Chapter 6

Data Motion

This chapter describes the utility procedures that perform three distinct kinds of data movement:

- array transfers between the control processor and the parallel processors
- data communication among the parallel processors
  - scatters with combining
  - parallel prefix operations
  - ranking and sorting
- data motion on serial (locally stored) array dimensions
  - table look-ups
  - gathers/scatters

6.1 Array Transfers

Two subroutines perform block transfers of array data between the serial control processor and the parallel processing unit:

```
CALL CMF_FE_ARRAY_TO_CM( DEST, SOURCE )

CALL CMF_FE_ARRAY_FROM_CM( DEST, SOURCE )
```

CMF_FE_ARRAY_TO_CM copies the contents of a front-end array **SOURCE** into a CM array **DEST**. **CMF_FE_ARRAY_FROM_CM** performs the opposite procedure. The source and destination arrays must match in shape and type.

Copyright © 1994 Thinking Machines Corporation

6.1.1 Language Comparison

The **FORALL** statement can express array transfers between CM and front-end, such as:

```
FORALL (I=1:N) FE_ARRAY(I) = CM_ARRAY(I)
```

However, in Version 2.2 this statement generates a **DO** loop with calls to **read-to-processor** or **write-from-processor**; that is, it transfers array data between the system components one element at a time. For this release, the array-transfer utilities give better performance.

Copyright © 1994 Thinking Machines Corporation

CMF_FE_ARRAY_TO_CM

Transfers the contents of a front-end array to a CM array of the same type and shape.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_FE_ARRAY_TO_CM( DEST, SOURCE )
```

Arguments

DEST	A CM array of the same type and shape as **SOURCE**. This array is the destination of the transfer.
SOURCE	A front-end array of any type.

Result

None.

Restriction

- The CM Fortran utility library procedures will not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.

See Also

CMF_FE_ARRAY_FROM_CM

Copyright © 1994 Thinking Machines Corporation

CMF_FE_ARRAY_FROM_CM

Transfers the contents of a CM array to a front-end array of the same type and shape.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_FE_ARRAY_FROM_CM( DEST, SOURCE )
```

Arguments

DEST	A front-end array of the same type and shape as **SOURCE**. This array is the destination of the transfer.
SOURCE	A CM array of any type.

Result

None.

Restriction

- The CM Fortran utility library procedures will not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.

See Also

CMF_FE_ARRAY_TO_CM

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

6.2 Scatters with Combining

The **CMF_SEND_** subroutines are used to scatter elements from a source array to specified locations in a destination array. If more than one value is sent to a single location, the values are combined together and with the original value in **DEST**, according to the operation specified in the subroutine name:

```
 CALL CMF_SEND_combiner
&    ( DEST, SEND_ADDRESS, SOURCE, MASK )
```

The combiners are **OVERWRITE**, **ADD**, **MAX**, **MIN**, **IOR**, **IAND**, and **IEOR**.

- **CMF_SEND_OVERWRITE** operates on CM arrays of any type. It arbitrarily chooses one of the colliding values to store in the destination location.
- **CMF_SEND_ADD** operates on any numeric type.
- **CMF_SEND_MAX** and **_MIN** operate on integer and real arrays (single- or double-precision).
- **CMF_SEND_IOR**, **_IAND**, and **_IEOR** operate on integer and logical arrays. They correspond to logical inclusive OR, logical AND, and logical exclusive OR, respectively. Integer operations are done on a bitwise basis.

The **MASK** argument controls which elements of **SOURCE** are selected for the operation. The **SEND_ADDRESS** argument is a CM array of destination addresses, constructed with the procedures described below. It must be conformable with the **SOURCE** array.

6.2.1 Language Comparison

The **FORALL** statement generates parallel send-with-combiner instructions for *n*-to-*m*-dimensional scatters when the possibility of data collisions exists. Except for arrays of high rank (as noted below), the performance of **FORALL** is comparable to that of **CMF_SEND_***combiner*.

To express send-with-combiner operations with **FORALL**, supply an index array (conformable with the source array) for each dimension of the destina-

Copyright © 1994 Thinking Machines Corporation

tion array. Then use a reduction function to combine multiple values being sent to the same destination element.

For example, a 1-to-1-dimensional send-with-add operation is written as:

```
FORALL(I=1:8) A(I)=SUM(B(1:1000),MASK=V(1:1000).EQ.I)
```

where
`A` is `A(8)` of numeric type.
`B` is `B(1000)` of numeric type.
`V` is `V(1000)` of type integer.

A 1-to-1-dimensional send-with-add operation that adds in the original destination value is written as:

```
FORALL(I=1:N) A(I) = A(I) + SUM(B(:), MASK=V(:).EQ.I)
```

For a 2-to-2-dimensional send-with-add, use an index array (conformable with the source array) for each dimension of the destination array:

```
 FORALL(I=1:N,J=1:M)
&       OUT(I,J) =
&       SUM(IN(:,:),
&           MASK=(X(:,:).EQ.I).AND.(Y(:,:).EQ.J))
```

A 1-to-2-dimensional send-with-add operation is written as:

```
 FORALL(I=1:N,J=1:M)
&       OUT(I,J)=
&       SUM(IN(:), MASK=X(:).EQ.I .AND. Y(:).EQ.J)
```

A 2-to-3-dimensional send-with-add operation is written as:

```
 FORALL(I=1:UBOUND(OUT,1), J=1:UBOUND(OUT,2)
&   K=1:UBOUND(OUT,3))
&   OUT(I,J,K) = SUM(IN,MASK=(     (COORD1.EQ.I).AND.
&                                   (COORD2.EQ.J).AND.
&                                   (COORD3.EQ.K)   ))
```

A permanent restriction on this use of **FORALL** is that it generates parallel instructions *only if* the rank of **OUT** *plus* the rank of **IN** is below a certain threshold. The threshold in Version 2.2 is 31. For arrays of higher rank, use the utility procedure **CMF_SEND_***combiner* for best performance.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

6.2.2 Constructing Send Address Arrays

A send address is an internal format for the linearized address of an *n*-dimensional coordinate. As such, it specifies an absolute location for a data element that is independent of its relative grid location.

Three procedures are used to convert grid coordinates (specifying the desired locations in the **DEST** array) into send addresses for use with **CMF_SEND_** :

```
      GEOMETRY = CMF_GET_GEOMETRY_ID( ARRAY )
      CALL CMF_MAKE_SEND_ADDRESS( ARRAY )
      CALL CMF_DEPOSIT_GRID_COORDINATE
     &    ( GEOMETRY, SEND_ADDRESS, AXIS, COORDINATE, MASK )
```

(A related subroutine, **CMF_MY_SEND_ADDRESS(ARRAY)**, fills an array with the send addresses of its own elements.)

To construct a send-address array for use with **CMF_SEND_**, perform the following steps:

1. Declare an array to hold the send addresses. The array must have the same shape and layout as the **SOURCE** array with which it will be used.

   ```
   INTEGER*8 SEND_ADDRESS
   DIMENSION SEND_ADDRESS( ... ) ! same shape as source
   ```

   NOTE: The CM-5 uses 8-byte send addresses. It accepts 4-byte send-address arrays, but there may be a performance penalty as the system coerces the values to 8-byte length. CM Fortran programs should declare 8-byte send-address arrays, preferably **INTEGER*8** under the vector-units execution model and **DOUBLE PRECISION** otherwise. **REAL*8** and **INTEGER*4** are also allowed.

2. Call **CMF_MAKE_SEND_ADDRESS** to initialize the send address array.

   ```
   CALL CMF_MAKE_SEND_ADDRESS( SEND_ADDRESS )
   ```

3. Use the function **CMF_GET_GEOMETRY_ID** to retrieve the geometry identifier of the **DEST** array:

   ```
   GEOMETRY = CMF_GET_GEOMETRY_ID( DEST )
   ```


Copyright © 1994 Thinking Machines Corporation

A geometry contains information about the shape and layout of a CM array, in this case, the array for which send addresses are being constructed. **GEOMETRY** should be declared as **INTEGER**. The value returned by **CMF_GET_GEOMETRY_ID** is passed to **CMF_DEPOSIT_GRID_COORDINATE**.

4. Call **CMF_DEPOSIT_GRID_COORDINATE** on the coordinates for one axis.

```
 CALL CMF_DEPOSIT_GRID_COORDINATE
&  ( GEOMETRY, SEND_ADDRESS, AXIS, COORDINATE, MASK )
```

   The subroutine **CMF_DEPOSIT_GRID_COORDINATE** incorporates the grid coordinates for one axis into the send addresses being constructed. The **COORDINATE** array contains the grid coordinates for the axis of **GEOMETRY** specified by **AXIS**.

   NOTE: The grid coordinates passed to **CMF_DEPOSIT_GRID_COORDINATE** should be 1-based. If you have specified a lower bound other than 1 for an array, you must adjust the coordinates specified in **COORDINATE** by subtracting 1 less than the lower bound.

