Chapter 5

CM Fortran I/O

This chapter provides an overview of CM Fortran I/O and describes some of its underlying implementation.

The CM Fortran **READ** and **WRITE** statements support both parallel and serial I/O of parallel data and serial I/O of serial (FE) data, to the Scalable Disk Array (SDA) and the DataVault. Parallel I/O refers to transferring data in multiple streams between the CM processing elements and an external device. Serial I/O takes place in a single stream, and is, therefore, slower. The **READ** and **WRITE** statements are described in the *CM Fortran Language Reference Manual*. See Section 5.3 for information about when they perform parallel I/O.

Front-end data can be written to the SDA or DataVault, including character data, although the transfer is done serially.

The CM Fortran utility library I/O procedures support I/O via CM sockets and devices, such as CM-HIPPI, in addition to performing parallel I/O to the SDA and DataVault. This is no longer the preferred way of performing file I/O. The utility library is described fully in the *CM Fortran Libraries Reference Manual*. See Section 5.4 for more information about using the utility library I/O procedures.

See Section 5.6 for information about what I/O is available from which hardware platforms and execution models.

Copyright © 1991–1994 Thinking Machines Corporation

5.1 File Systems

CM Fortran I/O statements support the following file systems:

- the UNIX file system, which resides on a serial computer
- the SFS, or Scalable File System, which is a UNIX file system that resides on the Scalable Disk Array
- the CMFS, or CM File System, which resides on the DataVault)

See the CM I/O documentation for more information on the two CM file systems.

All CM Fortran I/O statements can be used on files on these file systems.

5.1.1 Specifying a Target File

For systems with both a Scalable Disk Array (SDA) and a DataVault, the file affected by a CM Fortran I/O operation is governed by the file's pathname.

When you perform CM Fortran I/O, using **READ** and **WRITE**, the affected file does not depend on the setting of the environment variable **CMFS_PATHTYPE**, as described in the CM I/O documentation.

When you perform I/O by using the utility library routines and do not specify the DataVault name in the file's path, CMFS *does* use the value of **CMFS_PATHTYPE** to determine which file is affected by the I/O operation.

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

Specifying a Scalable Disk Array File

Files on a Scalable Disk Array (SDA) are managed by the Scalable File System, which shares name space with the UNIX file system. Your system administrator sets up the system so that some directories are mounted to the SDA and other directories are on the UNIX file system. To specify a target file on an SDA, you have two choices:

- Specify the full pathname of a file that is currently on a mounted SDA partition. For example, assume the system administrator has mounted an SFS on the directory **/sda**:

```
OPEN (9, FILE='/sda/dir1/data2')
```

- Or, execute in a directory on the SDA and specify just the filename:

```
OPEN (9, FILE='data2')
```

  When you specify a filename alone, the location of the file is assumed to be the directory that is current when you execute the code.

Specifying a DataVault File

To specify a target file on a DataVault, include the DataVault name in the file's path. The pathname is of the form *dvname*:*pathname*. For example, this code opens the file **/dir1/data2** on a DataVault called **dv1**:

```
OPEN (9, FILE='dv1:/dir1/data2')
```


Copyright © 1991–1994 Thinking Machines Corporation

5.2 I/O Streams and Files

5.2.1 Associating I/O Units with I/O Streams and Files

The UNIX **stdin** and **stdout** are associated in the usual way with unit control specifiers in a CM Fortran program. In a **READ** statement, a unit control specifier of * is always associated with the standard input stream, while in a **WRITE** statement, a specifier of * is associated with the standard output stream. The unit number 5 is preconnected to the standard input stream, while unit number 6 is preconnected to the standard output stream, and the unit number 0 is preconnected to the standard error stream.

NOTE

It is not possible to share open files between CM Fortran and Sun FORTRAN. In particular, an open unit under CM Fortran should not be used in I/O from Sun FORTRAN, and vice versa.

The **UNIT=** specifier of the CM Fortran **OPEN** statement specifies the external unit to be connected with a file. Valid unit numbers are in the range 0 to 2999.

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

5.2.2 Default Association of Unit Numbers and Filenames

The **FILE=** specifier of the **OPEN** statement provides for the explicit association of a unit number with a filename. However, if the **FILE=** specifier is not present, an association is made between a unit number and a default filename.

For a unit number ***nnnn*** (in the range 0000 to 2999) the associated default filename is

fort.*nnnn*

where the *nnnn* in the filename may be from one to four digits.

These naming conventions are chosen to be consistent with the conventions of Sun FORTRAN.

5.2.3 Redirection of Standard I/O Streams

The standard input stream may be redirected from a file by using a switch of the form '**<** *filename*' on the shell command line invoking a CM Fortran program. The standard output stream may be redirected to a file by using a switch of the form '**>** *filename*' on the shell command line invoking a CM Fortran program. For example, the command

```
% reduce < observations.in > reduced.dat
```

executes the program **reduce**, taking input from **observations.in** and sending output to **reduced.dat**.

CM Fortran supports applying the **BACKSPACE** and **REWIND** statements to standard I/O streams (**stdin** or **stdout**, default units 5 and 6) when redirected to files.

Copyright © 1991–1994 Thinking Machines Corporation

5.2.4 Printing the First Character in Output Streams

CM Fortran does not consider any device a printing device and prints or writes the first character in formatted output to **stderr** and **stdout** (default units 0 and 6). It does not consider other carriage control characters.

