Appendix A

Compiler Directives

This appendix explains the form and use of compiler directives. A *directive* (R19-1) specifies information to the CM Fortran compiler regarding the intended uses or desired memory layout of arrays or array-valued functions declared or referenced in a program unit.

A *directive* is one of: R19-1
- *layout-directive*
- *align-directive*
- *common-directive*

The *layout-directive* and *align-directive* provide some control over the allocation of arrays in distributed memory, and the proper use of these directives may significantly improve the performance of a CM Fortran program. These directives also may be used to specify information regarding array-valued functions declared or referenced in a program unit. The *common-directive* specifies how common blocks are to be used in a program.

A.1 Form and Placement of Directives

A directive is a comment line with the general form

`CMF$` *directive-name argument(s)*

where *directive-name* is one of `LAYOUT`, `ALIGN`, or `COMMON` and *argument(s)* is a list of arguments (separated by commas) appropriate to the particular directive. Column placement is not important except that the initial `C` must be in column one. The use of the character `$` following the `CMF` is optional, but if the `$` is omitted, one or more spaces must separate the `CMF` from the directive name. (For brevity, the fact that the `$` is optional is not reflected in the syntax rules below.)

Copyright © 1989–1994 Thinking Machines Corporation

NOTE: Blanks *are* significant in directives, unlike in other constructions of the language.

Directives may be continued over several lines. If a line that is part of a directive ends with the symbol `&` and the following line begins with the characters `CMF`, then the text following the characters `CMF` on the second line is taken as a continuation of the directive being processed.

An example of a `LAYOUT` directive extending over more than one line is:

```
CMF$  LAYOUT A ( :SERIAL, :NEWS, :SERIAL, :NEWS, :NEWS, &
CMF$   :NEWS )
```

The CM Fortran compiler generates a warning message for any comment line that begins with the letters `CMF`, but whose remaining characters do not have the syntax of a directive. The compiler generates an error for any comment line that has the syntax of a directive but which has improper arguments. (The compiler switch `-nodirective`, described in the *CM Fortran User's Guide*, can be used to inhibit recognition of directives altogether.) Note that the appearance of a comment within a directive (the comment form introduced by an exclamation point, as described in Section 2.3) causes the compiler to issue a warning message.

Directives are placed in the specification part of a program unit, or in an interface block. The CM Fortran compiler detects and reports as errors any inconsistent use of directives within a single program unit or interface block.

It is the programmer's responsibility to ensure that directives are used consistently throughout all program units comprising a program; inconsistent use of directives will cause run-time errors in lower-level software that may be difficult to relate to the program source. Use of an `INCLUDE` line is recommended to ensure that the specifications provided by a set of declarations and directives are used consistently throughout a program.

A.2 Directives That Control Array Layout

This section describes how the CM run-time system lays out arrays in the memories of the parallel processors, and how the compiler directives `LAYOUT` and `ALIGN` can be used to minimize interprocessor communication in array operations. The significance of the `LAYOUT` and `ALIGN` directives is the same regardless of execution model, although the system components that serve as "control processor" and "parallel processors" may differ. See the *CM Fortran Programming Guide* for a discussion of the various CM Fortran execution models and the *CM-5 CM Fortran Performance Guide* for more information on the performance implications of various data layouts on the CM-5 system.

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

A.2.1 Geometries

This section describes how the CM run-time system lays out arrays in the absence of directives. Later sections show how to use directives to alter the default layout.

Storage for a CM array is allocated at the same memory locations on all the available physical processors (nodes or vector units, depending on the execution model). The memory is organized as a *virtual grid* whose size is the number of physical processors times the *subgrid* (the amount of local memory). If the product of the array's dimensions is not a legal size for a virtual grid, the system allocates the next legal size and later masks out the unused memory locations. The system then lays out the array's dimensions within the virtual grid so as to minimize the amount of this *global padding*. The layout chosen is stored in a front-end object called a *geometry*.

A further constraint under the vector-units execution models (global, global/local, and nodal) is that the geometry's subgrid length (the number of elements allocated per processor) must be a multiple of 8, the length of the vector register. The system adds (and later masks) *vector-length padding* if needed to meet this constraint. The total grid size must therefore be a multiple of 8 x number-of-vector-units.

NOTE

The compiler switch `-nopadding` removes the vector-length constraint. With this switch, the subgrid can be any arbitrary length, and no vector-length padding is added. Note, however, that global padding is still added for any array whose size is not an integer multiple of machine size.

The geometry associated with an array is of the same rank as the array, though possibly larger if necessary to meet the constraints cited above on total size. Each array element can be regarded as a point in an *n*-dimensional Cartesian coordinate system: it has a *NEWS address* consisting of the *n* index values that locate the element along each axis of the geometry. Each element also has a *send address*, which uniquely identifies it within its virtual grid, regardless of the geometry's shape.

