Chapter 5

Compiling, Running, and Debugging Programs That Contain CMFS Code

This chapter provides information about compiling, running, and debugging CMFS code by presenting a sample program in both C* (see Figure 4) and CM Fortran (see Figure 5). These programs are skeletons of an image-processing program, a data-intensive application well suited to the CM-5 system. Each element of a parallel variable on the CM-5 represents a pixel; each pixel's color is described by a 32-bit number. To enhance the image, calculations are performed simultaneously within the parallel variable.

The program takes two input parameters: an input filename and an output filename. The program's input file comprises thousands of 32-bit pixels collected by field equipment and placed on tape; the data from the tape is transfered to a file on a UNIX file system. If that file system is not accessible to the CM-5, the file is transferred to a file system that is accessible. The program then reads the data from the input file into the parallel variable on the CM-5, manipulates the data, and then creates an output file and writes out the enhanced data to it.

Copyright © 1994 Thinking Machines Corporation

```
/*                  imageEnhance Program                */
/* Error checking & looping are omitted for brevity.  */

# include <stdio.h>
# include <cm/cmfs.h>
# include <cm/cm_file.h>
# define INPUT_FILE av[1]
# define OUTPUT_FILE av[2]
void
main(ac,av)
int ac;
char *av[];
{
    shape [1024] [1024] s;
    int:s pvar;
    int       image_fd, enhanced_image_fd;
    int       bytes_read, bytes_written;
    float:s   image_calc;    /* the parallel variable  */
                             /* in which the image    */
                             /* enhancement is done.  */

/*  Open and read the input file into the CM-5.       */

    image_fd = CMFS_open(INPUT_FILE, CMFS_O_RDONLY);
    bytes_read = CMFS_read_file_always(image_fd,
         &image_calc, boolsizeof(pvar));

/*  Code to manipulate the image would be here.       */

/*  Save the enhanced image to a new file.            */
    enhanced_image_fd = CMFS_open(OUTPUT_FILE,
         CMFS_O_CREAT|CMFS_O_WRONLY|CMFS_O_TRUNC, 0666);
    bytes_written =
         CMFS_write_file_always(enhanced_image_fd,
         &image_calc, boolsizeof(pvar));

/*  Close the input file and the output file.         */
    CMFS_close(image_fd);
    CMFS_close(enhanced_image_fd);
}        /* End of program. */
```

Figure 4. Sample program: C* with CMFS routines.

Copyright © 1994 Thinking Machines Corporation

```
C     imageEnhance PROGRAM
C     ERROR CHECKING & LOOPING ARE OMITTED FOR BREVITY.
      PROGRAM IMAGE_ENHANCE

      INTEGER CMFS_WRITE_FILE_ALWAYS, CMFS_OPEN
      INTEGER CMFS_READ_FILE_ALWAYS, CMFS_CREAT
      INTEGER CMFS_CLOSE
      INTEGER IMAGE_FD, ENH_IMAGE_FD

C     THE PARALLEL VARIABLE IN WHICH IMAGE EHANCEMENT IS DONE.
      INTEGER BUFFER(1024,1024)

      INTEGER BYTES_READ, BYTES_WR, N
      CHARACTER*4 INPUT_FILE
      CHARACTER*4 OUTPUT_FILE

      PRINT *,'ENTER FILE TO READ: '
      READ *, INPUT_FILE
      PRINT *,'ENTER FILE TO CREATE: '
      READ *, OUTPUT_FILE

C     OPEN AND READ THE INPUT FILE INTO THE CM-5.
      IMAGE_FD = CMFS_OPEN(INPUT_FILE, 0, O'444')
      BYTES_READ = CMFS_READ_FILE_ALWAYS(IMAGE_FD, BUFFER, 32)

C     CODE TO MANIPULATE THE IMAGE WOULD BE HERE.
      BUFFER = BUFFER*4

C     SAVE THE ENHANCED IMAGE TO A NEW FILE.
      ENH_IMAGE_FD = CMFS_CREAT(OUTPUT_FILE, O'666')
      BYTES_WR = CMFS_WRITE_FILE_ALWAYS(ENH_IMAGE_FD, BUFFER,
     *                                  32)

C     CLOSE THE INPUT FILE AND THE OUTPUT FILE.
      N = CMFS_CLOSE(IMAGE_FD)
      N = CMFS_CLOSE(ENH_IMAGE_FD)
      STOP
      END
```

Figure 5. Sample program: CM Fortran with CMFS routines.

