Chapter 2

Compiling and Linking CM Fortran Programs

This chapter explains how to use the **cmf** command to compile and link CM Fortran programs.

The **cmf** command operates much like commands for other compilers running under the UNIX operating system. It can be used to compile source files of different types and link them with object files. The command first invokes one or more of the following, as appropriate: the CM Fortran compiler, the C language preprocessor **cpp** (if the filename has the extension **.FCM** or **.F**), the compilers **f77**, **cc**, or **cs**, the Sun assembler **as**, or the vector-units assembler **dpas**. Next, it invokes either the linker **cmld** or the linker **cmmd-ld** (for nodal CM Fortran programs) to combine the resulting object files with other object files or libraries to form an executable load module.

Note that **cmf** invokes other compilers only as a convenience; it does not recognize all switches available for these compilers. In cases where unusual switches are desired, you can use the **-Z** switch (see Section 2.3), or call the desired compiler directly, producing an object file that is then given to **cmf** to link.

2.1 The Compiler Command

The **cmf** command invokes the CM Fortran compiler. Its syntax is:

```
% cmf [ switch-or-filename ]...
```


Copyright © 1991–1994 Thinking Machines Corporation

The optional *switch-or-filename* is either a *switch* or a *filename*. Each *filename* specifies a source or object file to be compiled or linked, and each *switch* is a command switch name preceded by a dash, in the usual UNIX style. Some switches require an argument; the form of these arguments is described in Section 2.3. Switches and filenames should be separated by one or more blank spaces; the order of switches and filenames does not matter.

For example:

```
% cmf my-program.fcm -o my-program
```

2.1.1 Filename Extensions

The **cmf** command determines the type of a file from its filename extension and processes the file accordingly. The command accepts filenames with the following extensions:

.fcm	CM Fortran source
.FCM	CM Fortran source (causes **cmf** to invoke **cpp**)
.dp	CM-5 DPEAC source
.s	Sun SPARC assembler source
.f	Sun FORTRAN source
.F	Sun FORTRAN source (causes **cmf** to invoke **cpp** before **F77**)
.c	Sun C source
.cs	C* source
.proto	prototype file used for global/local programs
.o	object file
.a	object library file

Object files may be from a previous invocation of **cmf**. They may also be the output of the Sun FORTRAN compiler **f77** or the Sun C compiler **cc**.

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

2.1.2 Linking with CM Libraries

In order to link, **cmf** must be passed either a **.o** file or a **.a** file on the link line.

CM Fortran Libraries

The CM Fortran utility library is linked automatically when needed. Program units that call procedures from this library need to include its header file **CMF_defs.h**.

```
INCLUDE 'CMF_defs.h'
```

CM **INCLUDE** files are also accessible by absolute pathname or by using the **-I** switch. Check with your site's system administrator.

The include file **CMF_defs.h** contains external declarations for certain functions. Their purpose is to prevent users from inadvertently referencing these functions as variables — for example, assigning the function name, such as, **n = CMF_NUMBER_OF_PROCESSORS**, instead of assigning its return value, **n = CMF_NUMBER_OF_PROCESSORS()**.

However, external declarations are not permitted in block data program units. Therefore, an alternative include file, **CMF_params.h** exists, containing only the parameter definitions from **CMF_defs.h**. Use **CMF_params.h** instead of **CMF_defs.h** in block data program units.

```
INCLUDE 'CMF_params.h'
```

The CM Fortran global/local library is linked explicitly. See Section 3.3 for information about linking for the global/local programming model and about which header files to include.

The CM Fortran library **cmf77** (**libcmf77.a**) requires no special include file or explicit linking. This library provides procedures comparable to those in the Sun library **libF77.a**. If the Sun library is installed on a CM system, the **cmf** command links with it as well as with **libcmf77.a** so that object files compiled with Sun's **f77** command will link successfully.

Copyright © 1991–1994 Thinking Machines Corporation

All three of these libraries are described in the *CM Fortran Libraries Reference Manual*.

CM System Libraries

The following libraries are supported on the CM-5:

- Prism
- CMSSL
- CMMD
- CMX11
- CM/AVS
- CMFS
- SFS

The CM simulator supports no libraries except Prism.

The libraries available may differ among the CM execution environments. See the documentation for the current versions of the libraries for details.

Some of the CM libraries require you to include header files and link explicitly when processing a CM Fortran program that uses them. CMFS and SFS are linked in automatically.

`cmf` is able to link with the appropriate version of the following libraries, based on either the default compiler mode or the use of the **`-vu`**, **`-sparc`**, and **`-node`** switches:

- **`-lcmssl`** (for the CMSSL library)
- **`-lcmx`** (for the CMX11 library)
- **`-lcmavsflow`**, **`-lcmavssim`** (for the CM/AVS libraries)
- **`-lcs`** (for C* codes that call CM Fortran subroutines, which are linked using **`cmf`**)
- **`-lcmfs_cs`** (for calling CMFS from C* code that is linked to a CM Fortran program)

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

For instance, you can now link with CMSSL by specifying **-lcmssl** on the link line instead of a form that indicates which execution model you are using, such as **-lcmsslcm5** or **-lcmsslcm5vu-node**. The old form is still supported, however, so there is no need to change your makefiles.

