Chapter 2

Hello Parallel World

This chapter examines some simple CM Fortran code to illustrate the operations that are fundamental to any array-processing program:

- declaring arrays
- initializing or otherwise moving data into arrays
- computing on arrays
- retrieving the results of computations
- compiling and executing a program

For simplicity, this chapter focuses on *elemental operations* on *whole arrays*.

- An elemental operation affects the elements of an array as if it had been applied separately to each element (in undefined order). Such an operation occurs within each CM processor independently of the others. Operations that specify data movement between processors are deferred to Chapter 8.
- A whole array is an array object specified simply by name, which indicates all the elements of the array. Such a reference has an implicit triplet subscript that corresponds to the declared bounds of the array. Chapter 5 shows operations that apply only to selected elements of an array.

Reminder: CM Fortran typically refers to the control processor and the parallel processors as *the front end* and *the CM,* respectively, regardless of which system components are performing these roles.

Copyright © 1989–1994 Thinking Machines Corporation

2.1 A Simple Program

The following program shows all the basic operations noted above. It declares and initializes two vectors, computes the sum of the squares of their corresponding elements, and prints out the result vector and the result vector's highest value. The remainder of this chapter steps through this program, pointing out the basic features of CM Fortran.

```
PROGRAM SIMPLE

IMPLICIT NONE
INTEGER A, B, C, N, MAXVALUE
PARAMETER (N=5)
DIMENSION A(N), B(N), C(N)

DATA A / 1,2,3,4,5 /
B = 2                       ! a CM array assignment

C = A**2 + B**2             ! array-valued expressions

PRINT *, 'Array C contains:'
PRINT *, C                  ! output of CM data

MAXVALUE = MAXVAL( C ) ! a CMF intrinsic function
PRINT *, 'The largest value in C is ', MAXVALUE

STOP
END
```

NOTE: This program is intended only to illustrate syntax and semantics, not parallel performance. Operations on very small arrays like these use only a fraction of the CM's processing resources. CM arrays should be at least machine size, and increasing array size further tends to increase performance.

Copyright © 1989–1994 Thinking Machines Corporation

2.2 Declaring and Initializing Arrays

CM Fortran supports all standard Fortran 77 syntax for declaring and initializing both scalar values and arrays.

Program **simple.fcm** uses Fortran 77 type specification statements, as well as the statements **DIMENSION**, **PARAMETER**, and **DATA**, to declare and initialize the scalar values **N** and **MAXVALUE** and the array **A**. Since CM Fortran supports standard Fortran I/O statements on the UNIX file system and on CM parallel storage devices, programs can also use the **READ** statement for initialization.

CM Fortran supports seven data types on all CM platforms, and an eighth, **INTEGER*8**, on CM-5 systems with vector units:

```
CHARACTER       REAL
LOGICAL         DOUBLE PRECISION
INTEGER         COMPLEX
INTEGER*8       DOUBLE COMPLEX
```

The **IMPLICIT** statement allows the programmer to override Fortran's implicit typing rules with specified typing rules, each of which associates one or more letters with a data type. The form used in program **simple**, **IMPLICIT NONE**, means that all identifiers must be declared. This form is useful for catching misspellings at compile type.

Arrays can be of any type, and from one to 31 dimensions. However, character arrays cannot be processed in parallel. That is, an array declared as type character is always stored in the memory of the control processor, and its elements are processed serially in the Fortran 77 manner. All other arrays can be stored and processed either on the control processor or on the parallel processors, depending on the use the program makes of them.

As an alternative to the Fortran 77 syntax shown, CM Fortran also offers the more economical Fortran 90 syntax. The Fortran 90 form can specify type, dimensionality, and initial values all in a single statement (see Chapter 4).

CM Fortran also provides dynamic array allocation, described in Chapter 7.

Copyright © 1989–1994 Thinking Machines Corporation

2.3 Array Operations

An array operation is any reference to an array object (that is, any reference using Fortran 90 syntax) in an expression, assignment, or intrinsic function call. Fortran 90 and CM Fortran extend the semantics of Fortran 77 such that operators and intrinsic functions can take array objects and operate on their elements. CM Fortran also adds a number of Fortran 90 intrinsic functions that manipulate or transform array objects.

