Chapter 1

What Is CM Fortran?

The CM Fortran language is an implementation of Fortran 77 supplemented with array-processing extensions from the ANSI and ISO standard Fortran 90. These 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
- The computational power of the CM system, which brings tens, hundreds or thousands of processors to bear on large arrays, processing array elements in unison

This manual introduces CM Fortran as implemented for all Connection Machine models: CM-2, CM-200, and CM-5. The language itself is completely portable among Connection Machine models, although some features of the underlying system architecture and operating system differ from one model to another.

1.1 Array Processing 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 any Fortran intrinsic function, and the operation is performed on every element of the array.

```
REAL A(40,40,40)

A = 8.0              ! Set all 64,000 elements to 8.0.
A = A * 2.0          ! All 64,000 elements contain 16.0.
A = SQRT( A )        ! All 64,000 elements contain 4.0.
```

Fortran 90 Array References

When an array is referenced in the Fortran 77 manner (with subscripts), its elements are treated as individual scalars. To operate on such an array, you need to step through its elements in a loop, nested as deeply as the number of dimensions. Fortran 90 array operations dispense with the individual subscript references and the **DO** loops.

Fortran 77 style	Fortran 90 style
`INTEGER B(100)` / `...` / `DO 30 I=1,100` / `B(I) = B(I)+1` / `30 CONTINUE`	`B = B + 1`
`INTEGER C(100,100)` / `...` / `DO 30 I=1,100` / `DO 40 J=1,100` / `C(I,J)=C(I,J)+1` / `40 CONTINUE` / `30 CONTINUE`	`C = C + 1`

Fortran 77 style

```
   INTEGER B(100)
   ...
   DO 30 I=1,100
     B(I) = B(I)+1
30 CONTINUE
```

Fortran 90 style

```
B = B + 1
```

Fortran 77 style

```
   INTEGER C(100,100)
   ...
   DO 30 I=1,100
     DO 40 J=1,100
       C(I,J)=C(I,J)+1
40   CONTINUE
30 CONTINUE
```

Fortran 90 style

```
C = C + 1
```

The results are the same, but the semantics are slightly different. The Fortran 77 constructs are evaluated in the order specified by the nested loops, whereas Fortran 90 allows the elements of **B** and **C** to be processed in any order, including simultaneously. The CM system takes advantage of this standard feature to process the array elements in parallel.

Data Parallel 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. The CM system stores the array elements in the memories of separate processors and operates on multiple elements at once.

For example, given a 40 x 40 x 40 array **`A`**, consider the statement:

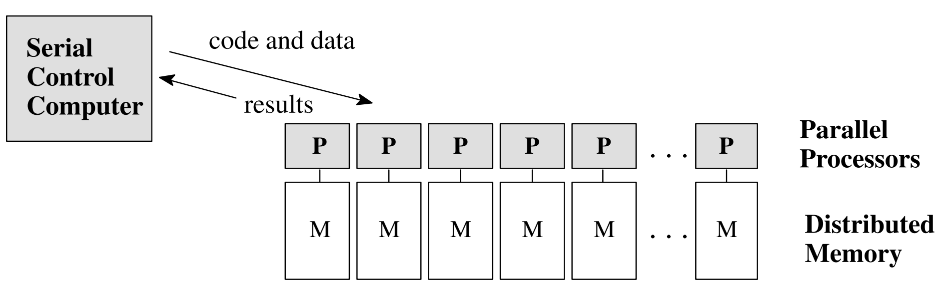
```
A = SQRT(A)/2
```

To execute this statement, a serial computer would need to perform 128,000 arithmetic computations. The CM system, in contrast, can provide a processor for each of the 64,000 data elements, and each processor needs to perform only two computations.

Because there is only one instruction stream, the CM processors are naturally synchronized. Race conditions cannot develop because no processor proceeds to the next instruction until all have finished the current instruction.

1.2 The Connection Machine System

The CM program execution environment consists of a set of processors, each with its own memory, all acting under the direction of a serial control processor. (In the CM-5, the control processor is called the *partition manager;* in the CM-2 and CM-200, it is called the *front end.)* To accommodate a data set that is larger than the set of physical processors, the processors subdivide their memory among multiple data elements and process each in turn. As a programming model, you can imagine the parallel processing unit as a set of *virtual processors*, one for each data element.



When the CM system executes a data parallel program, the serial processor directs the flow of control and also operates on scalar data stored in its own memory. For parallel operations, the serial processor directs all the parallel processors (or some selected subset of them) to execute instructions on user data stored in their own memories.