NOTE: For CMX11 and CM/AVS, you must still specify the non-CM libraries, such as **-lX11** and **-lflow_f**.

INCLUDE Files

The compiler accepts up to 252 **INCLUDE** files per source file, nested to a maximum depth of 19. The total refers to distinct files referenced in all the **INCLUDE** lines in all the compilation units in a source file. If, in the same file, **SUB1** includes **inc1.fcm**, **inc2.fcm**, and **inc3.fcm**, and **SUB2** includes **inc3.fcm** and **inc4.fcm**, the total charged against the 252 is 4.

The CM Fortran **INCLUDE** statement accepts absolute pathnames, such as

```
INCLUDE '/usr/include/cm/CMF_defs.h'
```

in addition to relative pathnames like

```
INCLUDE 'CMF_defs.h'
```

If the locations of standard CM system include files like **CMF_defs.h** are customized on your site, you may want to use relative pathnames. If not, absolute pathnames may be convenient.

The compiler switch **-I** enables the user to specify search paths. See Section 2.3.

2.1.3 Compiler and Linker Output

The **cmf** compiler and the **cmld** linker generate a single output file that combines a scalar executable for the partition manager and a parallel executable for the parallel processors. As intermediate output, however, the compiler generates separate files for the two target components. For example:

- With the **-S** switch, the compiler saves two assembler files: **myfile.s** and **myfile.pe.s**.

Copyright © 1991–1994 Thinking Machines Corporation

- With the **-c** switch, the compiler saves two object files: **myfile.o** and **myfile.pe.o**.
- The linker generates only one executable file: **a.out**. There is no file **a.out.pe** corresponding to the parallel intermediate files.

If you work with intermediate files — explicitly linking object files, for instance — you need only specify the scalar file. The corresponding parallel file is linked automatically.

```
% cmf -cm5 file_1.o file_2.o
```

This command line links by invoking the compiler, which in turn invokes the linker. It is also possible to invoke the **cmld** linker directly. (A CM-5 CMOST man page is provided.)

For more information about the output generated by the **cmf** compiler, see the *CM-5 CM Fortran Performance Guide*.

If you wish to disable the automatic processing of the parallel ("**.pe**") intermediate file when the corresponding scalar file is specified, set the environment variable **CMF_AUTO_PE_FILES** to **0**. Any positive value for this variable leaves the feature enabled.

In either case, recall that the separate intermediate files exist. If you have occasion to copy or move intermediate files to another directory, be sure to take both the scalar and the parallel files.

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

2.1.4 Command Line Arguments

You can write a routine to get command-line arguments by using operating system interfaces from the CM Fortran library **cmf77** (see the *CM Fortran Libraries Reference Manual*): a function **IARGC()** and a subroutine **GETARG(I, NAME)**. You might use them as follows:

```
PROGRAM LOOP
IMPLICIT NONE
INTEGER N1, N2, N3, N4, IARGC
CHARACTER*80 ARGUMENT

IF (4 .NE. IARGC()) THEN
   PRINT *, 'Usage : LOOP N1 N2 N3 N4'
   STOP
END IF

CALL GETARG(1, ARGUMENT)
READ (UNIT=ARGUMENT, FMT='(I6)') N1

CALL GETARG(2, ARGUMENT)
READ (UNIT=ARGUMENT, FMT='(I6)') N2

CALL GETARG(3, ARGUMENT)
READ (UNIT=ARGUMENT, FMT='(I6)') N3

CALL GETARG(4, ARGUMENT)
READ (UNIT=ARGUMENT, FMT='(I6)') N4

PRINT *, IARGC(), N1, N2, N3, N4

STOP
END
```

2.1.5 Sun Microsystems' Libraries

CM Fortran automatically links with Sun Microsystems' libraries if they are installed on the system. This behavior will be discontinued in future versions.

Copyright © 1991–1994 Thinking Machines Corporation

When a program that was linked with Sun Microsystems' dynamically bound libraries does not find them at run time, an error message like the following is displayed:

```
ld.so: libF77.so.1: not found
```

You can prevent this problem by linking statically by means of the **-Bstatic** switch, which **cmf** passes on to the linker:

```
% cmf myfile.fcm -Bstatic
```

2.1.6 Custom Link Libraries

The **cmf** compiler generates separate object files for scalar and parallel code, with the extensions **.o** and **.pe.o**, respectively. To build a custom link library, create two separate libraries for the scalar and parallel code:

- Take all the **.o** files and put them into **libfoo.a**, using the UNIX **ar** command.

```
% ar rv libfoo.a sub1.o
```

- Take all the **.pe.o** files and put them into **libfoo_pe.a**, again using the **ar** command. The underscore in the name is required; a filename of the form **libfoo.pe.a** does not work.

```
% ar rv libfoo_pe.a sub1.pe.o
```

- Add the location of **libfoo.a** to the library search path. When linking a program, you need only specify **libfoo.a** to the link command (**cmf** or **cmld**). The parallel library **libfoo_pe.a** is then linked in automatically.

```
% cmf -lfoo myfile.fcm
```


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

2.2 Some Important Switches

Section 2.3 describes all the switches accepted by the **cmf** command. This section draws your attention to some particularly important switches.