The various forms of array operations are all illustrated in program **simple**:

```
B = 2                      ! a CM array assignment
C = A**2 + B**2            ! array-valued expressions
PRINT *, C                 ! output of CM data
MAXVALUE = MAXVAL( C )     ! a CMF intrinsic function
```

Notice the use of the Fortran 77 operators **+**, **=**, and ******. These features have been extended to operate on all the elements of an array object. Similarly, array objects can be passed as arguments to all the Fortran 77 intrinsic numeric and mathematical functions, such as **ABS**, **MAX**, and **SIN**. However, because character arrays are not supported on the CM, the intrinsic character functions, such as **CHAR** and **INDEX**, cannot take array objects as arguments.

The function **MAXVAL** is an example of the array-processing intrinsic functions that CM Fortran adds to Fortran 77. The array argument(s) to these functions are taken to be CM array objects.

2.3.1 Conformable Arrays

When an expression or assignment involves two or more arrays, the arrays must be *conformable,* that is, they must be of the same size and shape. It is obviously an error to assign, say, a 20-element vector to a 4x4 matrix. It is also an error to assign a 16-element vector to a 4x4 matrix because the language does not prescribe the storage order of distributed data.

The corresponding elements of conformable arrays reside in the memory of the same CM processor, which performs the computation on that set of elements. For this reason, *elemental* (element-wise) operations and whole-array assignments of conformable arrays are extremely efficient. Such operations

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

need not move data into the appropriate processors: it is already there. Each processor need only index within its own memory to locate the operands.

For example, consider the layout of two conformable vectors, `V(128)` and `R(128)`, on 16 processors.

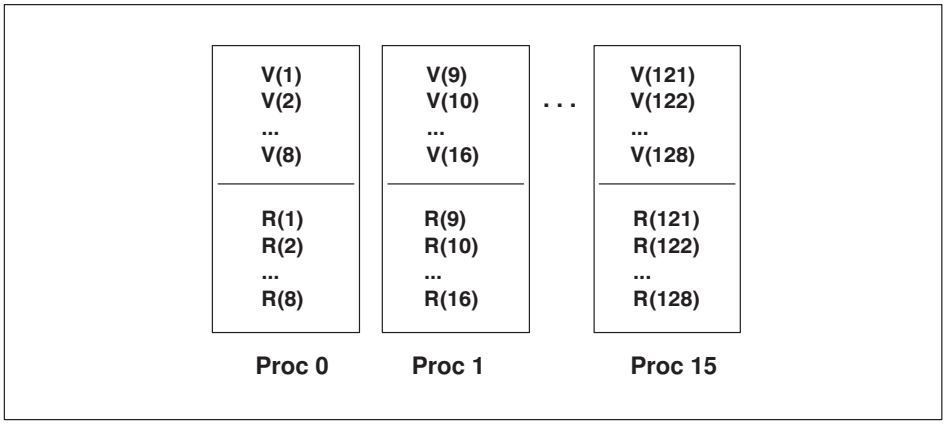

Figure 4. Two conformable vectors laid out on 16 processors.

In contrast, consider the layout of two nonconformable vectors, `V(128)` and `S(256)`. Their corresponding elements are not stored in the same processors.