Copyright © 1994 Thinking Machines Corporation

5.1 Preparing to Run Your Program

5.1.1 Our Sample Environment

Suppose our CM-5 system is the one shown in Figure 6, and that you are working from a workstation logged on to a CM-5 control processor, **cp1**. Your environment variables are set as follows:

- **DVHOSTNAME** is set to **dv1**.
- **DVWD** is set to **big_project**.
- **CMFS_PATHTYPE** is set to **mixed**.

Let's assume that the tape containing input data has been **tar**'ed off into a UNIX file — **pixel_data** — residing on the device **iop1**.

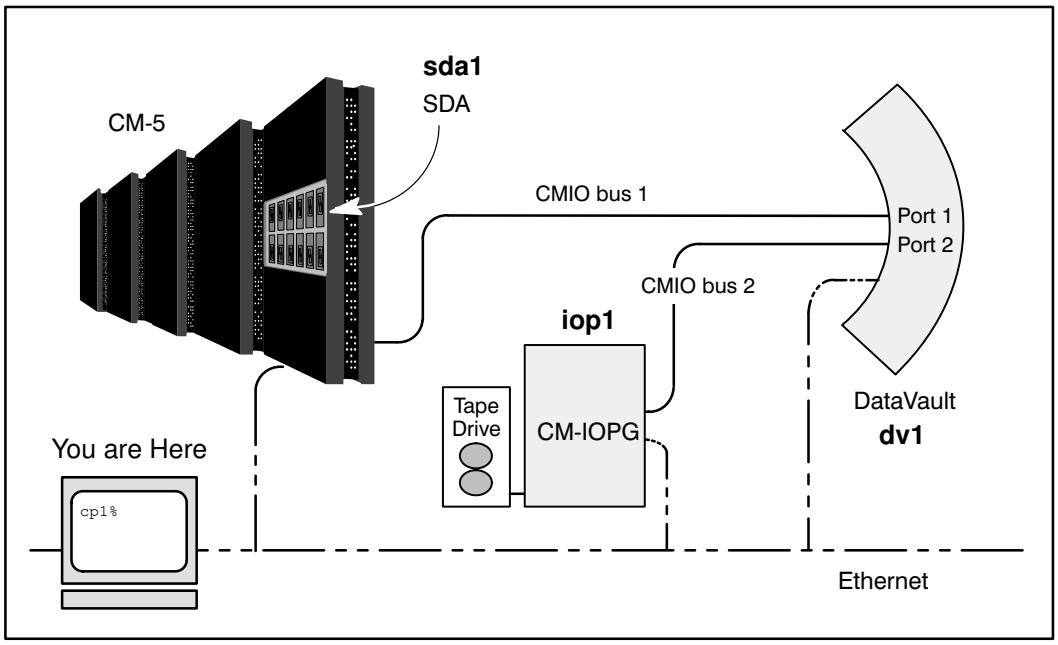

Figure 6. Our sample CM-5 system.

5.1.2 Using the SDA

Assuming the environment variables are set as described above, to use the SDA, first copy the input file from the UNIX file system on the CM-IOPG to the DataVault's CMFS file system, and then copy the file into the SDA's SFS

Copyright © 1994 Thinking Machines Corporation

file system (this method of copying uses the fast CMIO bus connections rather than the slow Ethernet):

```
cp1% rlogin iop1
iop1% copytodv iop1:/pixel_data :pixel_data
iop1% logout
cp1% copyfromdv :pixel_data pixel_data
cp1%
```

The input file **pixel_data** now resides in your current working directory on the SDA.