2.2.1 Specifying Execution Environment

Some switches have to do with specifying a hardware-dependent execution model. This section provides a brief overview of these switches. For more detailed information about using them to compile programs, see Chapter 3.

The CM Fortran *execution model* refers to the way the compiler makes use of the hardware. On CM-5 systems, the compiler can treat either the nodes or the vector units as the parallel processing elements. In the simulator model, a single Sun-4 computer is the processor.

Hardware Requirements

- The vector-units model executes only on a CM-5 with vector units. The (SPARC) nodes model executes on any CM-5 system.
- The simulator model executes on a Sun-4 computer, which may be a CM-5 partition manager.

The compiler signals an error if the options specifying hardware platform and execution model are incompatible. Notice that the option **-cmsim** is not compatible with any other option on the list.

The following table shows the compiler switches that control the execution model. It also shows the symbolic names of the execution models and the system component that serves as the processing element.

Copyright © 1991–1994 Thinking Machines Corporation

Table 1. CM Fortran execution models.

Compiler options	CMF_ARCHITECTURE returns	CMF_NUMBER_OF_ PROCESSORS returns
-cm5 -vu	**CMF_CM5_VU**	number of vector units
-cm5 -sparc	**CMF_CM5_SPARC**	number of nodes
-cmsim	**CMF_CMSIM**	number of processors (1)

See Chapter 3 for information about the nodal model and the local part of the global/local execution model (specified by **-cm5 -vu -node** and **-cm5 -vu ... -local**, respectively). Both of these execution models require the **-vu** switch and **CMF_ARCHITECTURE** returns **CMF_CM5_VU**.

NOTE

Object files compiled for different execution models are not compatible. If you link as a separate step, be sure that all the CM Fortran files were compiled for the same execution model.

Defaults

For systems with more than one CM hardware platform installed, the default platform is determined locally at installation time. For each platform (other than simulator), there is also a system-wide default execution model.

- If you specify hardware platform but not execution model on the **cmf** command line, the compiler uses the site default for the missing value.

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

Use the environment variable **CMF_DEFAULT_MACHINE** to change the default hardware platform for your shell environment. Its value is one of the following (case is ignored):

```
% setenv CMF_DEFAULT_MACHINE      [ CM5 | CMSIM ]
```

Unsetting this environment variable causes the shell environment to revert to the site's default hardware platform:

```
% unsetenv CMF_DEFAULT_MACHINE
```

- Use one of the following environment variables to change the default execution model for your shell environment (case is ignored).

```
% CMF_CM5_DEFAULT_MODE        [ SPARC | VU ]
```

Unsetting these environment variables causes the shell environment to revert to the site's default for each of the hardware platforms.

```
% unsetenv CMF_platform_DEFAULT_MODE
```

2.2.2 Listing Array Homes

Mistaking the homes of arrays is a frequent source of user errors and poor program performance. If you inadvertently perform a front-end operation on a CM array, performance degrades as the system moves the array, one element at a time, from its CM home to the partition manager where the operation executes. If you mismatch array homes across subroutine boundaries — passing a CM array to a procedure that expects a front-end array, or vice versa — the result is a run-time error or incorrect results.

The compiler switch **-list** allows you to check the location of particular arrays at compile time. This switch causes the compiler to produce a file *filename*.**lis**, which reports the home and other information about names used in the program being compiled.

Copyright © 1991–1994 Thinking Machines Corporation

For example, consider this code fragment:

```
      LOGICAL, DIMENSION(10) :: CML, FEL
      INTEGER, DIMENSION(10) :: CMI, FEI
      REAL, DIMENSION(10) :: CMR, FER
      COMPLEX, DIMENSION(10) :: CMC, FEC
      ...
```

Assume names beginning with the letters CM in this code fragment are CM arrays, and those beginning with FE are front-end arrays. The "ARRAYS" section of the listing produced for this program fragment indicates the home of each array under the column labeled "Home," as shown below:

```
ARRAYS
  Offset    Size  Type  Block/Class     Home     Name
     ---      80  C*8   local           CM       CMC
     ---      40  I*4   local           CM       CMI
     ---      40  L*4   local           CM       CML
     ---      40  R*4   local           CM       CMR
       0      80  C*8   local           FE       FEC
     120      40  I*4   local           FE       FEI
     160      40  L*4   local           FE       FEL
      80      40  R*4   local           FE       FER
```

The sizes and offsets are given in units of bytes. No offset is listed for CM arrays, as they are allocated at run time.

