Chapter 6

Language Interoperability

6.1 Calling F77 from CM Fortran

It is possible to combine CM Fortran program units with program units compiled by the Sun Fortran 77 compiler **`f77`**. There are no special conditions or restrictions, other than insuring that the same settings of switches (for example, **`-safety`**) are the same for all program units.

6.1.1 Linking with f77 Object Files

Sun Microsystems' **`f77`** compiler and **`cmf`** have incompatible calling conventions for functions returning character strings. The **`f77`** compiler passes and expects the address of the character string, the size of that string, and the explicit arguments. The **`cmf`** compiler, however, passes and expects only the address of the character string, followed by the explicit arguments; the size of the character string is not passed.

A workaround exists if the caller is compiled with **`cmf`** and the callee is compiled with **`f77`**. Simply pass an additional argument, specifying the length of the character string to be returned.

6.2 Calling C from CM Fortran

It is possible to combine CM Fortran program units with program units compiled by the Sun C compiler **`cc`**. When combining CM Fortran with C, be careful to follow the naming conventions established for the Sun C compiler.

Copyright © 1991–1994 Thinking Machines Corporation

Since the C language distinguishes identifiers with the same spelling but different case, it is important that C routine names to be referenced from CM Fortran be spelled with the proper case. Any C routine to be called from CM Fortran must declare each argument as a *pointer* to the appropriate type to accomplish the effect of passing arguments by reference. The correspondence between dummy argument declarations in CM Fortran and in C is as follows:

```
INTEGER I           int* i
REAL R              float* r
DOUBLE PRECISION D  double* d
COMPLEX C           struct {float real, imag;}* c
DOUBLE COMPLEX CD   struct {double real, imag;}* cd
CHARACTER CH*(*)    (front-end-specific)
```

A front-end array is passed by reference and should be declared as a pointer to an array of the appropriate type. CM arrays are passed by descriptor and are declared as arguments of the pointer type **`CMSS_desc_ptr_t`**. This and other type definitions may be found in the header file **`CMSS_object.h`**.

```
INCLUDE 'CMSS_object.h'
```

Within CM Fortran code, references to procedures implemented in C should be preceded by an underscore. The program below calls the subroutine **`_ARGS`**, passing it integer, real, and character string arguments.

```
      PROGRAM ARGSCALL
      INTEGER I, J
      REAL R
      CHARACTER*10 A, B
      I = 10
      A = "twenty"
      R = 30.0
      J = 40
      B = "fifty"
      CALL _ARGS( I, A, R, J, B )
      END
```

The implementation of **`_ARGS`** is given in the following sections.

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

The following example program illustrates valid calls to the subroutines **_CSUBL** and **_CSUBN**, whose C implementation is given in the following subsection.

```
      PROGRAM STRTEST
      CHARACTER*12, A, B
      A = "ABC"
      CALL _CSUBL( A )
      CALL _CSUBL( "ABC" )
      PRINT *
      B = "DEFG" // CHAR(0)
      CALL _CSUBN( B )
      CALL _CSUBN( "DEFG" // CHAR(0) )
      END
```

Strings passed to the subroutine **_CSUBL** need not be null-terminated, but those to subroutine **_CSUBN** must be null-terminated.

Calling Sun C

To conform with Sun C calling conventions, the names of C functions to be called from CM Fortran must be spelled with lowercase letters and must be suffixed by an underscore. Each CM Fortran string passed takes two argument positions. In the normal position, a pointer to the character data is passed. Following all normal arguments, arguments representing the lengths of all strings are passed.

Here is an implementation of the procedure **_ARGS** referenced in the previous section:

```
void args_( ip, ap, rp, jp, bp, an, bn )
int *ip, *jp;
float *rp;
char *ap, *bp;
int an, bn;
{
    printf( " %d %.*s %f %d %.*s\n",
              *ip, an, ap, *rp, *jp, bn, bp );
}
```


Copyright © 1991–1994 Thinking Machines Corporation

Here is an implementation of subroutines **_CSUBL** and **_CSUBN**, which simply print their string arguments and return:

```
void csubl_( s, len )
     char *s;
     int  len;
{ /* If <s> is not null-terminated */
  printf( "FORTRAN String: "%.*s"\n", len, s );
}

void csubn_( s )
     char *s;
{ /* assumes <s> is null-terminated! */
  printf( "Null-terminated String: "%s"\n", s );
}
```

The name **sparc** is predefined during C preprocessing. This can be useful when writing front-end-independent C code using the **#ifdef** preprocessor command. See Section 4.5.2 for a list of other predefined symbols.

6.3 Calling C* from CM Fortran

You can call C* from CM Fortran and vice versa. See the *C* User's Guide* for an example. When linking, be sure to include the C* library: **-lcs**.

Copyright © 1991–1994 Thinking Machines Corporation

6.4 Calling CM Fortran from F77 and C

To call CM Fortran from F77 or C, you must first initialize the parallel processing unit, using the CM Fortran utility library procedure **CMF_INIT**. You must call this procedure before calling any CM Fortran subprogram from a program compiled with a compiler other than **cmf**. This requirement applies to programs that run globally (on the CM-5 partition manager) or locally (on the individual nodes).

The **CMF_INIT** procedure is useful for calling CM Fortran kernels from serial Fortran, C, or other languages with C-compatible calling conventions. Both Fortran and C versions of the initialization procedure are provided.

Fortran syntax:

```
CALL CMF_INIT (PADDING, AXISREORDER)
```

C syntax:

```
void CMF_init (int padding, int axisreorder)
```

For more information on **CMF_INIT**, see the *CM Fortran Libraries Reference Manual*.

Copyright © 1991–1994 Thinking Machines Corporation