Few useful problems decompose into completely independent subproblems, of course; most require the parallel processors to interact. Fortran 90 defines numerous intrinsic functions and other features that transform arrays or move their elements. These features map naturally onto the CM's mechanisms for interprocessor communication.

1.3 CM Fortran on the CM System

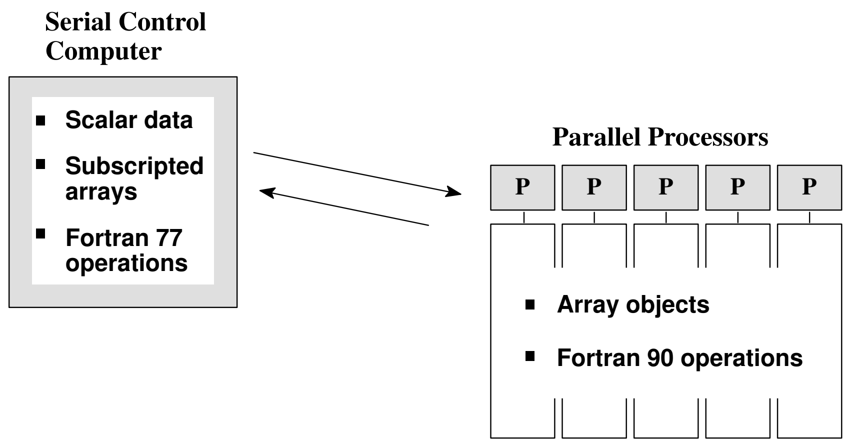
CM Fortran is a superset of Fortran 77. The differences between the two languages reflect a basic fact of CM architecture: a CM Fortran program is directing two CM system components with different memory organizations. An array can be stored — that is, its *home* can be — either in the memory of the serial processor or in the distributed memory of the parallel processors.

Memory Management

No new data structure is needed to express parallelism, and the programmer need not take any special action to invoke the parallel processing unit. Within each program unit — main program, subroutine, or function — the CM Fortran compiler allocates arrays on one system component or the other *depending on how they are used.*

- Arrays that are used *only* in Fortran 77 constructions in a program unit, and all scalar data, reside on the serial control processor (called the *front end* in CM Fortran). Essentially, the front end executes all of CM Fortran that is Fortran 77.

- Arrays that are used in array operations anywhere in a program unit reside on the parallel processing unit (called the *CM* in CM Fortran). Essentially, the CM executes all of CM Fortran that is Fortran 90.



CM Fortran programmers usually avoid using an array *both* as an array object and as a subscripted array. Such an array has a CM home, but the system moves it to the front end, one element at a time, to perform the serial operation. Naturally, this transfer exacts a performance cost.

CM Array Operations

CM Fortran supports all Fortran 77 features for operating on scalar data and subscripted arrays. For operating on CM arrays (array objects), CM Fortran includes these Fortran 90 array-processing features:

- *Elemental operations.* Most Fortran 77 features — operators, statements, and intrinsic functions — can be applied to CM arrays. These operations are called *elemental* because they behave as if they had been applied separately to each element of the array (in undefined order).

  Because of the CM's distributed memory, certain Fortran 77 features with storage-order dependencies — most notably, the **`EQUIVALENCE`** statement — cannot be used with CM arrays.

- *Conditionals and subarrays.* Fortran 90 syntax and statements can select subarrays for array operations. The elements of interest can be selected either by their values or by their positions in the array.

- *Array intrinsic functions.* The Fortran 90 intrinsic functions inquire about arrays' properties, shift or transpose elements, determine the location of certain elements (such as the maximum value), perform vector and matrix multiplication, reduce arrays to scalars, or construct new arrays from the information in argument arrays.

CM Fortran also includes some features that are specific to the CM system. These features are particularly useful for data parallel programming:

- *CM Fortran extensions.* CM Fortran offers the **FORALL** statement, which performs an elemental array assignment where the particular values assigned are location-dependent, and several intrinsic functions beyond those defined in Fortran 90.

- *Compiler directives.* Several compiler directives control the layout of arrays in CM memory, which can have major effects on program performance. Another directive controls whether arrays in common blocks are stored on the CM or the front end.

- *Utility Library routines.* These procedures provide a convenient interface to the capabilities of lower-level CM software. They serve a number of purposes, such as:

  - Providing Fortran 90 capabilities that are not yet implemented in CM Fortran, such as generating random numbers

  - Performing actions that are specific to the CM system, such as moving an array *en masse* between front-end memory and CM memory

  - Improving performance in cases where the CM Fortran compiler does not yet translate language syntax into the optimal parallel instruction