2.2.3 Identifying Communication Routines Generated

The listing file produced when the **`-list`** switch is specified identifies the communication routine references generated for a program unit, and the source code line numbers at which each reference occurs. For example, the

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

source lines of a (somewhat contrived) program **xref.fcm** would appear in the listing file **xref.lis** as:

```
Source Listing File:    /users/user-name/xref.fcm
   1      1      1      program xref
   2      2      1      parameter (m = 10)
   3      3      1      real a(m), b(m)
   4      4      1      integer v(m)
   5      5      1      a = [1:m]*17.0
   6      6      1      v = [1,4,3,2,7,6,9,8,10,5]
   7      7      1      a(v) = a*3
   8      8      1      print 10, a
   9      9      1      10 format( " A:", 10F9.3 )
  10     10      1      loop: do 100 i=1,m
  11     11      1      b(i) = log(real(i*i*i))
  12     12      1      a(i) = a(i)*b(v(i))
  13     13      1           if (i==9) exit loop
  14     14      1      100   continue
  15     15      1           print 10, a
  16     16      1      200 end
```

The listing file reports the communication routine references as:

```
COMMUNICATION ROUTINES
  Name                   Line Number (number of times)
  READ VALUE FROM PROCESSOR  12(2)
  VECTOR SEND                7
  FE TO CM ARRAY TRANSFER    6
```

The example code generates references to three different communication routines: **READ VALUE FROM PROCESSOR** on line 12, **VECTOR SEND** on line 7, and **FE TO CM ARRAY TRANSFER** on line 6. (**VECTOR SEND** is a general communication routine to handle vector-valued subscripting.) If more than one reference to a communication routine appears on a single line, that number is indicated in parentheses following the line number.

Many of the communication routines support the intrinsic functions directly, and references to them use the name of the intrinsic function itself (possibly qualified), such as **CSHIFT**, **MAXLOC**, **SUM** (into scalar), and **SUM** (into vector). Others refer to common CM communication patterns: **SEND**, **GET**, **VPMOVE**, **NEWS**, and **NEWS** (power of two). Still others refer to data transfers between the CM and the partition manager: **READ VALUE FROM PROCESSOR**, **FE TO CM**

Copyright © 1991–1994 Thinking Machines Corporation

ARRAY TRANSFER, and so on. The listing also reports uses of **SUBROUTINE ARGUMENT COPYOUT**.

When code is optimized with the **-O** switch, **-list** cannot always determine exactly where in the program optimized code originated. If a line number is approximate, it is marked with an asterisk. For example,

```
   COMMUNICATION ROUTINES
     Name                  Line Number (number of times)
     SUM (into scalar)     73*(2)
     SUM (into vector)     53, 73
```

2.2.4 Locating Line Labels and Names

When used together with the **-list** switch, the **-cross_reference** switch causes the listing file to include information that relates line labels and names (symbols) to source code lines. (Add the switch **-show_include** if you want the contents of include files to be listed also.) The **-cross_reference** switch is ignored if the **-list** switch is not specified. The default is **-nocross_reference**. The symbol and label cross reference listings generated for the program listed in the previous section are shown below.

```
Symbol Cross Reference   File: /users/usr-name/xref.fcm
Symbol        Line Number(s)

A             3     5     7     7     8    12    12    15
B             3     11    12
I             10    11    11    11    11   12    12    12    13
LOG           11
LOOP          10    13
M             2     3     3     4     5    10
REAL          11
V             4     6     7    12
XREF          1

Label Cross Reference   File: /users/usr-name/xref.fcm
Label  Defined  References(s)
10          9         8    15
100        14        10
200        16
```


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

2.2.5 Run-Time Safety Checking

To catch array-home errors and other run-time errors, you can direct the compiler to generate code that checks at run time for home mismatches and other mismatches between actual and dummy array arguments, and checks array-valued functions against the interface block in the calling program. The switch is **`-safety=`***level*, where *level* is an integer value indicating the level of run-time safety desired. The key levels are:

`0` No safety checking

`1–9` Checks that:

- the *number* of actual arguments equals the number of dummy arguments
- the *home* of each actual array argument is the same as the home of the corresponding dummy argument
- the *rank* of each CM actual array argument is the same as the corresponding dummy argument
- the *shape* (the extents in rank order) of each CM actual array argument is the same as the corresponding dummy argument
- the *ordering* (**`:NEWS`** or **`:SERIAL`**) of each axis of each CM actual array argument is the same as the corresponding dummy argument

Comparable checks are made for array-valued functions.

Note that the generated code does not check the *types* of arguments or array-valued functions.

`10` Provides the checks above plus checking for out-of-bounds vector-valued subscripts. This argument causes all bits in program memory to be initialized to one. Read as a two's complement integer, this is interpreted as −1; read as a floating-point number, it is interpreted as a NaN.

