Chapter 9

The CM Fortran Programming Language

Fortran for the Connection Machine system is standard Fortran 77 supplemented with the array-processing extensions of the ANSI and ISO (draft) standard Fortran 90. These extensions provide convenient syntax and numerous intrinsic functions for manipulating arrays.

Newly written Fortran programs can use the array extensions to express efficient data parallel algorithms for the CM. These programs will also run on any other system, serial or parallel, that implements Fortran 90. CM Fortran also offers several extensions beyond Fortran 90, such as the **FORALL** statement and some additional intrinsic functions. These features are well known in the Fortran community and are particularly useful in data parallel programming.

9.1 Structuring Parallel Data

Fortran 90 allows an array to be treated either as a set of scalars or as a first-class object. As a set of scalars, array elements must be referenced explicitly in a **DO** construct. In contrast, a reference to an array object is an implicit reference to all its elements (in unspecified order). For example, to increment the elements of the 100-element array **A** by 1, a program can reference the array either way:

A as a set of scalars

```
DO I=1,100
  A(I) = A(I) + 1
END DO
```

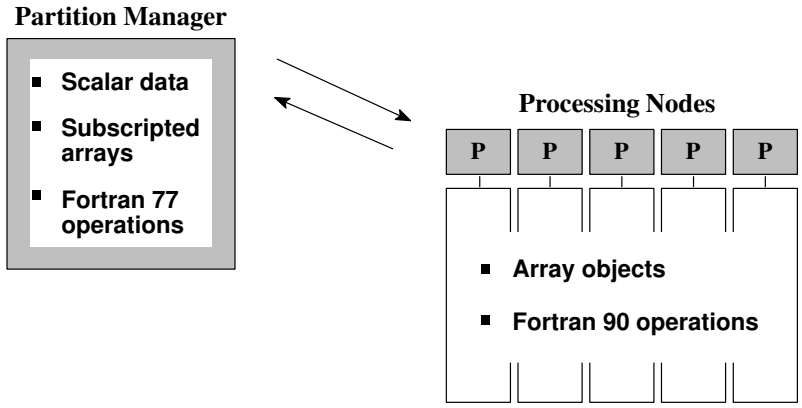
A as an object

```
A = A + 1
```


Copyright © 1993 Thinking Machines Corporation

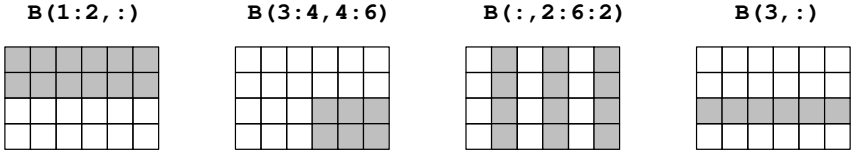
To operate on multidimensional arrays, **DO** loops must be nested to reference each element explicitly. In the statement **A = A + 1**, however, **A** could be a scalar, a vector, a matrix, or a higher-dimensional array.

CM Fortran takes advantage of this standard feature when allocating arrays on the CM system. An array that is used only as a set of scalars is stored and processed on the partition manager in the normal serial manner. Any array that is referenced as an object is stored in node memory, one element per processor, and processed in parallel. In essence, the partition manager executes all of CM Fortran that is Fortran 77, and the nodes execute all the array extensions drawn from Fortran 90. No new data structure is required to express parallelism.



The simple array reference **A** may be written more explicitly using a *triplet subscript*, **A(1:100:1)**, which resembles the control specification of a **DO** loop. Using triplet subscripts, you can replace one or more **DO** loops with an array reference that indicates all the elements of interest — and thereby cause the array to be processed in parallel.

An implicit triplet — that is, the array name alone — is usually used for whole arrays. You can, however, explicitly specify any of the index variables, just as in a **DO** loop, to indicate a *section* of the array. For example, some sections of array **B(4,6)** are:



November 1993
Copyright © 1993 Thinking Machines Corporation

Array sections can be used anywhere that whole arrays are used — in expressions and assignments and as arguments to procedures.