Copyright © 1989–1994 Thinking Machines Corporation

An array geometry's axes have the following attributes:

- *Size*. The axes can be any set of lengths whose product is a legal total size for a virtual grid (an integer multiple of the number of processors executing the program, times vector length where appropriate).
- *Element ordering.* This attribute indicates the order in which elements are encountered as index values increase along an axis of the geometry. The only meaningful ordering on a CM-5 is ordering by *NEWS address.* The CM-2/200 also supports ordering by *send address*, but this is not available on the CM-5.
- *Axis locality.* A given axis may be distributed across processors, or local to a single processor; that is, it may be either *parallel* or *serial.* In the absence of directives, all geometry axes for CM arrays are parallel.
- *Communications weight*. The *weight* of an axis indicates the relative importance of that axis with respect to efficiency of interprocessor communication. The higher the weight assigned to an axis (relative to the weights assigned to other axes of the geometry), the faster the interprocessor communication will be along that axis.

A.2.2 Canonical Layout of Arrays

In the absence of directives, a CM array is given a *canonical* layout in memory, meaning that:

- The first array element is placed at the origin of a geometry of the same rank. If any geometry axes are padded, the padding or *garbage elements* are placed at the ends of those axes.
- Every dimension is NEWS-ordered.
- The geometry is the smallest size that accommodates the array and meets the size requirements of the execution model. That is, the run-time system software distributes array elements in such as way as to minimize the amount of memory used, and thus to minimize the amount of global padding.
- All the axes of the geometry are given a communication weight of one. That is, none is particularly favored over the others for communication speed.

Conformable arrays with the canonical layout (or with identical noncanonical layouts) are given the same geometry, and their corresponding elements are stored in the same processors. As a result, arithmetic operations and assignments between conformable arrays with identical layouts require no interprocessor communication. If any dimensions differ in

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

their physical layout — because compiler directives have given them different relative weights, element orderings, or axis locality — the arrays are given different geometries. As a result, operations between them do require interprocessor communication.

A.2.3 Serial Ordering and Axis Locality

The LAYOUT directive can also be used to specify axis *locality* as an alternative to NEWS ordering.

NEWS-ordered array dimensions are treated as *parallel* dimensions in CM Fortran. In a parallel dimension, array elements are arranged across physical processors in *block* layout. That is, the geometry axis uses multiple physical processors and as many memory locations within each processor as needed. Array elements are arranged such that those with the lowest coordinates fill the first processor, then the next elements fill the second, and so on. The last few processors may contain only garbage data if the axis is padded.

In a *serial* dimension, the elements are all local to a single processor. Different subscript values in a serial dimension reference data elements at different *offsets* into processor memory.

In determining array layout, the run-time system initially considers only the parallel dimensions of an array. It determines the minimum grid size needed to accommodate the parallel (NEWS) dimensions and distributes and pads them as needed. Only then does it consider serial dimensions: it extends the per-processor subgrid to accommodate the serial dimensions. As a result, serial dimensions are always local and are never padded.

In CM Fortran, it is natural to view an array with mixed parallel and serial dimensions as an indexed collection of parallel arrays. An array with *p* parallel and *s* serial dimensions is treated as an *s*-dimensional serial array of *p*-dimensional parallel sections. Scalar index values are used to locate the appropriate parallel sections on which to perform array operations. Figure 7 illustrates this view of a 3-dimensional array that is serial in its first dimension. Figure 8 illustrates the view for a 4-dimensional array that is serial in its first and third dimensions.

Copyright © 1989–1994 Thinking Machines Corporation

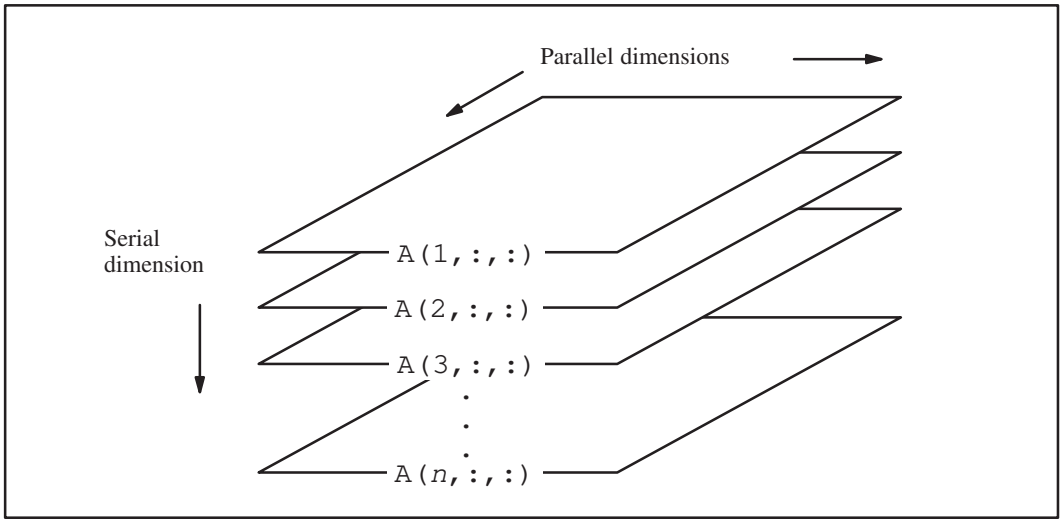

Figure 7. Layout of a 3-dimensional array whose first dimension is serial.

