Chapter 1

A Language for Array Processing

The CM Fortran language is an implementation of Fortran 77 supplemented with array-processing extensions from the ANSI and ISO standard Fortran 90. The array-processing features map naturally onto the data parallel architecture of the Connection Machine (CM) system, which is designed for computations on large data sets. CM Fortran thus combines

- the familiarity of Fortran 77, often the language of choice for scientific computing
- the expressive power of Fortran 90, which offers a rich selection of operations and intrinsic functions for manipulating arrays of data
- the computational power of the CM system, which brings multiple processors to bear on large arrays in either a data parallel fashion or a message-passing fashion

1.1 Array Operations in CM Fortran

The essence of the Fortran 90 array-processing features is that they treat arrays as first-class objects. An array object can be referenced by name in an expression or assignment or passed as an argument to a procedure, and the operation is performed on every element of the array.

Copyright © 1989–1994 Thinking Machines Corporation

1.1.1 Compared with Fortran 77

In Fortran 77, operations are defined only on individual scalars. Operating on an array requires stepping through its elements, explicitly performing the operation on each one. With Fortran 90 constructions, it is not necessary to reference array elements separately by means of subscripts, and it is not necessary to write **DO** loops or other such control constructs to have the operation repeated for each element. It is sufficient simply to name the array as an operand.

1.1.2 Example in Fortran 77 and CM Fortran

Consider a 4-element array **A**, initialized to **[1,2,3,4]**:

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

Suppose you want to increment each of the values by 1, so that **A** contains **[2,3,4,5]**. The familiar method in Fortran 77 is to reference the elements by subscript and, through a looping construct, explicitly increment each value:

```
      DO 30 I=1,4
           A(I) = A(I) + 1
30    CONTINUE
```

If the array is multidimensional, then the control sequence is nested to operate on all the elements:

```
      INTEGER A(4,4)

      DO 30 I=1,4
           DO 40 J=1,4
                A(I,J) = A(I,J) + 1
40         CONTINUE
30    CONTINUE
```

CM Fortran dispenses with the subscript references and the **DO** loops. Both the above operations are expressed simply as:

```
      A = A + 1
```


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

These code fragments perform the same set of operations, but their semantics are slightly different. The Fortran 77 statements are evaluated in the order specified by the nested loops, whereas the Fortran 90 construction allows the elements of **A** to be evaluated in any order. The CM system takes advantage of this feature to process the elements simultaneously.

A Fortran 90 array reference can be used for any size or shape array and for any array operation defined in CM Fortran. The array could, for example, be 4-dimensional, and the operation could be any Fortran operator or intrinsic function:

```
REAL B( 512, 64, 8, 4 )

B = 8.0         ! Set all 1,048,516 elements to 8.0.
B = B * 2.0     ! All 1,048,516 elements contain 16.0.
B = SQRT( B )   ! All 1,048,516 elements contain 4.0.
```

NOTE

The simple array reference **A** or **B** is the default form of a Fortran 90 *triplet subscript.* A triplet subscript, such as **A(1:4:1)**, contains the information that Fortran 77 expresses in the control specification of a **DO** loop: the first and last elements and the increment.

Fortran 90 thus replaces **DO** loops with a form of array reference that indicates all the elements of interest. See Chapter 5 for more information about Fortran 90 array references.

This manual uses the term *array object* or the CM Fortran term *CM array* to mean any array that is referenced in the Fortran 90 manner. That is, an array object is one for which the array reference contains an explicit or implicit triplet subscript that indicates all the elements that are to be operated upon.

Copyright © 1989–1994 Thinking Machines Corporation

1.2 Data Parallel Processing

Fortran 90 array operations map naturally onto the data parallel CM system. From the software perspective, an array object refers to all the data elements of the array simultaneously. From the hardware perspective, operations on the array's elements are performed simultaneously.

1.2.1 Compared with Serial Processing

A serial implementation of Fortran 90 would have the syntactical convenience of referencing arrays as objects, but the compiler would necessarily generate serial loops to process the elements sequentially. However, if the operations on individual data elements are independent of one another — that is, there is no loop-carried data dependence — there is no inherent need for them to be sequential. The sequence of operations is an artifact of the processor, not of the program.