9.2 Computing in Parallel

The most straightforward form of data parallel computing is *elemental* computing, that is, operating on array elements all at the same time, each independently of the others. An assignment statement where the entire array is referenced as an object has this effect. For example, consider the following assignment statement for an 8 x 8 x 8 array **C**:

```
C = C**2
```

The CM system allocates one element of **C** in each of 512 processors, and all the processors operate on their respective elements of **C** at the same time.

An expression or assignment can involve any number of arrays or array sections, as long as they are all of the same shape. Scalars can be intermixed freely in array operations, since Fortran 90 specifies that a scalar is effectively replicated to match any array. For example, the following statement assumes that **D** and **E** are 10 x 10 matrices and **F** is a 10 x 100 x 100 array:

```
D = E*2.0 + 1.0 + F(:,1:10,3)
```

Another form of array operation uses an *elemental* intrinsic function. Fortran 90 extends most of the intrinsic functions of Fortran 77 so that they can take either a scalar or an array as an argument. If **G** is an array, this statement operates elementally:

```
G = SIN(G)
```

An array assignment can be performed conditionally if it is constrained by a **WHERE** statement. This statement includes a logical mask; it behaves like a **DO** loop with an embedded **IF** statement (except that the order in which elements are processed is unspecified). For example, to avoid division by zero in an array assignment, one might say:

```
WHERE (D.NE.0)  E = E/D
```


Copyright © 1993 Thinking Machines Corporation

Finally, CM Fortran offers a form of elemental array assignment, the **FORALL** statement, whose action is position-dependent. The syntax of a **FORALL** statement resembles a **DO** construct, but the assignments can be executed in parallel. For example, to initialize **H** as a Hilbert matrix of size **N**:

```
FORALL (I=1:N, J=1:N)  H(I,J) = 1.0 / REAL( I+J-1 )
```

FORALL can use a mask to make its action dependent on either the value or the position of the individual array elements. For example, to clear matrix **H** below the diagonal, one can set a mask to select those positions where row index **I** is greater than column index **J**:

```
FORALL (I=1:N, J=1:N, I.GT.J )  H(I,J) = 0.0
```

To initialize a table of integer logarithms:

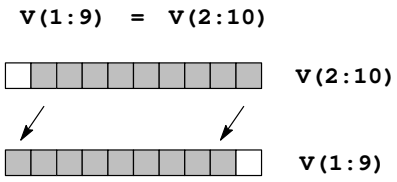
```
FORALL (I = 1:10)  LG (2**(I - 1): 2**I - 1) = I - 1
```

9.3 Communicating in Parallel

A second form of data parallel computing requires processors to access each other's memories, all at the same time. The pattern of interprocessor communication can be either regular (grid-based) or arbitrary. Fortran 90 defines a number of features that move data from one array position to another; these features map naturally onto the communication mechanisms implemented in CM hardware.

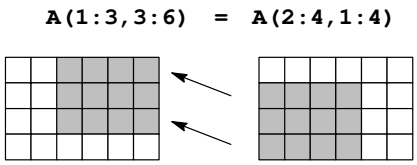
Grid-Based Communication

Many applications, such as convolutions and image rotation, need to move data in regular grid patterns. One way to specify such motion in Fortran 90 is by assigning array sections. For example, to shift vector values to the left:



November 1993
Copyright © 1993 Thinking Machines Corporation

To shift data on more than one dimension:



Fortran 90 also defines intrinsic functions that perform grid-based data motion. The function **CSHIFT** performs a circular shift of array elements, and **EOSHIFT** performs an end-off shift. For example, the following statement shifts the elements on the second dimension of **A** by one position to the left and assigns the result to **B**. (The **SHIFT** argument can also be an array, which shifts the rows by different offsets.)

```
B = CSHIFT( A, DIM=2, SHIFT=1 )
```

One notable use of **CSHIFT** is in so-called "stencils," array expressions that compute a weighted sum of neighboring points of a specific grid point. A simple example would be

```
A = C3*B + C1*CSHIFT (B,DIM=1,SHIFT=-1) + C2*CSHIFT (B,DIM=2,SHIFT=-1)
```

The CM Fortran compiler includes optimizations that provide particularly high performance for stencils.