5.1.3 Using the DataVault

To use the DataVault **dv1**, first copy the input file from the UNIX file system on the CM-IOPG to the DataVault's CMFS file system (in **/big_project**):

```
cp1% rlogin iop1
iop1% copytodv iop1:/pixel_data :pixel_data
iop1% logout
cp1%
```

The input file **:pixel_data** now resides on the DataVault in **/big_project**.

5.2 Compiling CMFS Code

Compile **imageEnhance.cs** as follows (see Table 6, which addresses language-specific CMFS compilation and linking requirements):

```
cp1% cs imageEnhance.cs -o imageEnhance -lcmfs_cs \
-lcmfs_cm5 [-vu|-sparc]
```
[4]

Upon execution, you can specify whether the program will use the DataVault or the SDA by simply modifying the program's pathname arguments (or the setting of the **CMFS_PATHTYPE** environment variable, as discussed in Section 1.4.1 of Chapter 1).

4. Compile with **-vu** if the program will run on a CM-5 partition that contains vector units.

Copyright © 1994 Thinking Machines Corporation

Table 6. Linking requirements for programs that use the CMFS library routines.

Programming Language	Compilation/Linking Requirements
C	No compilation requirements. Link with **libcmfs_cm5.a**.
C*	Must compile with **<cm/cmfs.h>** if the routines **CMFS_read|write_file_always** are called; otherwise, compile with **<cm/cmfs.h>** to avoid type-checking errors. Always link explicitly with **libcmfs_cs.a** and **libcmfs_cm5.a.**
FORTRAN 77	Link explicitly with **libcmfs_cm5.a.**
CM Fortran	Compiler links automatically with **libcmfs_cm5.a**. No explicit linking required.

NOTE: The CMFS software reserves the use of *CMFS_* , *CMIOC_* , *CMFS*, *CMFS-I*, *CMFS-LISP*, and *dv.* If used for any other purpose, these strings may produce compiler errors.

For example, if you have placed the input file on the SDA, execute the program as follows:

```
cp1% imageEnhance pixel_data rev_pixel_data
```

When created, the outputfile, **rev_pixel_data**, is placed in your current working directory on the SDA.

If you have placed the input file on the DataVault, execute the program as follows:

```
cp1% imageEnhance :pixel_data :rev_pixel_data
```

When created, the output file, **:rev_pixel_data**, is placed in the DataVault's **big_project** directory.

Copyright © 1994 Thinking Machines Corporation

5.3 Debugging CMFS Code

If a CMFS library call does not complete successfully, it returns a special value (usually −1) to the application program and sets **CMFS_errno** to indicate the specific error. See the **CMFS_errno** manual page (in Chapter 10) for a list of error numbers and their corresponding descriptions. Call the library routine **CMFS_perror** to print the error message corresponding to the value of **CMFS_errno**.

5.3.1 Automatic CMFS Debugging Messages

You can activate and deactivate the printing of automatic CMFS debugging messages by setting an environment variable or by calling a CMFS routine.

The CMFS_DEBUG Environment Variable

Activate the environment variable **CMFS_DEBUG** to send CMFS debugging messages to standard output. CMFS debugging messages include the arguments and return values of many of the CMFS library routines that a program calls. (It is not guaranteed, however, that *all* CMFS library calls will be noticed by **CMFS_DEBUG**.)

If you are using a C shell, activate automatic CMFS debugging messages by typing

```
% setenv CMFS_DEBUG 1
```

If you are using a Bourne shell, activate automatic CMFS debugging messages by typing

```
$ CMFS_DEBUG=1
$ export CMFS_DEBUG
```

For convenience, your **.login** file or your **.cshrc** file can activate automatic CMFS debugging messages each time you open a shell to work in the CM-5 programming environment. Deactivate automatic CMFS debugging messages by setting **CMFS_DEBUG** to 0.

Copyright © 1994 Thinking Machines Corporation

The CMFS_set_debug_mode Routine

`CMFS_set_debug_mode` allows you to explicitly control the portions of code for which CMFS debugging messages are printed: if *val*, its only argument, is nonzero, CMFS debugging messages are printed until the routine is called again with a *val* argument of 0.

Copyright © 1994 Thinking Machines Corporation