A data parallel computer — one that can store the data elements in the memories of several processors — is free to operate on more than one element at a time. For example, given a 10 x 10 x 10 array **A**, consider the expression

```
SQRT(A)/2
```

To evaluate this expression, a serial computer would need to perform 2000 arithmetic computations one after another. A 1024-processor CM, in contrast, could process all the elements of **A** at once, with each processor performing only two computations.

1.2.2 The Compiler's Machine Model

The CM Fortran compiler assumes that its target machine is a collection of parallel processors, each with its own memory, all acting under the direction of a serial control processor.

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

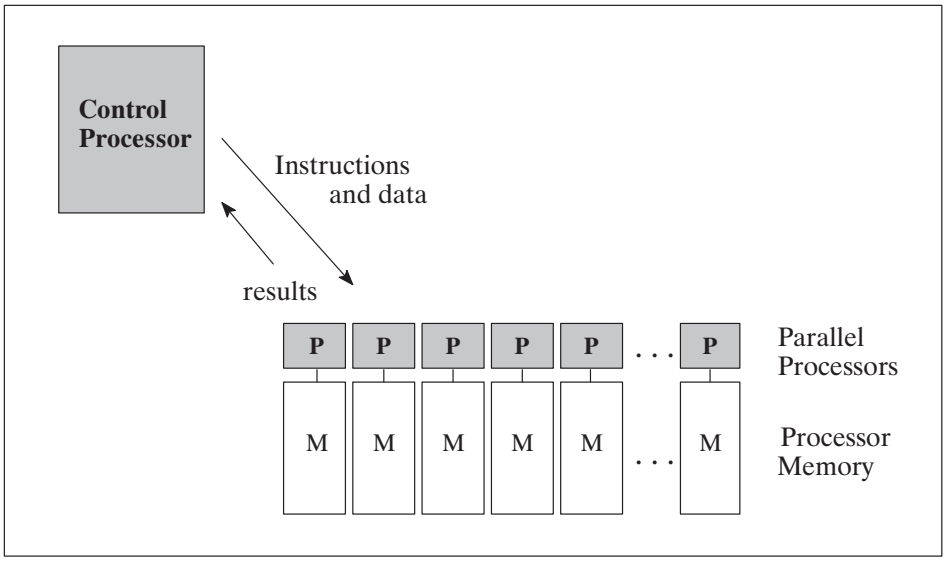

Figure 1. The CM Fortran compiler's machine model.

The compiler decomposes the input program into serial and parallel components:

- Arrays used in Fortran 90 constructions are distributed across the memory of the parallel processors. For historical reasons, these Fortran 90 array objects are called *CM arrays* in CM Fortran; some of the CM libraries also refer to them as *parallel arrays.*
- Scalar data and arrays used only in Fortran 77 loop constructions are stored in the memory of the control processor. These arrays are called *front-end arrays* in CM Fortran and *serial arrays* in some of the CM libraries.
- Arrays that are used in both ways are distributed across the parallel processors and moved to the control processor for serial operations.

The compiler generates serial instructions for the control processor to execute on scalar data stored in its own memory. It also generates blocks of parallel *computations*, meaning element-wise operations and certain other operations that involve no interprocessor communication.

Copyright © 1989–1994 Thinking Machines Corporation

The control processor sends these parallel code blocks to the parallel processors, and each executes them on its own portion of the data. The control processor also executes control-flow statements and calls run-time library routines whenever the parallel processors need to communicate with each other or with the control processor.