General Communication

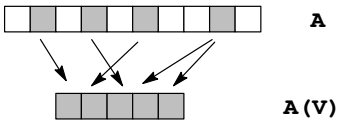
Processors must communicate in arbitrary patterns to map an unstructured problem onto a grid or to index into arbitrary locations of an array. To perform these operations in parallel, CM Fortran provides vector-valued subscripts and **FORALL.**

A vector-valued subscript is a form of array section that uses a vector of index values as a subscript. If **A** is a vector of length 10 and **P** is an array containing a permutation of the integers from 1 to 10, then **A = A(P)** applies this permutation to the values in **A**. The statement **A(P) = A** applies the inverse permutation.

The index values can be repeated, which causes element values to be repeated in the section. For example, if **V** is the vector **(/2,6,4,9,9/)**, then **A(V)** is a

Copyright © 1993 Thinking Machines Corporation

five-element vector whose values are **A(2)**, **A(6)**, **A(4)**, **A(9)**, and **A(9)**, in that order:



The **FORALL** statement provides the same arbitrary indexing into an array of any rank. For example, the following statement uses the two-dimensional index arrays **X** and **Y** to permute the values of a two-dimensional array **B**:

```
FORALL (I=1:N, J=1:M)  C(I,J) = B( X(I,J), Y(I,J) )
```

9.4 Transforming Parallel Data

Fortran 90 defines a rich set of intrinsic functions that take an array argument and construct a new array (or scalar). All these transformational functions take only array objects (not arrays subscripted in the Fortran 77 manner), and all are therefore computed in parallel on the CM.

One set of transformational functions is the reduction intrinsics, such as **SUM** or **MAXVAL**. These functions apply a combining operator to the elements of an array (or array section) and return the result as a scalar. For example, given a 100 x 500 matrix **D**, the following expression returns the sum of the elements in the upper left quadrant:

```
SUM( D(1:50,1:250) )
```

These functions can take a mask argument to make the reduction conditional. If applied only to a specified dimension, they return an array of rank one less than the argument array. For example, given the 100 x 500 matrix **D**, the following expression returns a 100-element vector containing the sums of the positive elements in each row.

```
SUM( D, DIM=2, MASK=D.GT.0 )
```

A parallel prefix, or *scan,* operation applies a combining operator cumulatively along a grid dimension, giving each element the combination of itself and all previous elements. These operations, which are useful in such algorithms as line-of-sight and convex-hull, can be expressed with the **FORALL** statement and

November 1993
Copyright © 1993 Thinking Machines Corporation

a reduction function. For example, in the following add-scan (or sum-prefix) operation, each element of **B** gets the sum of all elements up to and including the corresponding element of **A**:

```
FORALL (I=1:N)  B(I) = SUM( A(1:I) )
```

The array construction functions transform arrays in a wide variety of ways. For example, **TRANSPOSE** performs matrix transposition; **RESHAPE** constructs a new array with the same elements as the argument but a different shape; **PACK** and **UNPACK** behave as gather/scatter operations; and **SPREAD** replicates an array along a new dimension. CM Fortran also provides the Fortran 90 array multiplication functions, **DOTPRODUCT** and **MATMUL** In addition to the standard Fortran 90 intrinsics, CM Fortran also offers the functions **DIAGONAL**, **REPLICATE**, **RANK**, **PROJECT**, **FIRSTLOC**, and **LASTLOC**.

Copyright © 1993 Thinking Machines Corporation