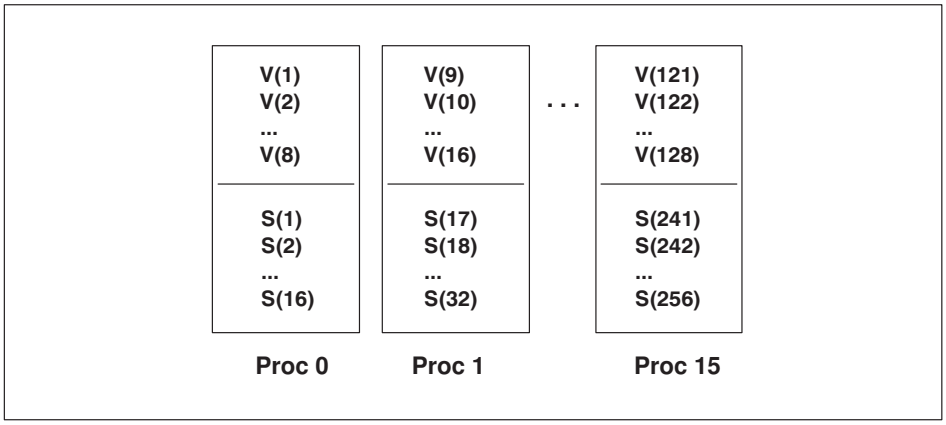

Figure 5. Two nonconformable vectors laid out on 16 processors.

Copyright © 1989–1994 Thinking Machines Corporation

CM Fortran provides syntax (shown in Chapter 5) to select a part of S that is conformable with **V** and which can therefore be used with it in an expression or assignment. Such an operation is legal but inefficient. For instance, imagine assigning the first half of **S** to **V**: Since the corresponding elements are not in the same processors, the system must send one of the operands to a temporary array to align the corresponding elements before the parallel processors can perform the assignment.

2.3.2 Scalar Extension

Scalars may be intermixed freely in expressions that have array-valued components. When a scalar appears in such an expression, it is treated as if it were an array conformable with the other array(s) in the expression. For example:

```
A = B/5

C = D * 3.14159
```

The first statement divides each element of **B** by the constant 5 and assigns it to the corresponding element of **A.** In the second statement, each element of **C** gets the circumference of a circle whose diameter is **D**.

Recall that scalars are stored in serial memory. When the CM system encounters a scalar in an array-valued expression, the control processor "broadcasts" the value to all the parallel processors. In effect, a scalar is conformable with any array and with different arrays in different expressions.

2.3.3 Array Homes

The system component on which an array is allocated is called its *home*. Aside from character arrays, which always reside on the control processor, an array's home is determined by how the array is used *within a program unit.* A program unit is a main program, a subroutine, or a function.

NOTE: Recall that CM Fortran uses *front end* to mean the control processor and *CM* to mean the system components that are serving as the parallel processors.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

- The front end stores all scalar data, including arrays that are referenced *only* as subscripted variables (in the Fortran 77 way) within a program unit. All serial operations, including looping operations on array elements, execute on the front end. Essentially, the front end executes all of CM Fortran that is Fortran 77.

  This is a front-end operation:

```
    INTEGER A(4)
    DATA A  / 1, 2, 3, 4 /

    DO 30 I=1,4
         A(I) = A(I) + 1     ! A is allocated on FE
30  CONTINUE
```

- The parallel processors (the CM) store all arrays that are referenced *at all* as array objects within a program unit. All operations on array objects execute on the CM. Essentially, the CM executes all of CM Fortran that is drawn from Fortran 90.

  This is a CM operation:

```
    INTEGER A(4)
    DATA A  / 1, 2, 3, 4 /

    A = A + 1                ! A is allocated on CM
```

- The CM stores all arrays that are referenced *both* as array objects and as subscripted variables within a program unit, although the serial operations *execute* on the front end.

  This is a (rather pointless) *mixed-home* operation:

```
    INTEGER A(4)
    A = 5                    ! A is allocated on CM

    DO 30 I=1,4
         A(I) = A(I) + 1     ! Loop executes on FE
30  CONTINUE
```

  These mixed-home operations tend to be inefficient, since the system moves the CM array one element at a time to the front end to perform the serial operation. If the algorithm demands a mixed-home operation, it is often advisable to use the CM Fortran utility routines

Copyright © 1989–1994 Thinking Machines Corporation

CMF_FE_ARRAY_TO_CM and **CMF_FE_ARRAY_FROM_CM**, which copy an array *en masse* from one system component to the other.

- Arrays placed in **COMMON** blocks reside on the CM unless otherwise specified with a compiler directive or switch. Discussion of common arrays is deferred until the chapter on subprograms, Chapter 6.