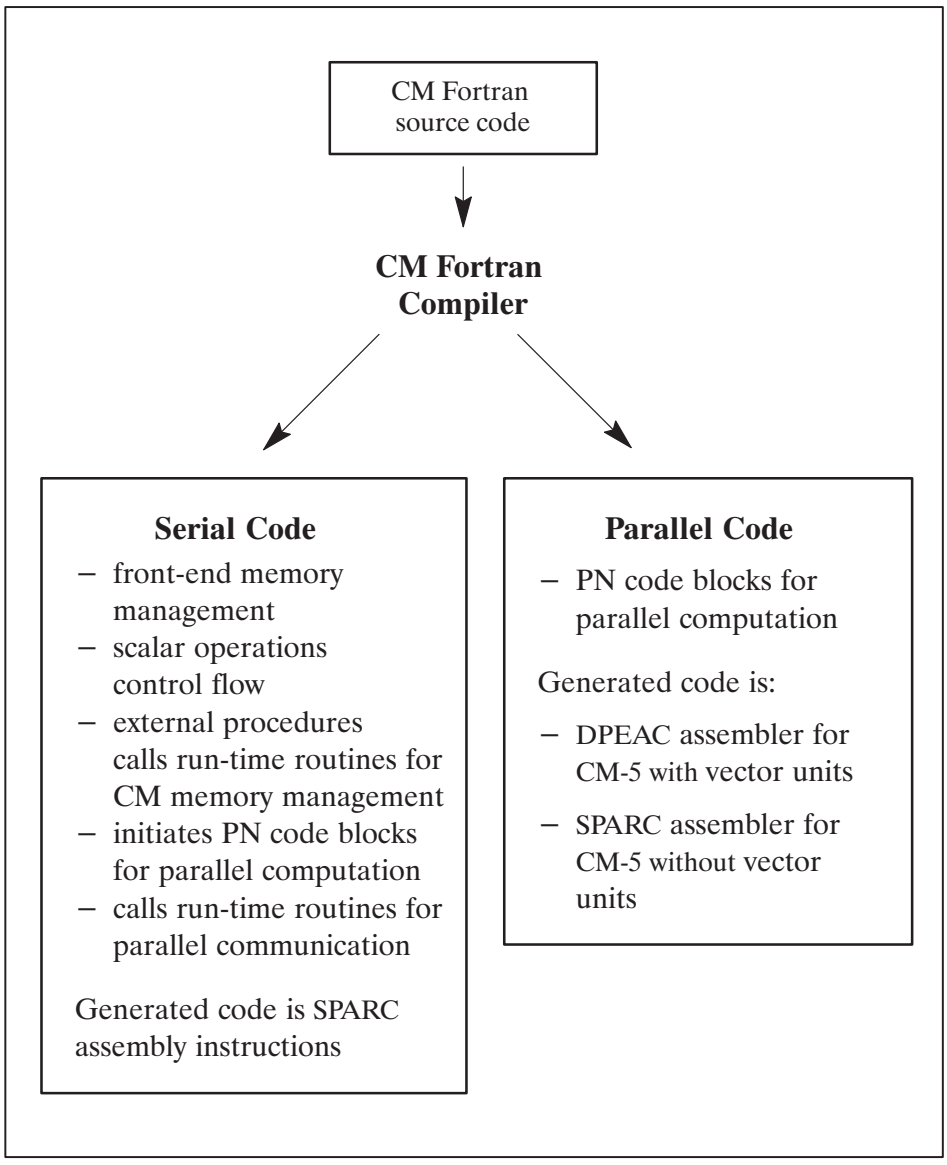

Figure 2. The compiler's "two machines" output.

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

Notice that no new data structure is needed to express parallelism in Fortran, and the programmer need not specify where arrays are to be allocated (although compiler directives and switches do provide this capability).

Nor does the programmer need to specify when or how processors are to communicate, since the compiler generates several kinds of communications instructions as needed:

- Nearest-neighbor, or *NEWS*, communication, whereby each processor gets a value from its neighbor on an *n*-dimensional grid, all at the same time
- General-purpose, or "send," communication, whereby each processor sends a value to any arbitrary processor, all at the same time
- Global communication, which includes cumulative computations along grid axes and reduction of an array to a single value

Because there is only one instruction stream for the parallel processors, they are naturally synchronized. Race conditions cannot develop because no processor proceeds to the next code block or communication instruction until all processors have finished the current one.

1.2.3 Data Parallel Processing on the CM System

The compiler's "two-machine" model reflects the data parallel architecture of CM systems. These systems provide a control processor, called the *partition manager*. The CM also provides a set of parallel processors, or *processing nodes.*

See the *Connection Machine CM-5 Technical Summary* for detailed information on system architecture.