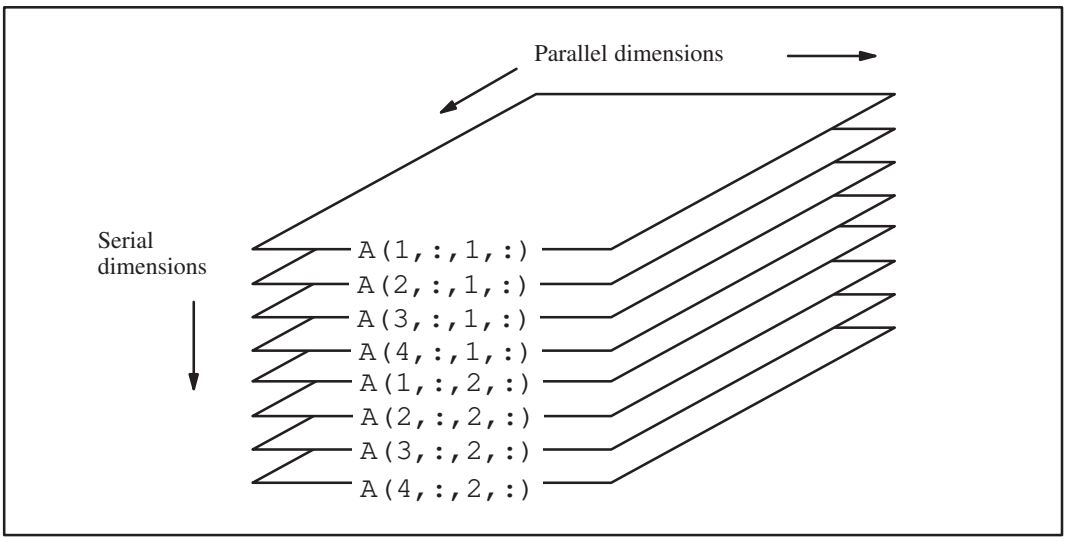

Figure 8. Layout of an array of shape `[4,m,2,n]` whose first and third dimensions are serial.

It also is sometimes appropriate to view an array with *p* parallel dimensions and *s* serial dimensions as a *p*-dimensional array of *s*-dimensional local arrays. Each *s*-dimensional array is stored in a single processor as a linear array of elements. This viewpoint is useful when referencing special-purpose external procedures that are designed to work with per-

Copyright © 1989–1994 Thinking Machines Corporation

processor arrays. It is also useful to the compiler in avoiding communication and applying special optimizations to operations on array dimensions that are known at compile time to be local.

A.2.4 The LAYOUT Directive (Nondetailed)

The LAYOUT directive (R19-2) provides a degree of program control over the layout of an array within distributed CM memory. It allows the user to specify the element ordering (distributed or local) and the relative communications weight of each axis of the array's geometry, or to specify that a dummy array is to assume the layout of the actual argument passed to it. The LAYOUT directive may indicate that *all* axes are serial, in which case the array is allocated in front-end memory.

(Section A.2.5 below describes additional axis descriptors that can be specified in a LAYOUT directive. These "detailed" forms of the directive provide program control over the exact mapping of user arrays to distributed memory.)

A *layout-directive* is: R19-2

`CMF$ LAYOUT` *layout-spec(s)*

A *layout-spec* is:

array-*name* (*axis-descriptor(s)*) R19-3

An *axis-descriptor* is one of: R19-4

- *[weight] [* `:NEWS` *]*
- `:SERIAL`
- `:`

A *weight* is: R19-5

scalar-integer-*constant*

- The number of elements in an *axis-descriptor(s)* list of a *layout-spec* must equal the rank of the array named in the *layout-spec*. (Note that an *axis-descriptor* may be empty, but the commas separating each *axis-descriptor* must be present.)
- The assumed layout *axis-descriptor* (`:`) is allowed only on assumed-shape and adjustable dummy arrays, but *not* on assumed-size dummy arrays or for the array return value of array-valued functions.
- The *axis-descriptor* `:SEND` is obsolete on CM-5 systems. If specified, it has the same effect as `:NEWS`. (Note that some CM-5 libraries do not accept arrays described with the *axis-descriptor* `:SEND`).

Copyright © 1989–1994 Thinking Machines Corporation

- It is illegal to align (see Section A.2.6) an assumed-layout dummy array with another array. (However, it is highly recommended that any local arrays that are to be used in the same expression as the assumed-layout array be aligned with the assumed-layout array.)

A given array-*name* must not appear in more than one `LAYOUT` directive in the same program unit or interface block. Also, in all the program units comprising a program, a common or dummy array must be given the same layout in the program unit in which it is defined and in all the interface blocks in which it is declared.

Each *layout-spec* (R19-3) specifies layout information for the array given by array-*name*. The array is given a geometry with the attributes specified by the *axis-descriptor(s)* list of that *layout-spec*. Each *axis-descriptor* (R19-4) defines attributes of the axis of the geometry in which the corresponding dimension of specified array is to be allocated. An optional *weight* associates an integer value with the corresponding axis of the geometry. The keywords `:NEWS` and `:SERIAL` control the ordering of the array elements in the geometry.

For each *axis-descriptor* in the list:

- `:NEWS` indicates that elements along that axis are to be ordered in NEWS ordering (the default order). NEWS-ordered elements are physically connected on a dimension if their grid indices on that dimension differ by one.
- `:SERIAL` indicates that array elements along that axis are to be allocated within the memory of a single processor.
- `:` indicates an assumed-layout array. This directive is used with dummy arrays and indicates that the ordering of this axis is to be assumed from the actual array passed to the procedure.