CM programmers need to be aware of where particular arrays are allocated so as to avoid using mixed-home operations unintentionally. Using array operations on CM arrays gives the program the performance benefits of parallelism, but using a front-end looping operation on a CM array can exact a high cost in performance. Not only does the system assign the elements one after another, but it also moves the array element by element from CM to front end and back again.

It is crucial to consider arrays' homes when calling subprograms, since CM Fortran requires that the home of an actual array argument be the same as the home of the dummy argument in the procedure. However, arrays' homes *are determined at compile time separately for each program unit*. Usually, the compiler goes by whether the array is used in an array operation in that procedure, although the programmer can control the decision with the compiler directive **LAYOUT**. Mismatched homes across procedure boundaries cause a run-time error.

For example:

```
PROGRAM FAILURE
INTEGER A(100,100)

DATA [initialize A]
CALL SUB(A)
PRINT *, A
STOP
END

SUBROUTINE SUB(B)
INTEGER B(100,100)
B = B**2
RETURN
END
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

The compiler allocates array **A** on the control processor, since there is nothing in the main program to indicate that **A** is to be processed in parallel. Array **B**, on the other hand, becomes a CM array by virtue of the array assignment in the subprogram. This program fails at run time when the front-end array is passed to the subroutine.

The programmer must therefore be aware of the homes of arrays used as arguments and must often take explicit steps to ensure that the homes of actual and dummy arguments match. See Chapter 6 for more information on arrays as arguments and on controlling their homes.

2.4 Retrieving CM Data

CM Fortran supports all Fortran I/O operations — the **READ**, **WRITE**, and **PRINT** statements — for both front-end and CM data. Using **READ** or **WRITE** for I/O between a CM array and a CM parallel storage device — the Scalable Disk Array or the DataVault — causes the data to be transferred in parallel, that is, in multiple streams.

There are various ways to retrieve and display the results of CM computations during program execution. Some intrinsic functions, such as **MAXVAL** or **SUM**, perform a combining operation on an array's elements and return the scalar result to the front end. The result of a *reduction* function such as this can, like any other scalar, be displayed with a **PRINT** statement. As shown in program **simple.fcm**:

```
INTEGER MAXVALUE
. . .
MAXVALUE = MAXVAL( C ) ! a CMF intrinsic function
PRINT *, 'The largest value in C is ', MAXVALUE
```

You can also retrieve a scalar value by subscripting a CM array in the Fortran 77 fashion to indicate the desired element. Notice that this is a deliberate use of the mixed-home construction: the array element that is referenced with a Fortran 77 subscript is automatically moved to the front end, where it is displayed by the **PRINT** statement:

```
PRINT *, 'The third element of array C is ', C(3)
```


Copyright © 1989–1994 Thinking Machines Corporation

You could also use the **PRINT** statement to view all the results stored in array **C**, since Fortran I/O statements are extended for use with CM data. As shown in program **simple.fcm**:

```
C = A**2 + B**2          ! array-valued expression
PRINT *, 'Array C contains:'
PRINT *, C               ! output of CM data
```

For large vectors or for matrices, a **FORMAT** statement might be used to improve the readability of the output:

```
DIMENSION D(4,4)
  . . .
PRINT 10, D
10  FORMAT (4I9)
```

Use the CM data visualization libraries, such as CM/AVS, for graphical display of CM array data.

2.5 Compiling and Executing

To compile and execute a CM Fortran program:

1. Place the program in a file with the filename extension **.fcm**.
2. Compile the file with the CM Fortran compiler command **cmf**.

   ```
   % cmf simple.fcm -o simple
   ```

3. Execute the program in the normal UNIX fashion on a CM-5 partition manager.

   ```
   % simple