5. Call **CMF_DEPOSIT_GRID_COORDINATE** again for each remaining axis of the **DEST** array, incorporating into the send address the **COORDINATE** values for that axis.

   At this point, the send-address array holds locations in the destination array; each location includes a processor number and an offset (measured in array elements) from the subgrid's base address in memory on that processor.

6. Pass the array of send addresses to the desired **CMF_SEND_** procedure.

```
 CALL CMF_SEND_combiner
&   ( DEST, SEND_ADDRESS, SOURCE, MASK )
```

   The call to **CMF_SEND_***combiner* performs the send, calculating absolute destination addresses using the base address in **DEST** and the relative offset from the base in **SEND_ADDRESS**.

Copyright © 1994 Thinking Machines Corporation

6.2.3 Address-Construction and Scatter Example

The example below shows how to construct send addresses for a call to **CMF_SEND_ADD**.

```
      SUBROUTINE SEND_ADD_EXAMPLE(OUT,IN,COORD1,COORD2,COORD3)
      IMPLICIT NONE
      REAL, DIMENSION(:,:,:) :: OUT
      REAL, DIMENSION(:,:) :: IN
      INTEGER, DIMENSION(:,:) :: COORD1,COORD2,COORD3
      DOUBLE PRECISION,DIMENSION(LBOUND(IN,1),UBOUND(IN,2)) :: SADDR
      INTEGER GEOM
CMF$  ALIGN SADDR(I,J) WITH IN(I,J)

      INCLUDE 'CMF_defs.h'
      ...
C   Get OUT array's geometry identifier
      GEOM = CMF_GET_GEOMETRY_ID(OUT)

C   Construct send addresses for OUT array
      CALL CMF_MAKE_SEND_ADDRESS(SADDR)
      CALL CMF_DEPOSIT_GRID_COORDINATE (GEOM, SADDR, 1, COORD1,
     &     .TRUE.)
      CALL CMF_DEPOSIT_GRID_COORDINATE (GEOM, SADDR, 2, COORD2,
     &     .TRUE.)
      CALL CMF_DEPOSIT_GRID_COORDINATE (GEOM, SADDR, 3, COORD3,
     &     .TRUE.)

C   Perform send-with-add
      CALL CMF_SEND_ADD(OUT, SADDR, IN, .TRUE.)

      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

CMF_SEND_[OVERWRITE, MAX, MIN, ADD, IOR, IAND, IEOR]

Sends elements from **SOURCE** to the locations in **DEST** specified by the corresponding addresses in **SEND_ADDRESS**. The procedure combines multiple values sent to the same **DEST** element, including the initial value of the **DEST** element, using the operation specified in the name of the send procedure.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_SEND_OVERWRITE( DEST, SEND_ADDRESS, SOURCE, MASK )

CALL CMF_SEND_ADD( DEST, SEND_ADDRESS, SOURCE, MASK )

CALL CMF_SEND_MAX( DEST, SEND_ADDRESS, SOURCE, MASK )

CALL CMF_SEND_MIN( DEST, SEND_ADDRESS, SOURCE, MASK )

CALL CMF_SEND_IOR( DEST, SEND_ADDRESS, SOURCE, MASK )

CALL CMF_SEND_IAND( DEST, SEND_ADDRESS, SOURCE, MASK )

CALL CMF_SEND_IEOR( DEST, SEND_ADDRESS, SOURCE, MASK )
```

Arguments

DEST — A CM array. The destination array. The data types allowed for each type of combiner are listed below.

SEND_ADDRESS — CM array. The send addresses used to determine where in **DEST** each element of **SOURCE** is sent. Send addresses are constructed using the CM Fortran utility library procedures **CMF_MAKE_SEND_ADDRESS**, **CMF_MY_SEND_ADDRESS**, and **CMF_DEPOSIT_GRID_COORDINATES**.

This array may be declared as **INTEGER*8** (**DOUBLE PRECISION** and **REAL*8** are also allowed) to support 8-byte send addresses, or as **INTEGER*4** to support 4-byte send addresses. (Note that the **INTEGER*8** type is supported only on CM-5 systems with vector units. On those systems, we recommend using **INTEGER*8**.) See **Description** below for details.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

SOURCE — A CM array. The source array. This array must have same shape and layout as **SEND_ADDRESS**. The data types allowed for each type of combiner are listed below.

MASK — A CM **LOGICAL** array or the scalar value **.TRUE.**. If **MASK** is a logical array, it must have the same shape and layout as **SEND_ADDRESS** and only those elements of **SOURCE** that correspond to **.TRUE.** values in **MASK** are sent to **DEST**. If **MASK** is the scalar value **.TRUE.**, all elements of **SOURCE** are sent.

Result

None.

Description

Each selected element of **SOURCE** is sent to the element of **DEST** specified by the send-address in the corresponding element of **SEND_ADDRESS**. If multiple elements of **SEND_ADDRESS** have the same value, the corresponding elements of **SOURCE** are combined. The arriving elements from **SOURCE** are combined with the initial element in **DEST**. The **MASK** argument controls which elements of **SOURCE** are selected for the computation.

NOTE: The CM-5 uses 8-byte send addresses. It accepts 4-byte send-address arrays, but there may be a performance penalty as the system coerces the values to 8-byte length. CM Fortran programs should declare 8-byte send-address arrays, preferably **INTEGER*8** under the vector-units execution model and **DOUBLE PRECISION** otherwise. **REAL*8** and **INTEGER*4** are also allowed.

CMF_SEND_OVERWRITE

The subroutine **CMF_SEND_OVERWRITE** operates on all the element types. The combining function used by this subroutine arbitrarily chooses one of the values being combined as the output. That is, if there are multiple elements of **INDEX** with the same index value, one of the corresponding values of **SOURCE** is arbitrarily chosen and written into **DEST**.

Copyright © 1994 Thinking Machines Corporation

CMF_SEND_ADD

The subroutine **CMF_SEND_ADD** can operate on numbers of the following types:

INTEGER*4

INTEGER*8

REAL

DOUBLE PRECISION

COMPLEX

DOUBLE COMPLEX

CMF_SEND_MAX, CMF_SEND_MIN

The subroutines **CMF_SEND_MAX** and **CMF_SEND_MIN** can operate on numbers of the following types:

INTEGER*4

INTEGER*8

REAL

DOUBLE PRECISION

CMF_SEND_IOR, CMF_SEND_IAND, CMF_SEND_IEOR

The subroutines **CMF_SEND_IOR**, **CMF_SEND_IAND**, and **CMF_SEND_IEOR** can operate on the following types:

INTEGER*4

INTEGER*8

LOGICAL

These operations correspond to logical inclusive OR, logical AND, and logical exclusive OR, respectively. For **INTEGER**s, these subroutines do the operation on a bitwise basis.

Restrictions

- The CM Fortran utility library procedures do not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.
- The **INTEGER*8** type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

See Also

```
CMF_MAKE_SEND_ADDRESS
CMF_MY_SEND_ADDRESS
CMF_DEPOSIT_GRID_COORDINATE
```

Copyright © 1994 Thinking Machines Corporation

CMF_GET_GEOMETRY_ID

Returns a geometry identifier for a CM array.

Syntax

```
INCLUDE 'CMF_defs.h'

GEOMETRY_ID = CMF_GET_GEOMETRY_ID( ARRAY )
```

Arguments

ARRAY	A CM array of any type.

Result

An **INTEGER*4** identifying the geometry of **ARRAY**. This identifier should only be passed to other CM Fortran library procedures that accept geometry identifiers, such as **CMF_DEPOSIT_GRID_COORDINATE**.

Description

This function returns an identifier for a geometry object that defines the shape and layout of **ARRAY**. Note that you cannot access the array information directly from this identifier. It can only be passed to other procedures, such as **CMF_DEPOSIT_GRID_COORDINATE**. Information about an array can be displayed by calling **CMF_DESCRIBE_ARRAY**.