If an *axis-descriptor* is empty, the elements along the dimension corresponding to that axis are ordered in NEWS ordering with a default weight of 1.

The *weight* (R19-5) specifies the communications weight for an axis. If only one axis is given a weight (greater than 1), that axis is most favored. The CM Fortran compiler supplies a default weight of 1 for axes that are not given a specific weight by this directive.

Specifying nondefault ordering or communication weights for the axes does not change the meaning of a program (provided it is done consistently throughout the program), although it can cause communication operations to be required between conformable arrays that would normally be performed without communication.

An axis for which a processor ordering of `:SERIAL` is specified is called a *serial axis*. An axis for which a processor ordering of `:NEWS` is specified is called a *parallel axis*. If the geometry of an array has at least one parallel axis, the array is allocated on the parallel

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

processors. If the *all* axes are serial, the array is given only a front-end home (and may not be used in an array operation). If an array appears in a LAYOUT directive specifying at least one parallel axis (i.e., all axes are not specifically designated as serial), the array is given a CM home even if it never appears in an array operation within the program. (The array will not be allocated unless it is actually used in the program.)

The assumed-layout directive (:) specifies assumed-shape or adjustable dummy arrays that assume the layout of the actual array passed to the procedure. Interface blocks are not required for subroutines whose dummy arrays have assumed-layout directives. An example of the use of the assumed-layout directive is provided in Section A.2.4.1.

NOTE

Like any array subject to a LAYOUT directive, assumed-layout arrays must not be aligned with any other arrays. However, to avoid unnecessary communication between arrays, all local arrays that are used in the same expression as the assumed-layout dummy array should be aligned with the assumed-layout array by means of the ALIGN directive (see Section A.2.6).

Because the layout of assumed-layout arrays is not known at compile time, some compiler optimizations have no effect on these arrays.

A.2.4.1 Examples of LAYOUT Directives

The following example directive lays out a two-dimensional array A in CM memory with the default (NEWS) processor ordering.

```
      DIMENSION A(200,300)
CMF$  LAYOUT    A(10,5)
```

Both axes of the array are parallel, but communication is favored along the first axis. (Weights of 2, and 1 would have the same effect.)

Copyright © 1989–1994 Thinking Machines Corporation

In the next example, each of the subarrays A(I,J,:,:,:), for all possible values of I and J, is given the same geometry as the array B.

```
      DIMENSION A(M1,M2,M3,M4,M5), B(M3,M4,M5)
CMF$  LAYOUT A(:SERIAL,:SERIAL,10,7,1), B(10,7,1)
```

Array operations such as A(2,3,:,:,:)*B will be performed without communication.

In the example below, array FE is allocated on the control processor and A is placed in distributed CM memory.

```
      INTEGER FE(N), A(N,NP,NQ)
CMF$  LAYOUT FE(:SERIAL), A(:SERIAL,5:NEWS,5:NEWS)
```

The three-dimensional array A will be given the same geometry as would be used for a corresponding fully parallel *two*-dimensional array of shape NP x NQ. The first axis of A runs up processor memory, the second and third axes are laid out in NEWS order. The two parallel axes have equal weight for communications.

The array MIXED has five dimensions, the first two of which are serial and the rest of which are parallel by default.

```
      DOUBLE PRECISION MIXED(20,32,5000,5000,5000)
CMF$  LAYOUT MIXED( :SERIAL, :SERIAL, , , )
```

The array T is allocated in CM memory with two NEWS-ordered axes.

```
      DIMENSION T(10000,10000)
CMF$  LAYOUT    T( W1, W2:NEWS )
```

W1 and W2 are named constants used to set the communication weights for the two axes of array T.

The following example uses the assumed-layout directive to write two general-purpose subroutines that can be used with any possible layout of the array A. Subroutine MM_2 uses the assumed-layout directive with an adjustable array, and subroutine MM_3 uses it with an assumed-shape array.

Copyright © 1989–1994 Thinking Machines Corporation

```
      SUBROUTINE MM(A)
      REAL A(:,:)

      CALL MM_2(A, UBOUND(A,1), UBOUND(A,2))
      CALL MM_3(A)
      .
      .
      .
      END

      SUBROUTINE MM_2(A,N,M)
      REAL A(N,M)
CMF$  LAYOUT A(:,:)
      .
      .
      .
      END

      SUBROUTINE MM_3(A)
      REAL A(:,:)
CMF$  LAYOUT A(:,:)
      .
      .
      .
      END
```

NOTE

LAYOUT (or ALIGN) directives that give a noncanonical layout to dummy array arguments of a procedure must appear in the definition of that procedure, in each program unit in which actual array arguments are passed to the procedure (to give the proper layout to the actuals), and, if interface blocks are used, in every interface block describing that procedure. (See Section 18.2 for more information regarding argument passing.)

Also, LAYOUT and ALIGN directives that control the layout of arrays in common must be present in every program unit that uses those arrays.