Copyright © 1989–1994 Thinking Machines Corporation

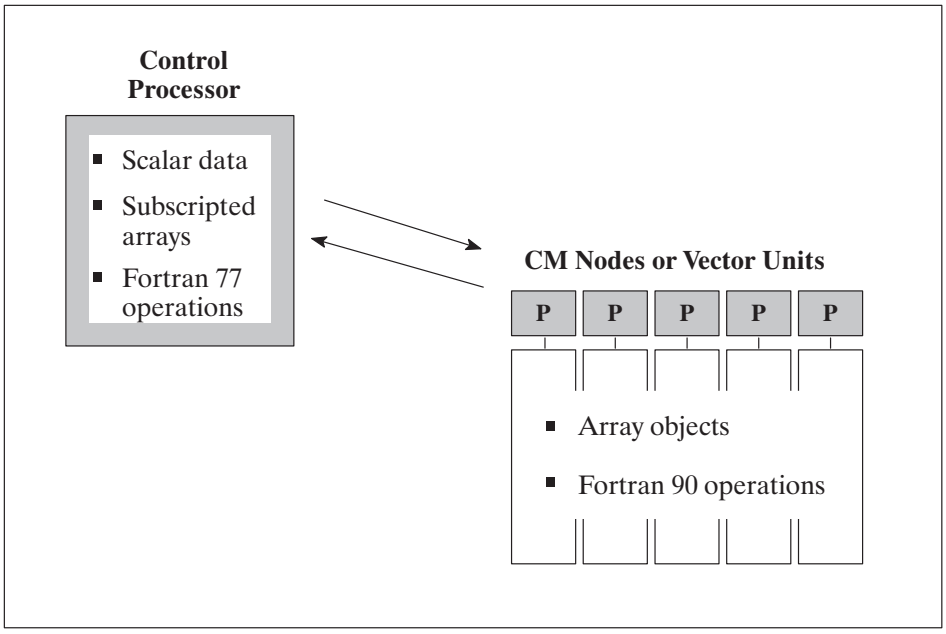

Figure 3. Division of labor between CM system components.

In the straightforward *global* model of program execution, the partition manager executes the code that the compiler generates for the control processor, and the nodes store and operate upon CM array objects. In a CM-5 with vector units, the vector units serve as the parallel processors instead.

On the CM-5, the three-level hierarchy of processors — partition manager, node microprocessors, and vector units — permits other mappings to hardware, including asynchronous message-passing programs that run on the individual nodes, each using the associated four vector units as its set of parallel processors.

Chapter 3 of this manual describes the CM Fortran execution models.

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

NOTE ON TERMINOLOGY

CM Fortran typically refers to the control processor and the parallel processors as *the front end* and *the CM*, respectively. Hence the terms *front-end array* and *CM array.*

On CM-5, and particularly in the message-passing execution models, it is important to recognize that *front end* means whatever system component is serving as the control processor (it may be the partition manager or a node), and that *CM* refers to whatever set of processors is executing CM Fortran array operations.

1.2.4 CM Fortran as a Superset of Fortran 77

CM Fortran is a superset of Fortran 77. The differences between the two languages reflect the compiler's "two-machine model," that is, the assumption that a CM Fortran program is directing two system components with different memory organizations.

Any Fortran 77 construction can be used with scalar data and front-end ("serial") arrays, since such data is stored and processed in the conventional serial manner.

Most Fortran 77 features are extended for use with array objects, as the `+` and `=` operators are used above in the array operation `A = A + 1`. However, certain features with storage-order dependences — most notably, the `EQUIVALENCE` statement — are not supported for use with CM arrays.

CM Fortran implements Fortran 90's array-processing features for use only with CM arrays. It also offers a subset of Fortran 90's precision-control syntax and certain other Fortran 90 features that are commonly found as extensions of Fortran 77 (see Section 1.3). CM Fortran does not implement Fortran 90's derived types ("structures"), modules, or pointer assignments.

Copyright © 1989–1994 Thinking Machines Corporation

1.3 The Features of CM Fortran

This section summarizes the major extensions of CM Fortran over Fortran 77. See the *CM Fortran Language Reference Manual* and the *CM Fortran Libraries Reference Manual* for complete information.