See Also

CMF_DEPOSIT_GRID_COORDINATE
CMF_DESCRIBE_ARRAY

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_MAKE_SEND_ADDRESS

Initializes a send address array.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_MAKE_SEND_ADDRESS( ARRAY )
```

Arguments

ARRAY	A CM array. This array may be declared as **INTEGER*8** (**DOUBLE PRECISION** and **REAL*8** are also allowed) to support 8-byte send addresses, or as **INTEGER*4** to support 4-byte send addresses. (Note that the **INTEGER*8** type is supported only on CM-5 systems with vector units. On those systems, we recommend using **INTEGER*8**.) See **Description** below for details.

Result

None.

Description

This subroutine initializes **ARRAY** with NULL send addresses. This should be done before calling **CMF_DEPOSIT_GRID_COORDINATE**. **ARRAY** can be an **INTEGER*8** (**DOUBLE PRECISION** and **REAL*8** are also allowed) CM array, or an **INTEGER*4** CM array.

The CM-5 uses 8-byte send addresses, but also accepts 4-byte send address arrays. However, there may be a performance penalty for using 4-byte addresses, as the system coerces the values to 8-byte length. CM Fortran programs that compute send addresses should declare them as 8-byte values: **INTEGER*8** under the vector-units execution model, and **DOUBLE PRECISION** otherwise.

Copyright © 1994 Thinking Machines Corporation

Restrictions

- The CM Fortran utility library procedures will not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.
- The **INTEGER*8** type is supported only on CM-5 systems with vector units.

See Also

CMF_MY_SEND_ADDRESS
CMF_DEPOSIT_GRID_COORDINATE
CMF_SEND_*combiner*

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_MY_SEND_ADDRESS

Calculates the send address of each element in an array.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_MY_SEND_ADDRESS( ARRAY )
```

Arguments

ARRAY	A CM array. Each element of this array is filled with its own send address. This array can be declared as **INTEGER*8** (**DOUBLE PRECISION** and **REAL*8** are also allowed) to support 8-byte send addresses, or as **INTEGER*4** to support 4-byte send addresses. (Note that the **INTEGER*8** type is supported only on CM-5 systems with vector units. On those systems, we recommend using **INTEGER*8**.) See **Description** below for details.

Result

None.

Description

This subroutine calculates the send address for each element of **ARRAY** and fills each element with its own send address. **ARRAY** can be an **INTEGER*8** (**DOUBLE PRECISION** and **REAL*8** are also allowed) CM array, or an **INTEGER*4** CM array.

The CM-5 uses 8-byte send addresses, but also accepts 4-byte send address arrays. However, there may be a performance penalty for using 4-byte addresses, as the system coerces the values to 8-byte length. CM Fortran programs that compute send addresses should declare them as 8-byte values: **INTEGER*8** under the vector-units execution model, and **DOUBLE PRECISION** otherwise.

Copyright © 1994 Thinking Machines Corporation

Restrictions

- The CM Fortran utility library procedures will not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.
- The **INTEGER*8** type is supported only on CM-5 systems with vector units.

See Also

CMF_MAKE_SEND_ADDRESS
CMF_DEPOSIT_GRID_COORDINATE
CMF_SEND_*combiner*

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_DEPOSIT_GRID_COORDINATE

Modifies a send address to incorporate grid coordinates for one axis.

Syntax

```
 INCLUDE 'CMF_defs.h'

 CALL CMF_DEPOSIT_GRID_COORDINATE
&   ( GEOMETRY, SEND_ADDRESS, AXIS, COORDINATE, MASK )
```

Arguments

GEOMETRY — An **INTEGER*4** geometry ID as returned by **CMF_GET_GEOMETRY_ID**. The send address is computed for the geometry specified by this argument.

SEND_ADDRESS — A CM array in which the send addresses are stored.

This array may be declared as **INTEGER*8** (**DOUBLE PRECISION** and **REAL*8** are also allowed) to support 8-byte send addresses, or as **INTEGER*4** to support 4-byte send addresses. (Note that the **INTEGER*8** type is supported only on CM-5 systems with vector units. On those systems, we recommend using **INTEGER*8**, and **DOUBLE PRECISION** otherwise.) See **Description** below for details.

AXIS — An **INTEGER*4** specifying the axis number of the coordinates being deposited into the send address.

COORDINATE — A CM **INTEGER*4** array of the same shape and layout as **SEND_ADDRESS** and **MASK**. This array contains the grid coordinates to be incorporated into **SEND_ADDRESS**. These coordinates should be one-based and not larger than the length of the axis of the specified **GEOMETRY**.

MASK — A CM **LOGICAL** array of the same shape and layout as **SEND_ADDRESS** and **COORDINATE**, or the scalar value **.TRUE.**. If **MASK** is the scalar value **.TRUE.**, all the elements of **SEND_ADDRESS** are modified. If **MASK** is a **LOGICAL** array, only the elements of **SEND_ADDRESS** corresponding to the elements of **MASK** that contain **.TRUE.** are modified.

Copyright © 1994 Thinking Machines Corporation

Result

None.

Description

This subroutine modifies send addresses stored in selected elements of **SEND_ADDRESS**, along the axis specified by **AXIS**, to incorporate the grid coordinates stored in the corresponding elements of **COORDINATE**. The **MASK** argument controls which elements are selected for the computation. The **SEND_ADDRESS** array should be initialized by calling **CMF_MAKE_SEND_ADDRESS** before calling **CMF_DEPOSIT_GRID_COORDINATE**. The complete send address is built by calling **CMF_DEPOSIT_GRID_COORDINATE** repeatedly, once for each axis of the geometry.

SEND_ADDRESS can be declared as an **INTEGER*8** (**DOUBLE PRECISION** and **REAL*8** are also allowed) CM array, or as an **INTEGER*4**. (Note that the **INTEGER*8** type is supported only on CM-5 systems with vector units.) The CM-5 uses 8-byte send addresses, but also accepts 4-byte send address arrays. However, there may be a performance penalty for using 4-byte addresses, as the system coerces the values to 8-byte length. CM Fortran programs that compute send addresses should declare them as 8-byte values: **INTEGER*8** under the vector-units execution model, and **DOUBLE PRECISION** otherwise.

Restrictions

- The CM Fortran utility library procedures will not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.
- This routine assumes that the arrays have a lower bound of 1. All other lower bound values are ignored. If you have specified a lower bound other than 1 for an array, you must adjust the coordinates specified in **COORDINATE** by subtracting 1 less than the lower bound.
- The **INTEGER*8** type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

See Also

CMF_GET_GEOMETRY_ID
CMF_MAKE_SEND_ADDRESS
CMF_MY_SEND_ADDRESS
CMF_SEND_*combiner*

Copyright © 1994 Thinking Machines Corporation

6.3 Parallel Prefix Operations

The subroutines in this section perform parallel prefix operations, or *scans*, on one axis of an array:

```
 CALL CMF_SCAN_combiner ( DEST, SOURCE, SEGMENT, AXIS
&        DIRECTION, INCLUSION, SEGMENT_MODE, MASK )
```

These subroutines apply a binary operator cumulatively over the elements of the **SOURCE** array **AXIS**, combining each value with the cumulative result from all the values that precede it. The result for each element is stored in the corresponding element of the **DEST** array. The destination array may be the same array as the source.

The combiners are **COPY**, **ADD**, **MAX**, **MIN**, **IOR**, **IAND**, and **IEOR**.

- **CMF_SCAN_COPY** operates on CM arrays of any type. It copies the first element of an axis to all the other elements of that axis.
- **CMF_SCAN_ADD** operates on any numeric type.
- **CMF_SCAN_MAX** and **_MIN** operate on integer and real arrays (single- or double-precision).
- **CMF_SCAN_IOR**, **_IAND**, and **_IEOR** operate on integer and logical arrays. They correspond to logical inclusive OR, logical AND, and logical exclusive OR, respectively. Integer operations are done on a bitwise basis.

DIRECTION can be **CMF_UPWARD** or **CMF_DOWNWARD**. If the value is **CMF_UPWARD**, the values are combined from the lower numbered elements toward the higher. If the value is **CMF_DOWNWARD**, the values are combined from higher numbered elements toward the lower.

The scan can be limited to selected elements of the array axis through the **MASK** argument, a logical CM array conformable with **SOURCE** and **DEST**. Selected elements are those that correspond to a **.TRUE.** element in the **MASK** array. Array elements that correspond to a **.FALSE.** value in **MASK** are excluded from the computation, and the corresponding element of **DEST** is not changed.

In addition, the array elements along the axis may be partitioned into distinct sets, called *segments*, through the use of the **SEGMENT**, **SEGMENT_MODE**, and **INCLUSION** arguments. Each segment is treated as a separate set of val-

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

ues. The **SEGMENT** argument is a logical CM array conformable with **SOURCE** and **DEST**; the **SEGMENT_MODE** and **INCLUSION** arguments are predefined integer constants.

6.3.1 Language Comparison

A parallel prefix, or scan, operation is expressed with **FORALL** as:

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

This statement gives performance comparable to that of the utility procedure **CMF_SCAN_***combiner*. See the *CM Fortran Programming Guide* for more examples of scans expressed with **FORALL**.

6.3.2 Scan Segments

Each element of **SEGMENT** that contains **.TRUE.** marks the corresponding element of **SOURCE** as a segment boundary (the start or end of a segment). Segments begin (or end) with an element in which the value of **SEGMENT** is **.TRUE.**, and continue up (or down) the axis through all elements for which the value of **SEGMENT** is **.FALSE.**. The effect of these boundaries depends on the value of **SEGMENT_MODE**.