Copyright © 1991–1994 Thinking Machines Corporation

Any argument of **10** or greater provides these checks.

NOTE: Do not use **-safety** when compiling a code in which one file has the extension **.fcm** and another has the extension **.f** or is an object file compiled by the **f77** compiler. In this case, **-safety** reports the following:

```
*** RTS-FATAL-NUMARGS: The numbers of actual and dummy
arguments differ.
```

2.2.6 Debugging

There are two compiler switches that produce information for the Prism debugger, **-g** and **-cmdebug**. The **-g** switch is recommended for most situations. (**NOTE:** Either of these switches or the **-cmprofile** switch links in the Prism library.)

The difference between the two switches is in their ability to suppress certain optimizations the compiler ordinarily performs. The compiler usually fuses multiple statements together, which increases execution speed but makes it difficult to relate debugger output to program source. The **-g** switch suppresses the fusing of source code statements, which makes debugging easier, although it degrades program performance significantly. The **-cmdebug** switch permits statements to be fused; this switch should be used only when you want to debug fully optimized code (a difficult task).

If you suspect a program error that the Prism debugger is unable to locate, try compiling with both **-g** and the run-time safety switch, **-safety=10**. Since run-time safety checking degrades performance even further, programmers usually avoid using it on the unoptimized code produced by **-g**. However, the combination of switches is sometimes useful in finding especially subtle bugs.

Using **-cmdebug** causes all front-end scalars to have memory homes. This can result in a slight degradation in front-end code performance. It also inhibits some dead code removal which may, in some circumstances, increase the number of code blocks. Overall, the overhead is around 1–2% on heavily parallel code.

The **-g** switch, on the other hand, works by putting each statement in its own parallel code block. This can result in a factor of 2 (or more) decrease in performance.

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

2.2.7 Performance Analysis

To produce information for performance analysis under the Prism development environment, use the **-cmprofile** switch. If you use this switch, include it during both compilation and linking.

By default, the **-cmprofile** switch activates the **-cmdebug** switch, causing Prism to analyze performance on a block-by-block basis.

See Section 4.3 for more information about using Prism.

2.2.8 Controlling Axis Sequence

The compiler transforms all array section code into one or more **DO** loops (collapsing the loops on more than one array dimension where possible). The dimension whose element indices vary fastest in local memory becomes the innermost loop. This loop is vectorized on a CM-5 with vector units; the vector units process eight "data objects" for each iteration of a loop on the vector axis.

CM Fortran provides the compiler switch **-[no]axisreorder**, which allows you to control which array dimension varies fastest in memory. The negative form **-noaxisreorder** causes the compiler to lay out the dimensions in the order you declare them, with the leftmost axis the fastest varying.

Consider arrays with two serial axes:

```
      REAL A(10,20,100), B(10,20,100)
CMF$  LAYOUT A(:SERIAL,:SERIAL,), B(:SERIAL,:SERIAL,)
      ...
      A(1,:,:) = B(2,:,:)
```

With **-noaxisreorder**, the middle axis has stride 10 in local memory. The double loop that implements this assignment vectorizes along the middle axis.

Copyright © 1991–1994 Thinking Machines Corporation

In previous releases, the compiler always laid out axes in memory in the following way, reordering them from the declared order if necessary:

- All **:NEWS** axes varied faster than any **:SERIAL** axis. (This guaranteed that a section formed by scalar indexing into all serial axes would be a contiguous block of memory.)
- The **:NEWS** axis declared rightmost varied faster than other **:NEWS** axes, the next rightmost **:NEWS** axis varied next fastest, and so on.
- The **:SERIAL** axis declared rightmost varied faster than other **:SERIAL** axes, the next rightmost **:SERIAL** axis varied next fastest, and so on.

For back-compatibility, this axis reordering is still the default. That is, the switch defaults to the positive form **-axisreorder**, and therefore the vector axis is always a **:NEWS** axis.

The current release allows you to choose the axis order in memory. When you supply the negative form **-noaxisreorder**, array dimensions are laid out in column-major order, with the leftmost declared axis the fastest-varying.

NOTE

All CM Fortran program units of a program must be compiled with the same setting of **-[no]axisreorder**.

Note that **-axisreorder** does not affect purely serial arrays, that is, arrays that do not reside on the CM at all. For those arrays, axes are ordered from left to right, regardless of whether **-[no]axisreorder** is used.

2.2.9 Controlling Vector-Length Padding (CM-5 with VUs only)

The **-[no]padding** switch determines whether subgrids — the on-processor portion of arrays — are padded to a multiple of vector length under the vector units execution model. Removing the padding constraint

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

gives the user added control over which data elements are co-resident in local memory.