Consider this example:

```
      PROGRAM OUTPUT
      WRITE(6,100) 'No leading blank'
      WRITE(6,100) ' One leading blank'
100   FORMAT(A)
      END
```

Previous releases gave:

```
% output
o leading blank
One leading blank
```

CM Fortran now gives:

```
% output
No leading blank
 One leading blank
```


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

5.3 When is I/O Parallel?

Parallel I/O refers to transferring data in multiple streams between the CM processing elements and an external device. Thus, it is faster than serial I/O. The **READ** and **WRITE** statements perform parallel I/O if *all* of the following conditions apply:

- The data involved is a CM array.
- The file is on the SDA or DataVault.
- The transfer is unformatted.
- Data being transferred is not subject to an implied **DO** loop.

Rather than using implied **DO** loops, use whole arrays or array sections to get parallel I/O.

Performing a serial transaction on a file unit does not make all other transactions on that file unit serial. For example, you can use

```
WRITE(10) INT1, INT2, REAL1, REAL2
WRITE(10) CM_ARRAY
```

to write a header and then a CM array to a file. The header is written serially, but the CM array is written in parallel.

5.4 I/O via the Utility Library Procedures

The CM Fortran utility library I/O procedures are, for most purposes, redundant with the CM Fortran language statements **READ** and **WRITE**. We recommend that you use the **READ** and **WRITE** statements instead of utility library I/O, except in these situations:

- Use the utility library procedures to perform parallel I/O via sockets or devices, such as CM-HIPPI. **READ** and **WRITE** support parallel I/O only to the SDA or DataVault.
- Use the utility library procedures to read files previously written with utility procedures.

Copyright © 1991–1994 Thinking Machines Corporation

NOTE

Always read a CM file with the same mechanism that was used to write it. That is, read with **READ** if the file was written with **WRITE**, read with **CMF_CM_ARRAY_FROM_FILE_SO** if the file was written with **CMF_CM_ARRAY_TO_FILE_SO**, and so on.

5.5 Implementation Notes

This section provides some miscellaneous information about the implementation of CM Fortran I/O that is useful for program development.

5.5.1 PHYSICAL Keyword for OPEN

In default CM Fortran I/O, arrays are usually written in array element order. This requires a transpose, which takes time and temporary space. I/O using the **PHYSICAL** keyword is faster and consumes less space than regular I/O.

When the following conditions apply, specify the **PHYSICAL** keyword to the **FORM=** specifier in an **OPEN** statement:

- The order of array elements in the file is not important.
- Data is read into arrays of the same shape and layout as those from which it is written.
- Data is written and read using the same execution model and the same machine size and, if using SDA, the same SDA size.

Using **FORM=PHYSICAL** eliminates the transpose, speeding up parallel I/O operation. This is useful, for example, if you are writing arrays temporarily to a file in order to gain working storage.

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

The **PHYSICAL** keyword is supported for sequential access files only. The order of array elements within a record is undefined. For more information about this keyword, see the *CM Fortran Language Reference Manual*.

5.5.2 CM Fortran Records

A CM Fortran file is a one-dimensional set of records.

There are two kinds of files: sequential access and direct access.

In a sequential access file, records may have different lengths, but they can only be accessed sequentially. In order to make these operations work on top of UNIX I/O, in which a file is just a one-dimensional set of bytes, the length of each record is placed at the beginning and end of each record.

In a direct access file, all records have the same length, which is provided in the **OPEN** statement. In direct access files, it is therefore unnecessary to include record length information.

In the current implementation individual record lengths are limited to 2 gigabytes in either sequential access (because the record length is a 4-byte integer) or direct access (because the record length is given in the **OPEN** statement as a 4-byte integer).

A Practical Example

The code

```
WRITE A,B      !where A and B are CM arrays
...
READ C         !C has length of A + length of B elements
```

writes arrays **A** and **B** to a file and then reads their data into array **C**, whose length is equal to the combined lengths of **A** and **B**.

Copyright © 1991–1994 Thinking Machines Corporation

The code

```
WRITE A
WRITE B
...
READ C
```

does not work the same way.

In the first case, **WRITE A,B** writes a single record; in the second case, two records are written, separated by eight bytes that encode the lengths of the two records.

In the latter case, to read the two records (**A** and **B**) into one array (**C**), use code similar to the following:

```
READ C(1:10)
READ C(11:60)
```

5.6 Hardware Platforms and Execution Models

Restrictions on the CM-5

Node-level programs on the CM-5 have restricted ability to perform CM Fortran I/O.

- From the nodal execution model, you cannot perform any parallel I/O, including using the utility library I/O procedures. For parallel I/O from nodal CM Fortran, use CMMD I/O routines. For serial I/O from a single node, use CM Fortran I/O (**READ** and **WRITE**).
- Global/local programs require that all I/O be done from a global program unit, using any of the CM Fortran I/O facilities.
- From the local part of a global/local program, you cannot perform any parallel I/O. See the current release notes for the status of serial I/O from local subroutines.

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

Restrictions on the CM Simulator

The CM simulator never performs parallel I/O and does not support the I/O utilities. Use **READ** and **WRITE** only when executing on the simulator.

5.7 Example

This example shows how to copy a file from a DataVault to an SDA.

```
      PROGRAM COPY_DATA
      IMPLICIT NONE
      INTEGER A(100)
CMF$  LAYOUT A(:NEWS)

      OPEN (9, FILE='dv6:/dir1/file1',
     $    ACCESS='SEQUENTIAL',
     $    FORM='UNFORMATTED',
     $    STATUS='OLD')

      OPEN (10, FILE='/sda/dir1/file1',
     $    ACCESS='SEQUENTIAL',
     $    FORM='UNFORMATTED',
     $    STATUS='NEW')

      READ (9, ERR=99) A
      WRITE (10, ERR=99) A
      CLOSE (9)
      CLOSE (10)
      STOP

99    PRINT *, 'error'
      END
```


Copyright © 1991–1994 Thinking Machines Corporation