- If **SEGMENT_MODE** is **CMF_NONE**, then:
  - **SEGMENT** should be **CMF_NULL** or 0.
  - The scan operation proceeds along the entire length of the array axis and the values in **SEGMENT** have no effect.
  - When the **INCLUSION** argument is **CMF_INCLUSIVE**, the result at a given position includes the source value from that position. When the **INCLUSION** argument is **CMF_EXCLUSIVE**, it does not.
- If **SEGMENT_MODE** is **CMF_SEGMENT_BIT**, then:
  - The **MASK** argument does not affect the use of the **SEGMENT** array. That is, elements containing **.TRUE.** in the **SEGMENT** array create a segment boundary even if the corresponding value of **MASK** is **.FALSE.**. (The **MASK** array still selects the elements of **SOURCE** to be included, as described above.)

Copyright © 1994 Thinking Machines Corporation

- A **SEGMENT** value of **.TRUE.** indicates the *start* of a segment for upward scans, but the *end* of a segment for downward scans. That is, the **SOURCE** element corresponding to a **.TRUE. SEGMENT** element is the first element in a segment for an upward scan, but the last element in a segment for a downward scan. In downward scans, the new segment begins with the first unmasked element following the segment boundary.

- When the **INCLUSION** argument is **CMF_EXCLUSIVE**, the first **DEST** element in each segment is set to zero. (There is no scan result value for this element because in exclusive mode the first element of each segment of **SOURCE** is excluded from the scan.) When the **INCLUSION** argument is **CMF_INCLUSIVE**, the first element receives the corresponding element of source without combination and contributes to the segment results.

- If the value is **CMF_START_BIT**, then:

  - The **MASK** argument applies to the **SEGMENT** array as well as to the **SOURCE** array. That is, elements containing **.TRUE.** in **SEGMENT** array create a segment boundary only if the corresponding element of **MASK** is also **.TRUE.**.

  - A **SEGMENT** value of **.TRUE.** indicates the start of a segment for both upward and downward scans.

  - When the **INCLUSION** argument is **CMF_EXCLUSIVE**, the first **DEST** element in each segment (which is set to zero in **CMF_SEGMENT_BIT** scans) is used to store the final scan result of the preceding segment. Note that this result value does not contribute to the scan result for the segment in which it is stored. When the **INCLUSION** argument is **CMF_INCLUSIVE**, the first element receives the corresponding element of source without combination and contributes to the segment results.

6.3.3 Scanning Example

The table below shows the results for a single row along an axis being scanned by the subroutine **CMF_SCAN_ADD**. The **SOURCE** argument is an integer array filled with the value **1**. The **MASK** and **SEGMENT** arguments are

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

logical arrays with the values indicated at the top of the table (where **T** stands for **.TRUE.** and **F** stands for **.FALSE.**).

The table shows scan results for both directions, both inclusion modes, and all three segmentation modes. The dots indicate masked elements; the underlining groups elements that are considered part of the same segment.

MASK	T	T	T	T	F	F	F	F	T	T	F	F	T	T	T	F
SEGMENT	F	F	T	F	F	F	T	F	F	F	F	F	F	T	F	F
SOURCE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

DIRECTION	INCLUSION	SEGMENT-MODE	DEST
upward	exclusive	none	0 1 2 3 . . . . 4 5 . . 6 7 8 .
downward	exclusive	none	8 7 6 5 . . . . 4 3 . . 2 1 0 .
upward	inclusive	none	1 2 3 4 . . . . 5 6 . . 7 8 9 .
downward	inclusive	none	9 8 7 6 . . . . 5 4 . . 3 2 1 .
upward	exclusive	segment	0 1 0 1 . . . . 0 1 . . 2 0 1 .
downward	exclusive	segment	1 0 1 0 . . . . 2 1 . . 0 1 0 .
upward	inclusive	segment	1 2 1 2 . . . . 1 2 . . 3 1 2 .
downward	inclusive	segment	2 1 2 1 . . . . 3 2 . . 1 2 1 .
upward	exclusive	start	0 1 2 1 . . . . 2 3 . . 4 5 1 .
downward	exclusive	start	2 1 5 4 . . . . 3 2 . . 1 1 0 .
upward	inclusive	start	1 2 1 2 . . . . 3 4 . . 5 1 2 .
downward	inclusive	start	3 2 1 5 . . . . 4 3 . . 2 1 1 .

Copyright © 1994 Thinking Machines Corporation

CMF_SCAN_[ADD, MAX, MIN, COPY, IOR, IAND, IEOR]

Performs a scan along an axis on the selected elements of the source array, optionally within segments.

Syntax

```
 INCLUDE 'CMF_defs.h'

 CALL CMF_SCAN_ADD( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&         INCLUSION, SEGMENT_MODE, MASK )

 CALL CMF_SCAN_MAX( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&         INCLUSION, SEGMENT_MODE, MASK )

 CALL CMF_SCAN_MIN( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&         INCLUSION, SEGMENT_MODE, MASK )

 CALL CMF_SCAN_COPY( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&         INCLUSION, SEGMENT_MODE, MASK )

 CALL CMF_SCAN_IOR( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&         INCLUSION, SEGMENT_MODE, MASK )

 CALL CMF_SCAN_IAND( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&         INCLUSION, SEGMENT_MODE, MASK )

 CALL CMF_SCAN_IEOR( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&         INCLUSION, SEGMENT_MODE, MASK )
```

Arguments

DEST	A CM array. The destination array. The **DEST** and **SOURCE** arrays must be of the same type, shape, and layout.
SOURCE	A CM array. The source array. The **SOURCE** and **DEST** arrays must be of the same type, shape, and layout.
SEGMENT	A **LOGICAL** CM array of the same shape and layout as **DEST**, **SOURCE**, and **MASK**. **.TRUE.** values in the **SEGMENT** array are used as segment delimiters for the corresponding elements of the **SOURCE** array.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

If **SEGMENT_MODE** has a value of **CMF_NONE**, then this argument is ignored and may be **CMF_NULL** or 0. Otherwise, the scan operation is performed independently for each segment of **SOURCE** defined by **SEGMENT**. The arguments **SEGMENT_MODE**, **DIRECTION**, **INCLUSION**, and **MASK** control the way the scan proceeds over the segments. See the **Description** section below for details.

AXIS — An integer. The axis of **SOURCE** along which the scan is performed.

DIRECTION — An integer. The value can be **CMF_UPWARD** or **CMF_DOWNWARD**. If the value is **CMF_UPWARD**, the values are combined from the lower numbered elements toward the higher. If the value is **CMF_DOWNWARD**, the values are combined from higher numbered elements toward the lower.

INCLUSION — An integer. The value can be **CMF_EXCLUSIVE** or **CMF_INCLUSIVE**. If the value is **CMF_EXCLUSIVE**, the first element in each **SOURCE** segment (as defined by the **.TRUE.** elements of **SEGMENT**) is not included in the computation. If the value is **CMF_INCLUSIVE**, the first value in each segment is included.

SEGMENT_MODE — An **INTEGER**. One of the following integer values: **CMF_NONE**, **CMF_SEGMENT_BIT**, or **CMF_START_BIT**. This argument controls how the segments of **SOURCE** defined by the **SEGMENT** array are interpreted for the scan operation. See **Description** below for more information.

MASK — A CM **LOGICAL** array, or the scalar value **.TRUE.**.

If the value of **MASK** is a scalar **.TRUE.**, all the values of **SOURCE** will be included in the computation.

If **MASK** is a logical array, it must be of the same shape and layout as **DEST**, **SOURCE**, and **SEGMENT**. The values in **SOURCE** corresponding to values of **.FALSE.** in **MASK** are not included in the computation.

Result

None.

Copyright © 1994 Thinking Machines Corporation

Description

Each subroutine in this group performs a scan operation along an axis of the source array on the selected elements and puts the results in the destination array. Optionally, you may specify scan segments for the source array so that the scan operation is performed independently on distinct sections of the array axis.

Each of these subroutines cumulatively applies a binary operator over the selected elements of one axis of the source array **SOURCE**. Selected elements are those that correspond to a **.TRUE.** element in the **MASK** array. The scan operation combines each selected element of the array with the cumulative result from all the selected elements that precede it. The result for each of these elements is stored in the corresponding element of the destination array **DEST**. Array elements that correspond to a **.FALSE.** value in the **MASK** argument are excluded from the computation and the corresponding element of **DEST** is not changed.

In addition, the array elements along the axis may be partitioned into distinct sets, called *segments*, through the use of the **SEGMENT** and **SEGMENT_MODE** arguments. Each element of **SEGMENT** that contains **.TRUE.** marks the corresponding element of **SOURCE** as a segment boundary (the start or end of segment). Segments begin (or end) with each element in which the value of **SEGMENT** is **.TRUE.**, and continue up (or down) the axis through all elements where the value of **SEGMENT** is **.FALSE.**. The effect of these boundaries depends on the value of **SEGMENT_MODE**.

If **SEGMENT_MODE** is **CMF_NONE**, the operation specified by the subroutine proceeds along the entire length of the array axis and the values in **SEGMENT** have no effect. **SEGMENT** may be **CMF_NULL** or 0.