Copyright © 1989–1994 Thinking Machines Corporation

A.2.5 The LAYOUT Directive (Detailed)

The LAYOUT directive accepts additional axis specifications that describe array layout in detail. These forms of the directive are intended for advanced users who are familiar with the architecture and the run-time system of the target CM platform. These features call for the user to specify such values as subgrid lengths and processor masks for array axes. These concepts are not meaningful at the language level or to the CM Fortran compiler, but are passed on as instructions to the run-time system.

A *layout-directive* is: R19-6

CMF$ LAYOUT *layout-spec(s)*

A *layout-spec* is:

array-*name* (*axis-descriptor(s)*) R19-7

A *detailed-axis-descriptor* is one of: R19-8

- :SERIAL
- :BLOCK= *axis-spec* :PROCS= *axis-spec*
- :BLOCK= *axis-spec* :PDESC= *proc-spec*

- An *axis-spec* is a decimal integer constant, an integer-valued dummy argument, or an integer variable in COMMON. Integer expressions are not supported.

- A *proc-spec* is a bit-mask specified as a decimal integer constant, an integer-valued dummy argument, or an integer variable in COMMON. Integer expressions are not supported.

The last two axis descriptors specify the subgrid length (:BLOCK) followed by either the number of processors desired for that axis (:PROCS) or a bit-mask indicating which processors are desired (:PDESC). The following restrictions also apply:

- While the subgrid length (:BLOCK) can be any integer; the number of processors (:PROCS) must be a power of 2.

- If one axis of an array is specified with :BLOCK :PROCS, then all axes of that array must be either :BLOCK :PROCS or :SERIAL.

- If one axis of an array is specified with :BLOCK :PDESC, then all axes of that array must be either :BLOCK :PDESC or :SERIAL.

- The compiler does not accept these detailed forms in an interface block when they use a dummy argument. Use only parameters, literal constants, or variables in COMMON in this situation.

- The bits specified in :PDESC for a given axis must be contiguous.

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

A.2.5.1 Examples of Detailed LAYOUT Directives

The array `A(16,16,4)` could be laid out on 8 processors as follows:

```
CMF$LAYOUT A(:SERIAL,:BLOCK=8:PROCS=2,:BLOCK=1:PROCS=4)
```

In this mapping, the first axis is totally local (it could also have been specified as `:BLOCK=16:PROCS=1`); the 16 elements of the second axis are divided equally between 2 processors; and the third axis is spread across 4 processors, one element in each.

The array `B(64,16)` could be laid out on 16 processors, using `:BLOCK :PDESC`, as follows:

```
CMF$  LAYOUT B(:BLOCK=16:PDESC=12,:BLOCK=4:PDESC=3)
```

Since the mask `:PDESC=12` has 2 nonzero bits, it represents a physical axis of length 4. The axis extent (64) equals subgrid length (16) times physical axis length (4). Similarly, the mask of 3 for the second axis has 2 nonzero bits, and 4*4 = 16.

A.2.5.2 Restrictions on LAYOUT with Deferred-Shape Arrays

Detailed `LAYOUT` directives on dynamic arrays use values upon entry, not at the time of allocation.

A.2.5.3 Restrictions on Array Size with Padding Enabled

There are certain restrictions on array size when specifying detailed layout. These restrictions apply only when the program's execution environment pads user arrays such that the number of elements allocated per processor (for non-serial axes) is evenly divisible by the length of the vector registers in the vector units.

These requirements vary by execution model; they apply only if the `cmf` switch `-padding` is enabled.

- For the CM-5 vector units (`-vu`) model with `-nopadding`, there are no restrictions on the product of the `:BLOCK` values.
- For the CM-5 vector units (`-vu`) model with `-padding`, the product of the `:BLOCK` values must be an integer multiple of 8. Axes labeled `:SERIAL` are excluded from the calculation.
- For the CM-5 nodes (`-sparc`) model, there are no restrictions on the product of the `:BLOCK` values.

Copyright © 1989–1994 Thinking Machines Corporation

For example, assume a 32-vector-unit partition and this detailed layout:

```
      REAL B(100,32)
CMF$  LAYOUT B(:SERIAL, :BLOCK=1:PROCS=32)
```

With `-padding` enabled, this layout causes a run-time error because it is not possible to to have a subgrid length of 1 for the parallel axis (the minimum subgrid length for all parallel axes is 8). With `-nopadding`, this directive gives the intended layout of one 100-element serial axis per processor.

A.2.6 The ALIGN Directive

The `ALIGN` directive (R19-9) specifies the layout of an array in terms of specified axes of another array whose layout is already determined. The `ALIGN` directive specifies which elements of two arrays are to be co-resident in the same processors by aligning each axis of a *source array* with a specified axis of a *target array.*

The `ALIGN` directive can speed up an application program by eliminating unnecessary communication of data when operations are to be performed involving arrays or array sections that would not normally be aligned in distributed memory. For example, if a particular vector is used repeatedly in array operations with rows of a matrix, it may pay to align the vector with the rows of the matrix.