When allocating memory for CM arrays, the CM run-time system always allocates the same memory locations in every processing element (node or vector unit, depending on the execution model). As a result, the size of a *virtual grid* — the block of distributed memory onto which a user array is mapped — is always an integer multiple of the value returned by the utility function **`CMF_NUMBER_OF_PROCESSORS()`**. If the product of an array's non-serial dimensions is not a legal size for a virtual grid, the system allocates the next legal size (and later masks out the unused memory locations). The system then lays out the array's parallel dimensions within the virtual grid so as to minimize the amount of this *global padding.*

Before Version 2.1 — and by default in the current version — the run-time system in the VU model met a second constraint in determining the grid size required for parallel dimensions: the on-processor subgrid had to be a multiple of the length of a vector register. As a result, machine geometry sizes (excluding serial dimensions) were integer multiples of 8 times the number of processors.

The SPARC nodes model, lacking vector registers, does not generate this *vector-length padding.*

After the system determines the grid size and layout needed to accommodate the array's parallel dimensions, it extends the per-processor subgrid by the product of the array's serial dimensions. Since serial dimensions are added after the padding has been determined, they are never padded.

The **`cmf`** compiler provides a switch, **`-nopadding`**, by which you can specify that the run-time system not pad virtual grids to a multiple of vector length. (*Global padding is not affected by the switch*; you can avoid it by declaring arrays that are evenly divisible by machine size.)

For back-compatibility, padding is enabled by default. Supply **`-nopadding`** explicitly to avoid vector-length padding.

Copyright © 1991–1994 Thinking Machines Corporation

NOTE

When compiling program units, only the setting of **-[no]padding** used in compiling the main program unit is used. That is, only that setting is passed to the run-time system to govern array layout decisions. Contrary settings in other program units are ignored.

For programs compiled without a CM Fortran main program, the operative setting is governed by an argument to the procedure **CMF_INIT**, described in Section 6.4.

2.2.10 Optimization

The global optimizer (**-O** switch) optimizes intermediate code, by performing dead-code elimination and copy propagation (along with common subexpression elimination and some code motion). Copy propagation is one pass, useless assignment elimination is a second pass.

Dead-Code Elimination

As an example of dead-code elimination, consider this subroutine:

```
SUBROUTINE FOO(C,A,B,N)
IMPLICIT NONE
INTEGER N
REAL *8 C(N),A(N),B(N)
REAL *8 TMP(N)

TMP = A + B
C = TMP

RETURN
END
```

The compiler generates 2 loads and 1 store for this subroutine.

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

Copy Propagation

As an example of copy propagation, consider these assignments (which do not copy propagate):

```
A = A + 1
A = A + 1
A = A + 1
```

If we copy propagate this, we get

```
A = A + 1
A = (A + 1) + 1        ! one copy prop
A = ((A+1) + 1) + 1    ! two copy props
```

If we now evaluate this, assuming **A** is 0 on entry, we get 3 from the first set of statements and 6 from the second. If, however, we have

```
B = A + 1
B = B + 1
A = B + 1
```

we can copy propagate to

```
B = A + 1
B = (A + 1) + 1
A = ((A + 1) + 1) + 1
```

and now the two stores to **B** can be killed, and we get **A = 3** when we execute it. In essence, you can't copy past the point at which any quantity on the right-hand side is modified. In the second example, the statement **B = B + 1** satisfies this condition because we can replace **B** with **A + 1** and still have a correct program with no dead code removed.

Copyright © 1991–1994 Thinking Machines Corporation

2.3 Compiler Switches

This section describes the switches accepted by the CM Fortran compiler. Switches are applied from left to right; if the same switch is specified several times on a command line, the last setting or value is used. Abbreviations are permitted provided that sufficient characters are used to name the switch unambiguously.

The following switches are accepted by the **cmf** command and passed to the CM Fortran compiler, to another compiler, or to the linker. The **cmf** command may conceivably fail if given options that only a foreign compiler accepts. To prevent this happening, use the **-Z** switch to pass the desired switches.

-c Suppresses the linker. Produces an object file for each source file.

-dryrun
Shows the commands that **cmf** builds, but does not execute them.

-D*name* | **-D***name*=*def*
Defines the symbol name for use by the C language preprocessor **cpp**. The first form sets the value of *name* to 1; the second form sets its value to *def*. (This option has the same effect as the C preprocessor directive **#define**.)

-g Produces additional symbol table information for use by the Prism debugger and suppresses the optimization of fusing source code lines into code blocks. This allows the debugger to step through source code line by line. This switch may drastically slow performance. Please refer to **-cmdebug** for more information.

-h Displays a summary of the options on standard output. If this option is used, other options and arguments are ignored.

-I*dir* Adds the directory *dir* to the list of directories to be searched for include files specified with relative pathnames (pathnames not beginning with "/"). Include files are sought first in the current directory, then in **-I** paths in the order specified by the user on the **cmf** command line, then in **-I** paths specified at installation

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

time, then in the directory containing the source file (if different from current directory), and finally in the default directories under **/usr/include**.