If **SEGMENT_MODE** is **CMF_SEGMENT_BIT**, then:

- The **MASK** argument does not affect the use of the **SEGMENT** array. That is, elements containing **.TRUE.** in the **SEGMENT** array create a segment boundary even if the corresponding value of **MASK** is **.FALSE.**. (The **MASK** array still selects the elements of **SOURCE** to be included as described above.)
- A **SEGMENT** value of **.TRUE.** indicates the *start* of a segment for upward scans, but the *end* of a segment for downward scans. That is, the **SOURCE** element corresponding to a **.TRUE. SEGMENT** element is the first element in a segment for an upward scan, but the last element in a segment for a downward scan. In downward scans, the new segment begins with the first unmasked element following the segment boundary.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

- When the **INCLUSION** argument is **CMF_EXCLUSIVE**, the first **DEST** element in each segment is set to zero. (There is no scan result value for this element because in exclusive mode the first element of each segment of **SOURCE** is excluded from the scan).

If **SEGMENT_MODE** is **CMF_START_BIT**, then:

- The **MASK** argument applies to the **SEGMENT** array as well as to the **SOURCE** array. That is, elements containing **.TRUE.** in the **SEGMENT** array create a segment boundary only if the corresponding element of **MASK** is also **.TRUE.**.
- A **SEGMENT** value of **.TRUE.** indicates the start of a segment for both upward and downward scans.
- When the **INCLUSION** argument is **CMF_EXCLUSIVE**, the first **DEST** element in each segment (which is set to zero in **CMF_SEGMENT_BIT** scans) is used to store the final scan result of the preceding segment. Note that this result value does not contribute to the scan result for the segment in which it is stored.

Information on each of the individual scan routines follows.

CMF_SCAN_ADD

The subroutine **CMF_SCAN_ADD** can operate on numbers of the following types:

```
INTEGER*4
INTEGER*8
REAL
DOUBLE PRECISION
COMPLEX
DOUBLE COMPLEX
```


Copyright © 1994 Thinking Machines Corporation

CMF_SCAN_MAX, CMF_SCAN_MIN

The subroutines **CMF_SCAN_MAX** and **CMF_SCAN_MAX** can operate on numbers of the following types:

```
INTEGER*4
INTEGER*8
REAL
DOUBLE PRECISION
```

CMF_SCAN_IOR, CMF_SCAN_IAND, CMF_SCAN_IEOR

The subroutines **CMF_SCAN_IOR**, **CMF_SCAN_IAND**, and **CMF_SCAN_IEOR** can operate on the following types:

```
LOGICAL
INTEGER*4
INTEGER*8
```

The operations **IOR**, **IAND**, and **IEOR** correspond to logical inclusive OR, logical AND, and logical exclusive OR, respectively.

For **INTEGER**s, these subroutines do the operation on a bitwise basis.

CMF_SCAN_COPY

The subroutine **CMF_SCAN_COPY** operates on all types. The binary operator used by this routine always returns its first argument. This subroutine is usually used to copy the first element in a segment to all the other elements of that segment.

Restriction

- The CM Fortran utility library procedures do not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

6.4 Ranking and Sorting

Two subroutines determine the numerical rank of the values along a dimension of a CM array; a third sorts the values by rank.

CMF_ORDER places the numerical rank of each element along the specified axis of a source array into the corresponding element of the destination array, under the control of a logical mask. The source, destination, and mask arguments must be conformable arrays.

```
CALL CMF_ORDER( DEST, SOURCE, AXIS, MASK )
```

CMF_RANK performs the same operation, but it also enables you to break the axis into segments. The direction argument (either **CMF_UPWARD** or **CMF_DOWNWARD**) determines whether the smallest or the largest value is given rank 1.

```
 CALL CMF_RANK( DEST, SOURCE, SEGMENT, AXIS,
&       DIRECTION, SEGMENT_MODE, MASK )
```

CMF_SORT places the sorted values themselves in the destination array. It, too, enables you to control the direction of the sort and to segment the source axis.

```
 CALL CMF_SORT( DEST, SOURCE, SEGMENT, AXIS,
&       DIRECTION, SEGMENT_MODE, MASK )
```

NOTE

The performance of these subroutines is significantly improved if the program is linked with the CM Scientific Software Library, **-lcmssl**.

6.4.1 Language Comparison

No comparable language features.

Copyright © 1994 Thinking Machines Corporation

6.4.2 Axis Segments

CMF_RANK and **CMF_SORT** take **SEGMENT** and **SEGMENT_MODE** arguments that partition the source array axis into distinct segments. Each segment is treated as a separate set of values for ranking purposes.

- The **SEGMENT** argument is a logical array that is conformable with **SOURCE** and **DEST**. Each element of **SEGMENT** that contains **.TRUE.** marks the corresponding element of **SOURCE** as a segment boundary. The effect of these boundaries depends on the value of **SEGMENT_MODE**. **SEGMENT** can be **CMF_NULL** or 0.

The **SEGMENT_MODE** argument is a pre-defined integer constant, one of **CMF_NONE**, **CMF_SEGMENT_BIT**, or **CMF_START_BIT**.

- If **SEGMENT_MODE** is **CMF_NONE**, the elements are sorted along the entire length of the array axis and the values in **SEGMENT** have no effect. **SEGMENT** can be **CMF_NULL** or 0.
- If **SEGMENT_MODE** is **CMF_SEGMENT_BIT**, then:
  - A **SEGMENT** value of **.TRUE.** indicates the *start* of a segment for upward sorts, but the *end* of a segment for downward sorts. That is, the **SOURCE** element corresponding to a **.TRUE. SEGMENT** element is the first element in a segment for an upward sort, but the last element in a segment for a downward sort. In downward sorts, the new segment begins with the first unmasked element following the segment boundary.
  - The **MASK** argument does not affect the use of the **SEGMENT** array. That is, elements containing **.TRUE.** in the **SEGMENT** array create a segment boundary even if the corresponding value of **MASK** is **.FALSE.**. (The **MASK** array still selects the elements of **SOURCE** to be included.)
- If **SEGMENT_MODE** is **CMF_START_BIT**, then:
  - A **SEGMENT** value of **.TRUE.** indicates the start of a segment for both upward and downward sorts.
  - The **MASK** argument applies to the **SEGMENT** array as well as to the **SOURCE** array. That is, elements containing **.TRUE.** in the **SEGMENT** array create a segment boundary only if the corresponding element of **MASK** is also **.TRUE.**.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Specific behavior of **CMF_RANK** and **CMF_SORT** on segmented axes is illustrated in the examples shown below. (Any element marked XXX contains garbage data.) Note that the segmentation is not carried over into the destination array:

- **CMF_RANK** ranks each element within its own segment, but the numbering of the elements is continuous along the entire length of the axis. In the final example below, **DEST** is `[XXX 1 3 2 ]`, not `[XXX 1 2 1 ]`.

- **CMF_SORT** sorts each segment independently, but the values are placed in the destination without regard to segments. In the final example below, **DEST** is `[7.0, 2.0, 3.0, XXX]`, not `[7.0, XXX, 2.0, 3.0]`).

See Section 6.3.3 for more examples of how segments behave under the various segmentation modes.

6.4.3 Ranking and Sorting Examples

Upward sort and rank:

```
If SOURCE =          [1.0 7.0 3.0 2.0]
and SEGMENT =        [T   F   F   F  ]

then rank DEST =     [1   4   3   2]
and sort DEST =      [1.0 2.0 3.0 7.0]
```

Downward sort and rank:

```
If SOURCE =          [1.0 7.0 3.0 2.0]
and SEGMENT =        [T   F   F   F  ]

then rank DEST =     [4   1   2   3]
and sort DEST =      [7.0 3.0 2.0 1.0]
```

Upward sort and rank with mask:

```
If SOURCE =          [1.0 7.0 3.0 2.0]
and SEGMENT =        [T   F   F   F  ]
and MASK =           [T   T   F   T  ]
```


Copyright © 1994 Thinking Machines Corporation

then rank **DEST** = [1 3 XXX 2]
and sort **DEST** = [1.0 2.0 7.0 XXX]

Segmented upward sort and rank:

If **SOURCE** = [1.0 7.0 3.0 2.0]
and **SEGMENT** = [T F T F]

then rank **DEST** = [1 2 4 3]
and sort **DEST** = [1.0 7.0 2.0 3.0]

Segmented upward sort and rank with mask:

If **SOURCE** = [1.0 7.0 3.0 2.0]
and **SEGMENT** = [T F T F]
and **MASK** = [F T T T]

then rank **DEST** = [XXX 1 3 2]
and sort **DEST** = [7.0 2.0 3.0 XXX]