An *align-directive* is: R19-9
`CMF$ ALIGN` *source-spec* `WITH` *target-spec*

A *source-spec* is: R19-10
source-array-*name* (index-*name(s)*)

A *target-spec* is: R19-11
target-array-*name* (*target-axis-spec(s)*)

A *target-axis-spec* is one of: R19-12
- index-*name*
- index-*name* + *offset-value*
- index-*value*
- + *offset-value*

An *offset-value* is: R19-13
integer-constant

An *index-value* is: R19-14
[−] integer-constant

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

- The number of index-*name*s in a *source-spec* must equal the rank of the array source-array-*name*.
- The number of elements in the *target-axis-spec(s)* list of a *target-spec* must equal the rank of the array target-array-*name*.
- Each index-*name* appearing in a *source-spec* must be distinct.
- Each index-*name* appearing in a *source-spec* must appear in a *target-axis-spec* for some axis of the associated *target-spec*.
- A given index-*name* must not be referenced by more than one *target-axis-spec*.
- The index-*name*s must appear in a *target-axis-spec* in the same order in which they appear in the *source-spec*.

The source-array-*name* of a *source-spec* (R19-10) indicates an array to be aligned, and each index-*name* specifies a symbolic name for the corresponding axis of that source array.

Within an `ALIGN` directive, a named integer constant may be referenced in a *target-axis-spec* (as an *offset-value* or an *index-value*) provided the constant name has not been used as an index-*name* in the *source-spec* of the directive (in which case it represents a source axis position rather than a symbolic constant value).

Each source array axis is aligned with a single target array axis, but it may be the case that a target array axis is not aligned with any source array axis. Whether and how a target array axis is aligned with a source array axis depends on the form of the *target-axis-spec* corresponding to that axis. For an `ALIGN` directive of the form

CMF$ ALIGN A(s_1,...,s_i, ..., s_n) WITH T(t_1,...,t_j,...,t_m)

where each s_i is an index-*name* and each t_j is a *target-axis-spec* (R19-12), the meaning of a particular t_j is given below for the four different forms it can take. In the discussion of each form, the section subscript triplets $a_{k\mathrm{L}}:a_{k\mathrm{U}}:a_{k\mathrm{S}}$ and $x_{k\mathrm{L}}:x_{k\mathrm{U}}:x_{k\mathrm{S}}$ are abbreviated as a_k and x_k in subscript positions where the lower bound, upper bound, and stride are not relevant.

CMF$ ALIGN A(s_1,...,s_i,...,s_n) WITH T(t_1,...,t_{j-1},s_i,t_{j+1},...,t_m)

This directive form gives array `A` the same geometry $\mathrm{G_T}$ as array `T`, aligning the ith axis of `A` with the jth axis of `T`. (The other axes of `A` are aligned as designated by the t_k for $k \neq j$.)

Copyright © 1989–1994 Thinking Machines Corporation

If all axes of T have parallel ordering (NEWS), the alignment causes each array element of the section

A(a_1, a_2, ..., a_{i-1}, a_{iL}:a_{iU}:a_{iS}, a_{i+1}, ..., a_n)

to be placed at a fixed offset into the corresponding position of the section given by

G_T(x_1, x_2, ..., x_{j-1}, x_{jL}:x_{jU}:x_{jS}, x_{j+1}, ..., x_m)

where

$x_{jL} = a_{iL}$ – LBOUND(A, i)
$x_{jU} = a_{iU}$ – LBOUND(A, i)
$x_{jS} = a_{iS}$

If $n < m$, the additional section subscripts of G_T are of the form c for some scalar integer constant c, so that the rank of each section is equal to n and the number of elements in the array section matches the number of elements in the geometry section.

CMF$ ALIGN A(s_1, ..., s_i, ..., s_n) WITH T(t_1, ..., t_{j-1}, s_i+c, t_{j+1}..., t_m)

This directive form gives array A the same geometry G_T as array T, aligning the ith axis of A with the jth axis of T, offset by c array elements. (The other axes of A are aligned as designated by the t_k for $k \neq j$.)

If all axes of T have parallel ordering (NEWS), the alignment causes each array element of the section

A(a_1, a_2, ..., a_{i-1}, a_{iL}:a_{iU}:a_{iS}, a_{i+1}, ..., a_n)

to be placed at a fixed offset into the corresponding position of the section given by

G_T(x_1, x_2, ..., x_{j-1}, x_{jL}:x_{jU}:x_{jS}, x_{j+1}, ..., x_m)

where

$x_{jL} = a_{iL}$ – LBOUND(A, i) + c
$x_{jU} = a_{iU}$ – LBOUND(A, i) + c
$x_{jS} = a_{iS}$

If $n < m$, the additional section subscripts of G_T are of the form c for some scalar integer constant c, so that rank of each section is equal to n and the number of elements in the array section matches the number of elements in the geometry section.

CMF$ ALIGN A(s_1, ..., s_n) WITH T(t_1, ..., t_{j-1}, c, t_{j+1}, ..., t_m)

This directive form gives array A the same geometry as array T, in the hyperplane defined by the array section T(:, :, ..., c, :, ..., :), for c in the jth subscript position. The value of c must be in the range LBOUND(T, j) to UBOUND(T, j). For a target specification t_j of this form, n must be less than m.