The user-specified **-I** switches are passed through to the C compiler **cc** and preprocessor **cpp**, which treat them in their own way. The **cmf** compiler does not pass **-I** paths specified at CM Fortran installation time through to **cc** or **cpp**.

-L*directory*

Adds *directory* to the list of directories containing object-library routines (for linking using **cmld** or **cmmd-ld**).

-l*lib* Adds *lib* to the list of libraries to be searched during linking. All user libraries are searched before any system libraries; user libraries are searched in the order in which they appear on the command line. The arguments are passed to the **cmld** linker or the **cmmd-ld** linker (for nodal CM Fortran programs), which searches the standard path (**/usr**, **/usr/lib**, **/usr/local/lib**) for a library with the name *lib* to add to the list (the linker adds a **lib** prefix and a **.a** suffix).

Using a library filename *libraryname* **.a** on the **cmf** command line adds the library file to the list to be searched. This is used for libraries that are not in the standard search path.

-O Performs optimization during compilation. (This takes effect even if the **-g** switch is also specified.)

-o *output*

Names the executable file produced by the linker *output* rather than the default name **a.out**. Note the space between **-o** and *output*.

-pg Produces profiling code. A run-time routine accumulates counts and a **gmon.out** file is produced. The linker searches profiling libraries instead of the standard libraries. An execution profile can then be generated by using **gprof**.

-q Suppresses messages describing the progress of the compilation or link.

Copyright © 1991–1994 Thinking Machines Corporation

-S Produces an assembly language representation of each source file.

-v Shows each command built by **cmf** before executing it.

-version
Prints compiler and driver versions.

-Z*cmp* *"switches"*
Passes a list of switches to the compiler or linker specified by *cmp*. The name *cmp* must immediately follow the **-Z** switch, but must be separated by a space from the *switches*. The *switches* should be enclosed in quotes (") and should appear as they would if passed directly.

-cm5 Compiles for a CM-5 system. This switch may not be necessary if the compiler driver is configured so that the CM-5 is the default target.

-cmsim
Compiles for the CM Fortran simulator. This switch may not be necessary if the compiler driver is configured so that the simulator is the default target.

-[no]axisreorder default: **-axisreorder**
The positive form specifies that axes are reordered such that all NEWS axes vary faster than all serial axes, the rightmost NEWS axis varies faster than other NEWS axes, and the rightmost serial axis varies faster than other serial axes. The negative form specifies that array axes are laid out in memory in the order declared, with the leftmost axis varying fastest. The default positive form is back-compatible with previous releases. All CM Fortran program units of a program must be compiled with the same setting of **-[no]axisreorder**.

-[no]cmdebug default: **-nocmdebug**
Produces additional symbol table information for use by the Prism debugger. When used without the **-g** switch, several parallel assignment statements may happen during one single step command issued to the debugger. This switch should be

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

used in conjunction with the **-g** switch if single stepping on each parallel assignment is desired.

-[no]cmprofile default: **-nocmprofile**

Produces information needed for performance analysis under the Prism development environment. If used, this switch must be used during both compilation and linking.

By default, the **-cmprofile** switch activates the **-cmdebug** switch, causing Prism to analyze performance on a block-by-block basis (with source code lines fused together). To analyze a program on a line-by-line basis, relating performance to individual source code lines, specify the **-g** switch along with **-cmprofile**. The switch **-g** suppresses the optimization of fusing source code lines into code blocks; note that this action makes the code execute artificially slowly.

-comphost *compiler*

Requests that the linker treat the host program (identified by the **-host** switch) as if it had been compiled by *compiler*. Currently, *compiler* can be either **cc** or **f77**. This switch is available only on CM-5 systems with vector units, and the **-node** switch must also be specified.

-continuations=*number* default: **-continuations=19**

Specifies the maximum number of continuation lines in a Fortran statement. The *numbe*r must be a non-negative integer value; by default, a Fortran statement can be continued on up to 19 lines.

-[no]common_initialized default: **-nocommon_initialized**

Allocates a copy of a CM array in common on the partition manager. The copy is necessary only when a CM array in common is initialized via block data. (A compiler directive is also provided. Refer to the *CM Fortran Language Reference Manual* for information on directives.)

-[no]cross_reference default: **-nocross_reference**

When used together with the **-list** switch, causes the listing file to include information that relates line labels and names (symbols) to source code lines. Add the switch **-show_include** to cause this information to be generated for include files also.

Copyright © 1991–1994 Thinking Machines Corporation

If the **-list** switch is not specified, the **-cross_reference** switch is ignored.

-[no]d_lines default: **-nod_lines**

Specifies whether lines beginning with a **D** in column 1 are to be compiled as ordinary statements, as if the **D** were not present, or treated as comments. This option allows "debug" statements to be incorporated into a program unit for testing; the statements can in effect be removed without changing the source code.

-[no]directive default: **-directive**

Indicates whether comment lines that have the syntax of compiler directives should be processed as directives, or regarded as comments. (Refer to the *CM Fortran Language Reference Manual* for information on directives.)