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_ORDER

Places the numerical rank of each element of a source array in the corresponding element of the destination array.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_ORDER( DEST, SOURCE, AXIS, MASK )
```

Arguments

DEST	A CM **INTEGER** array. The destination array. The **DEST** and **SOURCE** arrays must be of the same shape and layout.
SOURCE	A CM array of any type except for **COMPLEX** or **DOUBLE COMPLEX**. The order of this array is stored in **DEST**.
AXIS	Integer. The axis over which to do the ordering.
MASK	A logical CM array conforming to **DEST**, or the scalar value **.TRUE.**. If **MASK** is the scalar value **.TRUE.**, all the elements of **DEST** are modified. If **MASK** is a logical array, only the elements of **DEST** corresponding to the elements of **MASK** that contain **.TRUE.** are modified.

Result

None.

Description

For each element of **SOURCE** with a **MASK** value of **.TRUE.**, **CMF_ORDER** places the numerical rank of that element in the corresponding element of **DEST**. Each row along the specified **AXIS** is treated as a separate set of values to be ordered. The rank values computed by this subroutine will always be 1 to *n* inclusive, where *n* is the number of items in each set of values to be ordered. This is true regardless of the lower bound of **SOURCE**.

Copyright © 1994 Thinking Machines Corporation

Note

The performance of this subroutine is significantly improved if the program is linked with the CM Scientific Software Library, **`-lcmssl`**.

Restriction

- The CM Fortran utility library procedures do not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.

See Also

`CMF_RANK`
`CMF_SORT`

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_RANK

Places the numerical rank of each selected element along an array axis, or axis segment, into the corresponding element of the destination array.

Syntax

```
 INCLUDE 'CMF_defs.h'

 CALL CMF_RANK( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&               SEGMENT_MODE, MASK )
```

Arguments

DEST — A CM **INTEGER** array. The destination array. The **DEST** and **SOURCE** arrays must be of the same shape and layout.

SOURCE — A CM array of any type except for **COMPLEX** or **DOUBLE COMPLEX**. The source array. The **SOURCE** and **DEST** arrays must be of the same shape and layout.

SEGMENT — A **LOGICAL** CM array of the same shape and layout as **DEST**, **SOURCE**, and **MASK**. **.TRUE.** values in the **SEGMENT** array are used as segment delimiters for the corresponding elements of the **SOURCE** array.

If **SEGMENT_MODE** has a value of **CMF_NONE**, then this argument is ignored and may be **CMF_NULL** or 0. Otherwise, each segment is ranked independently. The arguments **SEGMENT_MODE**, **DIRECTION**, and **MASK** control the way the ranking proceeds over the segments. See the **Description** section below for details.

AXIS — An integer. The axis of **SOURCE** to be ranked.

DIRECTION — An integer. The value can be **CMF_UPWARD** or **CMF_DOWNWARD**. If the value is **CMF_UPWARD**, the values are ranked from the smallest value to the largest; rank 1 is assigned to the smallest value. If the value is **CMF_DOWNWARD**, the values are ranked from largest value to the smallest; rank 1 is assigned to the largest value.

Copyright © 1994 Thinking Machines Corporation

SEGMENT_MODE — An **INTEGER**. One of the following integer values: **CMF_NONE**, **CMF_SEGMENT_BIT** or **CMF_START_BIT**. This argument controls how the segments of **SOURCE** defined by the **SEGMENT** array are interpreted. See **Description** below for more information.

MASK — A CM **LOGICAL** array, or the scalar value **.TRUE.**

If the value of **MASK** is a scalar **.TRUE.**, all the values of **SOURCE** will be included in the ranking.

If **MASK** is a logical array, it must be of the same shape and layout as **DEST**, **SOURCE**, and **SEGMENT**. The values in **SOURCE** corresponding to values of **.FALSE.** in **MASK** are not included in the ranking.

Result

None.

Description

This subroutine determines the numerical ranking of the values stored in the selected elements along the specified axis of the **SOURCE** array and places the rank of each element in the corresponding element of the destination array **DEST**. Selected elements are those that correspond to a **.TRUE.** element in the **MASK** array. The rankings computed by this subroutine are always 1 to *n* inclusive, where *n* is the number of elements in the set of values to be ordered. Array elements that correspond to a **.FALSE.** value in the **MASK** argument are not included in the ranking and the corresponding element of **DEST** is not changed.

The rank is always stable; for each pair of elements that contain equal values, the element with the lower grid coordinate along the ranking axis is assigned the lower numbered rank, regardless of the direction of the ranking.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

In addition, the array elements along the axis may be partitioned into distinct sets, called *segments*, through the use of the **SEGMENT** and **SEGMENT_MODE** arguments. Each segment along the specified **AXIS** is treated as a separate set of values to be ordered. Each element of **SEGMENT** that contains **.TRUE.**, marks the corresponding element of **SOURCE** as a segment boundary (the start or end of a segment). Segments begin (or end) with each element in which the value of **SEGMENT** is **.TRUE.**, and continue up (or down) the axis through all elements where the value of **SEGMENT** is **.FALSE.**. The effect of these boundaries depends on the value of **SEGMENT_MODE**.

If **SEGMENT_MODE** is **CMF_NONE**, the elements are ranked along the entire length of the array axis and the values in **SEGMENT** have no effect. **SEGMENT** may be **CMF_NULL** or 0.

If **SEGMENT_MODE** is **CMF_SEGMENT_BIT**, then:

- The **MASK** argument does not affect the use of the **SEGMENT** array. That is, elements containing **.TRUE.** in the **SEGMENT** array create a segment boundary even if the corresponding value of **MASK** is **.FALSE.** (The **MASK** array still selects the elements of **SOURCE** to be included as described above.)

- A **SEGMENT** value of **.TRUE.** indicates the *start* of a segment for upward ranks, but the *end* of a segment for downward ranks. That is, the **SOURCE** element corresponding to a **.TRUE. SEGMENT** element is the first element in a segment for an upward rank, but the last element in a segment for a downward rank. In downward ranks, the new segment begins with the first unmasked element following the segment boundary.

If the value is **CMF_START_BIT**, then:

- The **MASK** argument applies to the **SEGMENT** array as well as to the **SOURCE** array. That is, elements containing **.TRUE.** in **SEGMENT** array create a segment boundary only if the corresponding element of **MASK** is also **.TRUE.**.

- A **SEGMENT** value of **.TRUE.** indicates the start of a segment for both upward and downward ranks.

Note

The performance of this subroutine is significantly improved if the program is linked with the CM Scientific Software Library, **-lcmssl**.

Copyright © 1994 Thinking Machines Corporation

Restriction

- The CM Fortran utility library procedures do not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.

See Also

`CMF_ORDER`
`CMF_SORT`

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_SORT

Sorts the elements along an array axis (or axis segment) by numerical ranking and places the values in order in the destination array.

Syntax

```
 INCLUDE 'CMF_defs.h'