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

If all axes of T have parallel ordering (NEWS), the alignment causes the array element

$A(a_1, a_2, \ldots, a_n)$

to be placed at a fixed offset into the corresponding position of the section given by

$G_T(x_1, x_2, \ldots, x_{j-1}, x_{jL}, x_{j+1}, \ldots, x_m)$

where

$x_{jL} = c - 1$

CMF$ ALIGN $A(s_1, \ldots, s_n)$ WITH $T(t_1, \ldots, t_{j-1}, +c, t_{j+1}, \ldots, t_m)$

This directive form gives array A the same geometry as array T, in the hyperplane defined by the array section $T(:, :, \ldots, k, :, \ldots, :)$, for k in the jth subscript position, k = LBOUND(T, j) + c. The value of c must be in the range 0 to SIZE(T, j) −1. For a target specification t_j of this form, n must be less than m.

If all axes of T have parallel ordering (NEWS), the alignment causes the array element

$A(a_1, a_2, \ldots, a_n)$

to be placed at a fixed offset into the positions of the geometry given by

$G_T(x_1, x_2, \ldots, x_{j-1}, x_{jL}, x_{j+1}, \ldots, x_m)$

where

x_{jL} = LBOUND(T, j) + c − 1

Aligning a source array A with a target array T causes each axis of A to assume the ordering and weight of the corresponding axis of T. An axis of A that is aligned with a serial axis of T itself becomes a serial axis. The layout of a target array is never altered by an ALIGN directive.

A source array may be aligned with only one target array, but several source arrays may be aligned with the same target array. A source array may be aligned with a target array that is itself aligned with a different target array. Such a "chain" of alignments is permitted so long as it is not circular, as this would require the alignment of a source array to depend indirectly on its own alignment. Note that in the case of such chaining of alignments, the x_{jL} and x_{jU} values computed by the formulas above may require adjustment to compensate for the "shifting" of the target array T with respect to the origin of the geometry.

Each dimension of a source array must be completely subsumed by the dimension of the target array with which it is aligned. In other words, a source dimension axis may not be aligned in such a way that it would extend "beyond" either end of its associated target array axis.

Copyright © 1989–1994 Thinking Machines Corporation

A source array must not be aligned with a target array that has a shorter lifetime or smaller scope than the source array. These rules are summarized in Table 26. An x in table position *row* and *column* indicates that the source array kind indicated by the *row* heading may be aligned with the target array kind indicated by the corresponding column heading.

An array that has been given a layout specification using a LAYOUT directive must not appear as a source array in an ALIGN directive. Also, if the target array of an ALIGN directive also appears in a LAYOUT directive, it must appear first in the LAYOUT directive. The ALIGN directive may cause a source array to be given a geometry with serial axes.

Table 26. Alignment constraints by array kind.

	Target			
Source	Common	Saved	Dummy	Local
Common[1]	**x**	**–**	**–**	**–**
Saved	**x**	**x**	**–**	**–**
Dummy[2]	**x**	**x**	**x**	**x**
Local	**x**	**x**	**x**	**x**

[1]Arrays in common need not be in the same common block.
Includes explicit-shape and assumed-shape arrays.
[2]Includes automatic arrays.

A.2.6.1 Examples of ALIGN Directives

The ALIGN directive below causes V to be given the same geometry as A and placed in the same processors occupied by the first row of A (that is, A(1,:)):

```
      DIMENSION V(5), A(5,5)
CMF$  ALIGN V(I) WITH A(1,I)
```

as illustrated by:

```
VVVVV          AAAAA
.....          AAAAA
.....          AAAAA
.....          AAAAA
.....          AAAAA
```

The illustration depicts the layout of arrays V and A in CM memory. The identical shape of the two rectangular matrices representing V and A illustrates the fact that they are given geometries with identical shapes; array elements in the same relative position in the

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

matrices are in the same processors. A dot in a matrix diagram corresponds to memory in a processor that is not occupied by an array element.

The ALIGN directive below causes V to be given the same geometry as A and placed in the same processors occupied by the fifth row of A (that is, A(5,:)):

```
      DIMENSION V(5), A(5,5)
CMF$  ALIGN V(I) WITH A(5,I)
```

as illustrated by:

```
.....          AAAAA
.....          AAAAA
.....          AAAAA
.....          AAAAA
VVVVV          AAAAA
```

The ALIGN directive below causes V to be given the same geometry as A and placed in the same processors as occupied by the first column of A:

```
      DIMENSION V(5), A(5,5)
CMF$  ALIGN V(I) WITH A(I,1)
```

as illustrated by:

```
V....          AAAAA
V....          AAAAA
V....          AAAAA
V....          AAAAA
V....          AAAAA
```

Note that when aligning V with A above, a non-zero offset value can never be used in a *target-axis-spec* because the length of V is equal to the length of each axis of A. However, when aligning V with an axis of an array that is longer than V (such as B below), offsets are possible, though not required. When no offset is provided, the lower bound of the source array is aligned with the lower bound of the target array. In other words, the directive

```
CMF$  ALIGN V1(I) WITH V2(I)
```

is equivalent to

```
CMF$  ALIGN V1(I) WITH V2(I+0)
```

Following are some examples of ALIGN directives in which a target axis is specified using an index value (R19–14).