-[no]double_precision default: **-nodouble_precision**

Indicates whether to interpret the type specifications **REAL** and **COMPLEX** as double-precision data or single-precision data.

-[no]extend_source default: **-noextend_source**

Extends the source line length to 132 characters (from 72 characters).

-[no]f90syntax default: **-nof90syntax**

Controls the compiler's interpretation of arguments in certain intrinsic functions, including argument order when argument keywords are not used. The functions affected are **AINT**, **ANINT**, **REAL**, **CMPLX**, **RESHAPE**, **CSHIFT**, and **EOSHIFT**. With the negative form, the default, the compiler interprets function references as in previous releases of CM Fortran (based on an early draft of Fortran 90); with the positive form, the compiler interprets function references as defined in the Fortran 90 standard. The default will change in a future version.

-[no]fecommon default: **-nofecommon**

Specifies the default allocation of common arrays. Ordinarily, the CM Fortran compiler allocates common arrays (other than character arrays) in CM memory. This switch changes the default allocation of common arrays to be in front-end memory.

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

(Compiler directives may force arrays to be allocated in front-end memory or CM memory. Refer to the *CM Fortran Language Reference Manual*.) Programs that use *only* Fortran 77 language features are *required* to set this switch for correct execution.

A common array that has been allocated in front-end memory cannot be used in an array operation. Thus, some programs that compile correctly in the absence of **-fecommon** may fail if the option is set.

-host *filename*

Identifies *filename* as being part of the host program, for the host/node programming model. The linker links the host program object files as specified by the switch **-comphost**. The **-host** switch must be repeated for every host file. This switch is available only on CM-5 systems with vector units, and the **-node** switch must also be specified.

-[no]implicit_none default: **-noimplicit_none**

Inhibits implicit typing of variable, constant, and function names appearing in the program. Otherwise, undeclared names beginning with the letters **I** through **N** are assumed to be **INTEGER**, and other names are assumed to be **REAL**. The effect of this option is to require explicit declaration of all names. This option overrides **IMPLICIT** statements appearing in the program unit.

-[no]list default: **-nolist**

Produces a source file listing named *filename***.lis**. The listing contains the source, information about variables used, and information about the communication routines generated from the source.

-local *filename*

Links *filename* as a local subprogram to a global CM Fortran program. The **-local** switch must be repeated for every local file. See the *CM Fortran Libraries Reference Manual* for full information about compiling and linking global/local programs. This switch requires **-vu**; it is incompatible with **-sparc**.

Copyright © 1991–1994 Thinking Machines Corporation

-node Links for the CM-5 nodal execution model. Use this switch only for programs that run separately on each node and communicate via CMMD calls. See Section 3.2 for information about compiling and linking nodal programs. This switch requires **-vu**; it is incompatible with **-sparc**.

-[no]padding default: **-padding**

On CM-5 systems with vector units, specifies whether an array's per-processor subgrid (excluding serial dimensions) needs to be an integer multiple of 8 (the vector-length). If so, the subgrid may be padded with garbage elements to reach a legal size. The switch has no effect under the CM-5 SPARC nodes model (which does not add vector-length padding). The default positive form is back-compatible with previous releases.

-safety=*number* default: **-safety=0**

Causes code to be generated that performs some run-time checking. At level **safety=0** (zero), no run-time checking is performed. At level **safety=1**, code is generated that detects some mismatches between actual and dummy arguments and also compares array-valued functions with the results described in the interface block in the calling program. In particular, the run-time code checks that the number of actual arguments equals the number of dummy arguments, the home of each actual argument is the same as the home of the dummy argument, the rank of each CM actual array argument is the same as the dummy argument, the shape (the extents in rank order) of each CM actual array argument is the same as the dummy argument, and the ordering (**:NEWS** or **:SERIAL**) of each axis of each CM actual array argument is the same as the dummy argument. Comparable checks are made of array-valued functions. At level **safety=10**, all run-time safety checks are enabled. Currently these are argument and function checking (as listed above), and checking for out-of-bounds vector-valued subscripts. Note that the generated code does not check the types of arguments or of array-valued functions.

-[no]show_include default: **-noshow_include**

Causes the listing file to contain text from **INCLUDE** statements

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

in the source code. This option has no effect if the **-list** option is not specified.

-[no]sparc

Compiles and links for the CM-5 nodes only (SPARC) execution model. The output module can execute on any CM-5 system.

-[no]veccode default: **-noveccode**

Produces an assembly language representation of the parallel part of each source file for the CM-5 (vector unit model only). The output file will have an extension **.pe.dp**.

-[no]vu

Compiles and links for the CM-5 vector unit execution model. The output module can execute only on CM-5 systems with vector units.

-[no]warning default: **-warning**

Suppresses warning messages during compilation. When enabled, these warnings are written to the listing file if the **-list** option is also specified; otherwise, they are written to standard output.

Copyright © 1991–1994 Thinking Machines Corporation