 CALL CMF_SORT( DEST, SOURCE, SEGMENT, AXIS, DIRECTION,
&               SEGMENT_MODE, MASK )
```

Arguments

DEST — A CM array of the same type as **SOURCE**. The destination array. The **DEST** and **SOURCE** arrays must be of the same shape and layout.

SOURCE — A CM array of any type except for **COMPLEX** or **DOUBLE COMPLEX**. The **SOURCE** and **DEST** arrays must be of the same shape and layout.

SEGMENT — A **LOGICAL** CM array of the same shape and layout as **DEST**, **SOURCE**, and **MASK**. **.TRUE.** values in the **SEGMENT** array are used as segment delimiters for the corresponding elements of the **SOURCE** array.

If **SEGMENT_MODE** has a value of **CMF_NONE**, then this argument is ignored and may be **CMF_NULL** or 0. Otherwise, each segment is sorted independently. The arguments **SEGMENT_MODE**, **DIRECTION**, and **MASK** control the way the sorting proceeds over the segments. See the **Description** section below for details.

AXIS — An integer scalar indicating the axis of **SOURCE** to be sorted.

DIRECTION — An integer. The value can be **CMF_UPWARD** or **CMF_DOWNWARD**. If the value is **CMF_UPWARD**, the values are sorted from the smallest value to the largest; the smallest value is assigned to the first element of the corresponding axis of **DEST**. If the value is **CMF_DOWNWARD**, the values are sorted from the largest value to the smallest; the largest value is assigned to the first element of the corresponding axis of **DEST**.

Copyright © 1994 Thinking Machines Corporation

SEGMENT_MODE — An **INTEGER**. One of the following integer values: **CMF_NONE**, **CMF_SEGMENT_BIT** or **CMF_START_BIT**. This argument controls how the segments of **SOURCE** defined by the **SEGMENT** array are interpreted. See **Description** below for more information.

MASK — A CM **LOGICAL** array, or the scalar value **.TRUE.**.

If the value of **MASK** is a scalar **.TRUE.**, all the values of **SOURCE** are sorted.

If **MASK** is a logical array, it must be of the same shape and layout as **DEST**, **SOURCE**, and **SEGMENT**. The values in **SOURCE** corresponding to values of **.FALSE.** in **MASK** are not sorted.

Result

None.

Description

This subroutine sorts the values stored in the selected elements along one axis of the **SOURCE** array by numerical ranking, and stores the values in order into the **DEST** array. Selected elements are those that correspond to a **.TRUE.** element in the **MASK** array. Array elements that correspond to a **.FALSE.** value in the **MASK** argument are not sorted and the corresponding element of **DEST** is not changed.

In addition, the array elements along the axis may be partitioned into distinct sets, called *segments*, through the use of the **SEGMENT** and **SEGMENT_MODE** arguments. Each segment along the specified **AXIS** is treated as a separate set of values to be sorted. Each element of **SEGMENT** that contains **.TRUE.**, marks the corresponding element of **SOURCE** as a segment boundary (the start or end of a segment). Segments begin (or end) with each element in which the value of **SEGMENT** is **.TRUE.**, and continue up (or down) the axis through all elements where the value of **SEGMENT** is **.FALSE.**. The effect of these boundaries depends on the value of **SEGMENT_MODE**.

If **SEGMENT_MODE** is **CMF_NONE**, the elements are sorted along the entire length of the array axis and the values in **SEGMENT** have no effect. **SEGMENT** may be **CMF_NULL** or 0.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

If **SEGMENT_MODE** is **CMF_SEGMENT_BIT**, then:

- The **MASK** argument does not affect the use of the **SEGMENT** array. That is, elements containing **.TRUE.** in the **SEGMENT** array create a segment boundary even if the corresponding value of **MASK** is **.FALSE.** (The **MASK** array still selects the elements of **SOURCE** to be included as described above.)
- A **SEGMENT** value of **.TRUE.** indicates the *start* of a segment for upward sorts, but the *end* of a segment for downward sorts. That is, the **SOURCE** element corresponding to a **.TRUE. SEGMENT** element is the first element in a segment for an upward sort, but the last element in a segment for a downward sort. In downward sorts, the new segment begins with the first unmasked element following the segment boundary.

If the value is **CMF_START_BIT**, then:

- The **MASK** argument applies to the **SEGMENT** array as well as to the **SOURCE** array. That is, elements containing **.TRUE.** in **SEGMENT** array create a segment boundary only if the corresponding element of **MASK** is also **.TRUE.**.
- A **SEGMENT** value of **.TRUE.** indicates the start of a segment for both upward and downward sorts.

Note

The performance of this subroutine is significantly improved if the program is linked with the CM Scientific Software Library, **-lcmssl**.

Restriction

- The CM Fortran utility library procedures do not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.

See Also

CMF_RANK
CMF_SORT

Copyright © 1994 Thinking Machines Corporation

6.5 Table Look-Ups

Three procedures are used to perform "table look-ups," that is, vector indirection on a serial dimension of a CM array. Under the conditions noted below, the look-up utility uses the indirect addressing hardware on the CM processing elements to perform local memory accesses, rather than generating communication.

```
 TABLE_ID = CMF_ALLOCATE_TABLE
&   ( TYPE, ELEMENT_COUNT, INITIAL_VALUES )

 CALL CMF_LOOKUP_IN_TABLE
&   ( DEST, TABLE_ID, INDEX, MASK )

 CALL CMF_DEALLOCATE_TABLE( TABLE_ID )
```

The function **CMF_ALLOCATE_TABLE** allocates and initializes a look-up table, placing a copy in the memory of each processing element; it returns an integer that serves as a pointer to the table. **TYPE** is the type of data to be stored in the table; it is specified as one of:

```
CMF_LOGICAL,
CMF_S_INTEGER, CMF_LONG_S_INTEGER
CMF_FLOAT, CMF_DOUBLE,
CMF_COMPLEX, CMF_DOUBLE_COMPLEX
```

The initial values must be stored in a front-end array and must be of the appropriate type.

CMF_LOOKUP_IN_TABLE uses an array of (integer) indices to retrieve values from the look-up table, and stores them in a destination array of the same type. **CMF_DEALLOCATE_TABLE** deallocates a look-up table. For example,

```
 REAL DEST(8192), TABLE_VALUES(100)
 INTEGER TABLE
 INTEGER INDEX(8192)
 ...

 TABLE = CMF_ALLOCATE_TABLE
&   (CMF_FLOAT, 100, TABLE_VALUES)

 CALL CMF_LOOKUP_IN_TABLE
&   (DEST, TABLE, INDEX, .TRUE.)
 ...
 CALL CMF_DEALLOCATE_TABLE(TABLE)
 ...
```


Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

6.5.1 Language Comparison

Under certain circumstances, the table look-up procedures are significantly faster than assignments of conventionally allocated arrays using **FORALL**. The circumstances are:

- The contents of the look-up table rarely or never change.
- The look-up table is relatively small, that is, it fits into the memory of a single processing element. The size restriction by CM Fortran execution model is:
  - CM-5 vector-units model: Table size is limited by the amount of memory on a vector unit. (Note that this applies to the global vector-unit execution model, the global/local model, and the nodal model.)
  - CM-5 SPARC nodes model: Table size is limited by the amount of memory on a SPARC node.

Copyright © 1994 Thinking Machines Corporation

CMF_ALLOCATE_TABLE

Allocates a lookup table and returns a table identifier.

Syntax

```
INCLUDE 'CMF_defs.h'

TABLE_ID = CMF_ALLOCATE_TABLE( TYPE, ELEMENT_COUNT,
&                              INITIAL_VALUES )
```

Arguments

`TYPE` A predefined integer constant that indicates the data type of the table elements. It must be one of:

- `CMF_LOGICAL`
- `CMF_S_INTEGER`
- `CMF_LONG_S_INTEGER`
- `CMF_FLOAT`
- `CMF_DOUBLE`
- `CMF_COMPLEX`
- `CMF_DOUBLE_COMPLEX`

`ELEMENT_COUNT` An integer scalar specifying the number of elements in the lookup table.

`INITIAL_VALUES` A front-end array of the same type as **`TYPE`** containing the values to be used to initialize the table.

Result

An **`INTEGER`** scalar that serves as an identifier for the lookup table.

This result can only be used as argument to the other CM Fortran utility routines, **`CMF_LOOKUP_IN_TABLE`** and **`CMF_DEALLOCATE_TABLE`**, when operating on this table.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Description

CMF_ALLOCATE_TABLE allocates a table as specified by **TYPE** and **ELEMENT_COUNT**, initializes it to the values specified in **INITIAL_VALUES**, and returns a table identifier. Values can be retrieved from this table in parallel by passing the table identifier as an argument to **CMF_LOOKUP_IN_TABLE**.

These lookup table procedures are significantly faster than indexing into a conventionally allocated table with **FORALL**, as long as the following conditions are met:

- The content of the table never or rarely changes.
- The table is relatively small. Specifically, it must use less memory than is available on a single processing element (vector unit or node, depending on the execution model).

See Also

CMF_LOOKUP_IN_TABLE
CMF_DEALLOCATE_TABLE

Copyright © 1994 Thinking Machines Corporation

CMF_LOOKUP_IN_TABLE

Performs parallel array reference on a lookup table.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_LOOKUP_IN_TABLE( DEST, TABLE, INDEX, MASK )
```

Arguments

DEST	A CM array. The destination array. The values retrieved from the table are stored into this array.
TABLE	Integer. The identifier for the lookup table to be referenced as returned by **CMF_ALLOCATE_TABLE**. Only lookup tables allocated by the **CMF_ALLOCATE_TABLE** subroutine can be referenced by this procedure.
INDEX	An **INTEGER** CM array containing the indices to be used to reference **TABLE**. The indices must have a lower bound of 1.
MASK	A CM **LOGICAL** array or the scalar value **.TRUE.**. If **MASK** is the scalar value **.TRUE.**, all the elements of **DEST** are modified. If **MASK** is a **LOGICAL** array, only the elements of **DEST** corresponding to the elements of **MASK** that contain **.TRUE.** are modified.

Result

None.

Description

CMF_LOOKUP_IN_TABLE performs a parallel array reference on **TABLE**.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

DEST, **INDEX**, and **MASK** (if an array) must be conformable CM arrays. Each element of **INDEX** is used as an index into **TABLE**, and the value retrieved from that location in **TABLE** is stored into the corresponding element of **DEST**.

Using **CMF_ALLOCATE_TABLE** and **CMF_LOOKUP_IN_TABLE** to perform indirect indexing is significantly faster than using a conventionally allocated table when:

- The content of the table never or rarely changes.
- The table is relatively small. Specifically, it must consume less memory than is available on a processing element (vector unit or node, depending on the execution model). The function **CMF_AVAILABLE_MEMORY** returns the amount of memory left in each processing element in units of bytes.

When these constraints are met, **CMF_LOOKUP_IN_TABLE** stores a copy of the table into each processing element. This allows the subroutine to do local memory references into the local copy of the table using indirect addressing hardware.

Also, **CMF_LOOKUP_IN_TABLE** is substantially faster when:

- the **MASK** argument has a value of **.TRUE.**
- the product of the dimensions of **INDEX** is an integer multiple of the number of nodes or processors available to the program. The number of processing elements is returned by the function **CMF_NUMBER_OF_PROCESSORS**.

