf -cm5 -vu global.fcm
% cmf -cm5 -vu local.fcm
% cmf -cm5 -vu global.o -local local.o file.proto
```


Version 2.2, October 1994
Copyright © 1991–1994 Thinking Machines Corporation

3.4 CM Fortran Simulator

In the simulator model, a CM Fortran program executes on a single Sun-4 processor, which may be a CM-5 partition manager or a stand-alone workstation. This model is convenient for developing and debugging programs without tying up parallel processing resources.

To compile and link for the CM Fortran simulator:

```
% cmf -cmsim myfile.fcm
```

Notice that you do *not* specify a hardware platform, like **-cm5**, or the execution model you eventually intend to use for the program, such as **-vu** or **-sparc**. The compiler generates both serial and parallel code as usual, but links with a library that causes all operations to be performed sequentially.

You can also compile a program for execution under the simulator and the Prism development environment, and examine it with Prism's debugging and performance analysis tools:

```
% cmf -cmsim [ -g | -cmprofile ] myprogram.fcm
```

See Section 4.6 for more information about the CM Fortran simulator.

Copyright © 1991–1994 Thinking Machines Corporation