Copyright © 1989–1994 Thinking Machines Corporation

The directive

```
      DIMENSION V(5), B(10,10)
CMF$  ALIGN V(I) WITH B(2,I+4)
```

causes V to be given the same geometry as B and placed in the same processors that contain B(2,5:9), as illustrated by:

```
..........  BBBBBBBBBB
....VVVVV.  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
```

The directive

```
      DIMENSION V(5), B(10,10)
CMF$  ALIGN V(I) WITH B(I+2,5)
```

causes V to be given the same geometry as B and placed in the same processors that contain B(3:7,5):

```
..........  BBBBBBBBBB
..........  BBBBBBBBBB
....V.....  BBBBBBBBBB
....V.....  BBBBBBBBBB
....V.....  BBBBBBBBBB
....V.....  BBBBBBBBBB
....V.....  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
```


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

The directive

```
      DIMENSION A(5,5), B(10,10)
CMF$  ALIGN A(I,J) WITH B(I+5,J+2)
```

causes A to be given the same geometry as B and placed in the same processors that contain B(6:10,3:7):

```
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..AAAAA...  BBBBBBBBBB
..AAAAA...  BBBBBBBBBB
..AAAAA...  BBBBBBBBBB
..AAAAA...  BBBBBBBBBB
..AAAAA...  BBBBBBBBBB
```

The following examples illustrate the alignment of arrays declared with nondefault bounds.

Note that an *offset-value* in a *target-axis-spec* is always considered relative to the lower bound of the target array, regardless of what the lower bound is declared to be. The directive

```
      DIMENSION  R(5:10), Q(0:20)
CMF$  ALIGN R(I) WITH Q(I)
```

aligns the lower bound of R with the lower bound of Q (i.e., R(5) with Q(0)), so R is given the same geometry as Q and placed in the same processors as Q(0:5):

```
RRRRRR..............

QQQQQQQQQQQQQQQQQQQQQ
```

The directive

```
      DIMENSION  R(5:10), Q(0:20)
CMF$  ALIGN R(I) WITH Q(I+5)
```

aligns the elements of R with Q(5:10), as shown below:

```
.....RRRRRR.........

QQQQQQQQQQQQQQQQQQQQQ
```


Copyright © 1989–1994 Thinking Machines Corporation

The directive

```
      DIMENSION V(5), B(-8:1,-5:4)
CMF$  ALIGN V(I) WITH B(I+2,+4)
```

aligns the elements of `V` with `B(-6:-2,-2)`, as shown below:

```
..........  BBBBBBBBBB
..........  BBBBBBBBBB
....A.....  BBBBBBBBBB
....A.....  BBBBBBBBBB
....A.....  BBBBBBBBBB
....A.....  BBBBBBBBBB
....A.....  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
..........  BBBBBBBBBB
```

The following directive would achieve the same alignment:

```
CMF$  ALIGN V(I) WITH B(I+2,-1)
```

When the target array has rank higher than the source array, the index value that indicates the desired hyperplane of the target is *relative to* the declared bounds of the target.

The following example

```
      DIMENSION  R(5:10), M(100:109, 100:109)
CMF$  ALIGN R(I) WITH M(70,I)       ! ERRONEOUS
```

is in error because the index value `70` is not in the range `100` to `109`. The following alignment is correct:

```
      DIMENSION  R(5:10), M(100:109, 100:109)
CMF$  ALIGN R(I) WITH M(100,I)
```

Array `R` is given the same geometry as `M` and placed in the same processors occupied by `M(100,100:105)`, as illustrated below:

```
RRRRRR....  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
..........  MMMMMMMMMM
```


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

A.3 The COMMON Directive

The COMMON directive (R19-15) establishes a default home for all arrays in a given common block, to be used in the absence of LAYOUT or ALIGN directives. The directive may specify that front-end memory be allocated for all arrays in a common block.

A *common-directive* is one of: R19-15

- CMF$ COMMON *[* CMONLY *]* / *[* block-*name* *]* /
- CMF$ COMMON FEONLY / *[* block-*name* *]* /
- CMF$ COMMON INITIALIZED / *[* block-*name* *]* /

- If a block-*name* is specified, it must be the name of a common block that is declared in the specification part of the program unit in which the directive appears; otherwise, a blank common block must be declared in the specification part.

If a block-*name* is specified, the directive applies to all arrays in the common block with that name. If a block-*name* is not specified, the directive applies to all arrays in blank common.

The first and third forms of the directive indicate that the default array home for arrays in the specified common block is CM memory; the second form indicates that the default array home is front-end memory. In addition to indicating a CM array home, the third form indicates that front-end memory is to be allocated for the common block to allow static initialization of its arrays.

If any CM arrays in the specified common block are to be initialized in a BLOCK DATA program unit, the third form of the directive must be used. DATA statements supply initial values for the front-end initialization copy, and the initialized arrays are then copied to their CM home at program initialization. The values are never copied back to front-end memory during program execution, so they cannot be shared by program units compiled by other FORTRAN compilers.

The amount of front-end memory required for initialization is equal to the storage sequence size of the common block, i.e., the total size of all the arrays in the block. For large arrays, this can exceed the memory size of the control processor, resulting in a link-time failure.

NOTE: This directive *overrides* the default home of arrays in common that is established by the -fecommon or the -common_initialized compiler switches. Also, the home of an array established by the LAYOUT and ALIGN directives is not affected by the presence of the COMMON directive. See Section 18.1.2 for details.

Copyright © 1989–1994 Thinking Machines Corporation