Restriction

- The CM Fortran utility library procedures do not operate on arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank but with offsets for the individual axes.

See Also

CMF_ALLOCATE_TABLE
CMF_AVAILABLE_MEMORY
CMF_DEALLOCATE_TABLE

Copyright © 1994 Thinking Machines Corporation

CMF_DEALLOCATE_TABLE

Deallocates all storage associated with a lookup table allocated by **CMF_ALLOCATE_TABLE**.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_DEALLOCATE_TABLE( TABLE )
```

Arguments

TABLE	An integer scalar that identifies the table to be deallocated. Only tables allocated by the **CMF_ALLOCATE_TABLE** subroutine can be deallocated by this procedure.

Result

None.

See Also

CMF_ALLOCATE_TABLE
CMF_LOOKUP_IN_TABLE

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

6.6 Gathers/Scatters on Serial Axes

Two subroutines transfer array-indexed values between two CM arrays. Under the conditions noted below, these procedures use the special indirect addressing hardware for local transfers.

```
CALL CMF_AREF_1D( DEST, ARRAY, INDEX, MASK )

CALL CMF_ASET_1D( ARRAY, SOURCE, INDEX, MASK )
```

The **ARRAY** argument can be multidimensional. The "**1D**" in the procedure names refers to the fact that the indirect addressing occurs only on a single axis.

CMF_AREF_1D extracts array-indexed values from the serial axis of **ARRAY**. **INDEX** is an **INTEGER** array of the same shape and layout as **DEST**. Each element of **INDEX** provides an index into **ARRAY** for the value to be stored in the corresponding element of **DEST**.

CMF_ASET_1D performs the opposite operation. **INDEX** is an **INTEGER** array of the same shape and layout as **SOURCE**. In this operation, each element of **INDEX** specifies the location in **ARRAY** at which to store the corresponding element of **SOURCE**.

6.6.1 Language Comparison

FORALL also generates fast indirect addressing on serial axes (often faster than the utilities), as in these examples:

```
FORALL(I=1:8192, MASK(I)) DEST(I) = ARRAY(INDEX(I),I)

FORALL(I=1:8192, MASK(I)) ARRAY(INDEX(I),I) = SOURCE(I)
```

Also, with **FORALL**, the serial axis need not be the first axis, but can be in any position.

Copyright © 1994 Thinking Machines Corporation

6.6.2 Conditions for Fast Performance of Utility Procedures

These subroutines use the fast indirect addressing hardware when the **ARRAY** argument meets the following conditions:

- Its first dimension must be serially ordered (that is, local to a processing element).
- It must have one more dimension than the **INDEX**, **MASK**, and **DEST** arrays.
- Excluding its first axis, its remaining axes must have the same shape and layout as the **INDEX**, **MASK**, and **DEST** arrays.

In addition, these subroutines are substantially faster when:

- the **MASK** argument is the scalar **.TRUE.**.
- the product of the dimensions of the **INDEX** argument is an integer multiple of the number of processing elements executing the program. (This number is returned by the function **CMF_NUMBER_OF_PROCESSORS**.)

A restriction that affects the performance of these subroutines is:

- The serial dimension of **ARRAY** must fit into the memory of a single processing element. The size restriction by CM Fortran execution model is:
  - CM-5 vector-units model: Serial dimension extent is limited by the amount of memory on a vector unit. (Note that this applies to the global vector-unit execution model, the global/local model, and the nodal model.)
  - CM-5 SPARC nodes model: Serial dimension extent is limited by the amount of memory on a SPARC node.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

6.6.3 Gather/Scatter Examples

This call to **CMF_AREF_1D** is functionally equivalent to the **DO** loop shown:

```
      INTEGER I
      INTEGER DEST(8192), ARRAY(10,8192), INDEX(8192)
CMF$  LAYOUT ARRAY(:SERIAL, :NEWS)
      LOGICAL MASK(8192)
      ...
      DO I=1,8192
          IF (MASK(I)) DEST(I) = ARRAY(INDEX(I),I)
      END DO
      ...
      CALL CMF_AREF_1D(DEST, ARRAY, INDEX, MASK)
      ...
```

This call to **CMF_ASET_1D** is functionally equivalent to the **DO** loop shown:

```
      INTEGER I
      INTEGER SOURCE(8192), ARRAY(10,8192), INDEX(8192)
CMF$  LAYOUT ARRAY(:SERIAL, :NEWS)
      LOGICAL MASK(8192)
      ...
      DO I=1,8192
          IF (MASK(I)) ARRAY(INDEX(I),I) = SOURCE(I)
      END DO
      ...
      CALL CMF_ASET_1D(ARRAY, SOURCE, INDEX, MASK)
      ...
```


Copyright © 1994 Thinking Machines Corporation

CMF_AREF_1D

Extracts array-indexed values from the serial axis of a CM array.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_AREF_1D( DEST, ARRAY, INDEX, MASK )
```

Arguments

DEST	A CM array of the same type as **ARRAY** and conforming to **INDEX** and **MASK**. Values referenced from **ARRAY** are stored in **DEST**.
ARRAY	The CM array to be referenced. **ARRAY** must be of the same type as **DEST** and have one more dimension than **DEST**, **INDEX**, and **MASK**. The first axis must have **:SERIAL** ordering and all axes after the first must have the same shape and layout as the other arguments.
INDEX	An integer CM array conformable with **DEST** and **MASK**. These values are used as indices into the **:SERIAL** axis of **ARRAY**.
MASK	A logical CM array conforming to **DEST** and **INDEX**, or the scalar value **.TRUE.**. If **MASK** is the scalar value **.TRUE.**, all the elements of **DEST** are modified. If **MASK** is a logical array, only the elements of **DEST** corresponding to the elements of **MASK** that contain **.TRUE.** are modified.

Result

None.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Description

This subroutine places into selected elements of **DEST** the value from the first axis of **ARRAY** referenced by the corresponding elements of **INDEX**. The elements selected are those that correspond to a **.TRUE.** element in **MASK**. Note that even though **CMF_AREF_1D** operates only on the first axis of **ARRAY**, **ARRAY** must have a rank greater than one.

CMF_AREF_1D uses indirect addressing hardware to perform this reference when the following conditions are met:

- **MASK** is the scalar value **.TRUE.**.
- The product of the dimensions of **INDEX** is an integer multiple of the number of processing elements available to the program. (The number of processing elements is returned by the function **CMF_NUMBER_OF_PROCESSORS**.)

Restrictions

- This routine assumes that the arrays have a lower bound of 1. All other lower bound values are ignored.
- The CM Fortran utility library procedures do not support arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank with nonzero axis offsets.

Copyright © 1994 Thinking Machines Corporation

CMF_ASET_1D

Stores values into the serial axis of a CM array at array-indexed locations.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_ASET_1D( ARRAY, SOURCE, INDEX, MASK )
```

Arguments

ARRAY	A CM array of the same type as **SOURCE**. **ARRAY** must have one more dimension than **SOURCE**, **INDEX**, and **MASK**. The first axis must have **:SERIAL** ordering, and all axes after the first must have the same shape and layout as the other arguments. Values referenced from **SOURCE** are stored in **ARRAY**.
SOURCE	A CM array of the same type as **ARRAY** and the same shape and layout as **INDEX** and **MASK**.
INDEX	An integer CM array of the same shape and layout as **SOURCE** and **MASK**. These values are used as indices into the **:SERIAL** axis of **ARRAY**, specifying the location at which to store the corresponding value of **SOURCE**.
MASK	A logical CM array of the same shape and layout as **SOURCE** and **INDEX**, or the scalar value **.TRUE.**. If **MASK** is the scalar value **.TRUE.**, all the elements of **SOURCE** are stored. If **MASK** is a logical array, only the elements of **SOURCE** corresponding to the elements of **MASK** that contain **.TRUE.** are stored.

Result

None.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Description

This subroutine stores selected elements of **SOURCE** into the first (serial) axis of **ARRAY** at the locations specified by the corresponding elements of **INDEX**. The elements selected are those that correspond to a **.TRUE.** element in **MASK**. Although **CMF_ASET_1D** operates on only a single axis, **ARRAY** must have a rank greater than one.

CMF_ASET_1D uses indirect addressing hardware to perform this reference when the following conditions are met:

- **MASK** is the scalar value **.TRUE.**.
- The product of the dimensions of **INDEX** is an integer multiple of the number of nodes or processors available to the program. (The number of processing elements is returned by the function **CMF_NUMBER_OF_PROCESSORS**.)

Restrictions

- The CM Fortran utility library procedures do not support arrays that have been aligned with other arrays of greater rank or with other arrays of the same rank with nonzero axis offsets.
- This routine assumes that the arrays have a lower bound of 1. All other lower bound values are ignored.

Copyright © 1994 Thinking Machines Corporation