1.3.1 Fortran 90 Array Processing

The array-processing features that CM Fortran draws from Fortran 90 include:

- *expanded semantics* for Fortran 77 operators and intrinsic functions, such that they can take an array object and operate on its elements
- *array sections* and *vector-valued subscripts,* new syntax for selecting subarrays from array objects
- *the* **WHERE** *statement* and construct*,* which operate conditionally on an array's elements depending on the elements' values
- *new intrinsic functions* for permuting and transforming arrays, as well as for constructing arrays and inquiring about their properties
- *dynamic memory management* for **ALLOCATABLE** arrays and array pointers by means of the statements **ALLOCATE** and **DEALLOCATE**
- *attributed type declarations*, an alternative to Fortran 77 type declarations and the **DIMENSION** statement for declaring arrays

1.3.2 Other Array Extensions

CM Fortran includes some extensions over Fortran 90 that are particularly useful for data parallel programming. Several of these features have been adopted by the emerging industry standard High Performance Fortran.

- *The* **FORALL** *statement*, a powerful facility for initializing arrays, for selecting subarrays, and for specifying data movement in terms of array indices.

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

- *The intrinsic functions* **FIRSTLOC**, **LASTLOC**, and **PROJECT**, which return the locations of certain array elements (such as the first true element); the array transformation functions **DIAGONAL** and **REPLICATE**; and the inquiry function **RANK**.

- *Compiler directives* **LAYOUT** and **ALIGN**, which control the layout of arrays in distributed memory. The choice of array layout can have major effects on program performance.

1.3.3 Precision Control

CM Fortran provides Fortran 90 syntax for specifying the *kind* of numeric data types (for both scalars and arrays). A type's kind indicates either 32-bit or 64-bit precision.

- **KIND** *keyword* and predefined *kind type parameters* for declaring the precision of numeric variables, plus the **KIND** intrinsic function for inquiring about precision

- syntax for specifying the *kind type of literal constants*

1.3.4 Other Fortran 90 Features

CM Fortran also includes some Fortran 90 features that are not specifically related to array processing, but are commonly found in implementations of Fortran 77. Examples of these are:

- control-flow statements **CASE**, **DO TIMES**, **DO WHILE**, and **END DO**
- **NAMELIST** I/O
- **DOUBLE COMPLEX** data type
- **INCLUDE** lines
- **IMPLICIT NONE** statement

Copyright © 1989–1994 Thinking Machines Corporation

1.3.5 CM Fortran Libraries

CM Fortran provides three libraries: the utility library, the cmf77 library, and the global/local library.

The *utility library* provides:

- some language-level operations that are not available in CM Fortran, such as generating random numbers in an array and ranking or sorting array elements by value
- some performance-enhancing capabilities beyond what the language delivers, such as fast data transfers between the CM's serial and parallel processors

In earlier versions, the utility library served mainly as a stopgap for language features that were inefficient or were not yet implemented. In this version, many of the utility procedures are redundant, or nearly so, with language features. The *CM Fortran Libraries Reference Manual* compares each set of procedures with the corresponding language features, if any, and points out any significant differences in behavior or performance.

The library *libcmf*77 is similar to Sun Microsystem's library *libF77.* It provides interfaces to OS system calls, such as **`TIME`** and **`IARGC`**. Two particularly useful routines are **`FMALLOC`** and **`FFREE`**, which permit dynamic allocation of front-end arrays, a feature not otherwise available in CM Fortran. (This operation is illustrated below in Chapter 7).

A third CM Fortran library is the global/local library CMGL. These procedures are used only in the global/local model of programming, described below in Chapter 13.

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

1.3.6 CM System Libraries

Besides the CM Fortran libraries described above, CM systems offer the following libraries callable from CM Fortran:

- CM Scientific Software library, CMSSL: highly optimized routines for performing common scientific and mathematical tasks
- CM Message Passing library, CMMD: routines for handling interprocessor communication, synchronization, and I/O from within CM Fortran programs running locally on the CM-5 nodes
- CM data visualization libraries, such as CM/AVS
- CM low-level I/O libraries: SFS for parallel I/O between the CM-5 and the Scalable Disk Array, and CMFS for parallel I/O between any CM and the DataVault or devices such as CM-HIPPI

See the documentation for the individual libraries for information on calling them from CM Fortran programs.

Copyright © 1989–1994 Thinking Machines Corporation