   Array C contains:
            5         8         13        20        29
   The largest value in C is  29
    FORTRAN STOP
   ```

See the *CM Fortran User's Guide* for more information about compiling and executing CM Fortran programs.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

2.5.1 Specifying Execution Model

CM Fortran programs can be compiled for a number of execution models, which differ in the way they use the underlying CM hardware. The following chapter describes the execution models and indicates how to compile for each. A default execution model is established at installation time.

2.5.2 The CM-5 Compilation Process

When compiling for the CM-5, the compiler generates separate assembly-language files for serial and parallel code. The two code streams are assembled separately into serial and parallel object files and then rejoined into a single executable by the CM linker **cmld**.

For example, Figure 6 outlines the compilation process for a CM-5 system with vector units. The compiler command invokes the data parallel assembler **dpas**, the SPARC assembler **as**, and the CM linker **cmld** to process the intermediate files. Don't forget the second set of intermediate files if you move intermediate files to another directory or construct your own link libraries. See the *CM Fortran User's Guide* for more information.

2.5.3 Incorporating Existing Routines

Object modules generated by Sun Microsystems' FORTRAN compiler may be linked with modules produced by the CM Fortran compiler. This facility is useful for incorporating existing library routines into a CM Fortran application, as well as supporting the incremental conversion of an application from serial code to parallel array operations. Procedures compiled by foreign Fortran compilers will not, of course, make use of the CM processors.

If you include Sun FORTRAN object files on the **cmf** command line, be sure that the appropriate Sun libraries are available on your system.

Copyright © 1989–1994 Thinking Machines Corporation

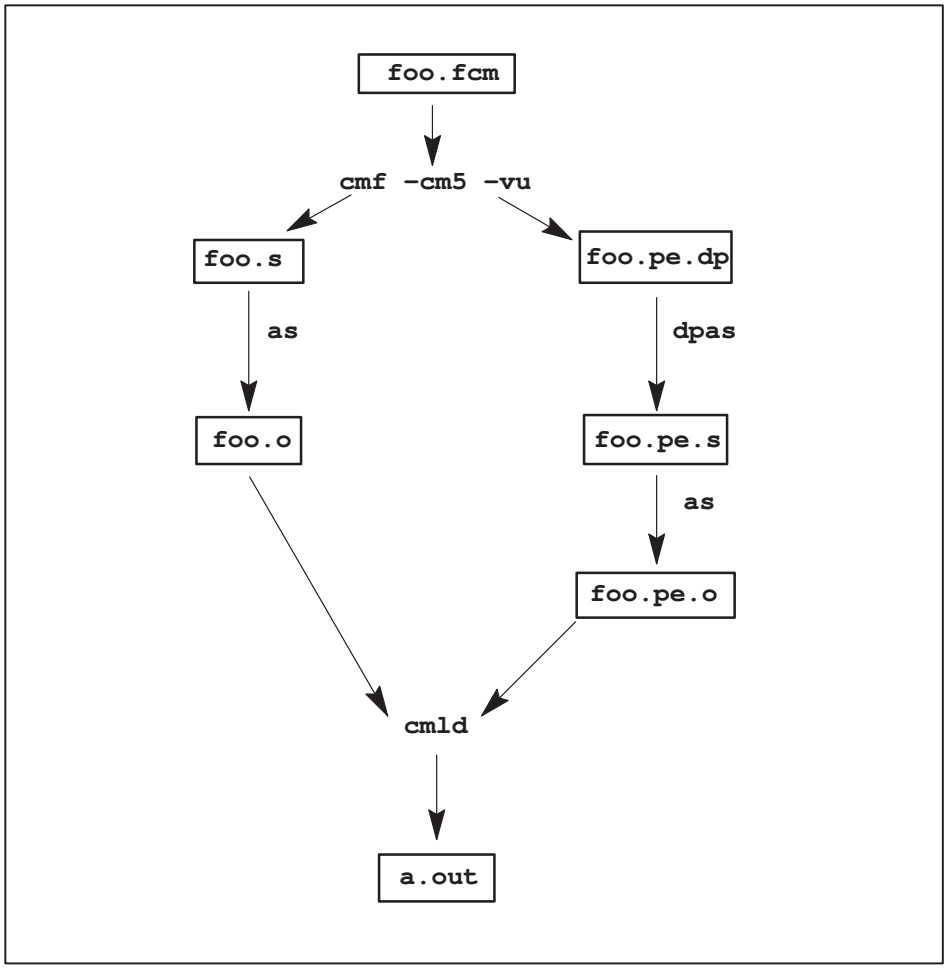

Figure 6. The CM Fortran compilation process for CM-5 with vector units.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation