Chapter 9

Global Communication

CMMD provides functions for several types of global communication:

- broadcast operations:
  `CMMD_bc_to_nodes`
  `CMMD_receive_bc_from_node`
- reduce:
  `CMMD_reduce_`*`type`*
  `CMMD_reduce_`*`type`*`_start`
  `CMMD_reduce_done`
  `CMMD_reduce_`*`type`*`_finish`
  `CMMD_reduce_v`
- scans (parallel prefix):
  `CMMD_scan_`*`type`*
  `CMMD_scan_`*`type`*`_start`
  `CMMD_scan_done`
  `CMMD_scan_`*`type`*`_finish`
  `CMMD_scan_v`
- concatenation:
  `CMMD_concat_with_nodes`

CMMD global functions take advantage of the CM-5's hardware support for global communications.

Users should note some important facts about these operations:

- Global functions require that all nodes in the partition execute the same operation simultaneously. If any node fails to execute the appropriate global operation, the remaining nodes are unable to proceed. (Chapter 18 describes a method for defining a logical partition that contains only a subset of the nodes in the physical partition, should this be required.)

Copyright © 1995 Thinking Machines Corporation

- At present, global operations do not support parallel array arguments. The array arguments to a global function must be either serial arrays or vector serial arrays.
- CMMD expects typed data, such as that specified for global functions, to be aligned on its natural data-type boundaries. In particular, double-precision data must be doubleword-aligned.
- If interrupts are enabled, then asynchronous operations (arrival of messages from the network) will interrupt global operations.

  If interrupts are not enabled, message-passing calls involving a node that has called a global operation may not complete until after the global operation is complete. This should not be a problem unless the operation's uncompletedness prevents some other node from reaching the global operation. In that case, both operations will hang.

9.1 Broadcasts

In a broadcast, a message is sent from a single source to all nodes. All nodes receive the identical message, and all receive the entire message. For a broadcast to occur, all nodes in a partition must be synchronized.

CMMD offers several facilities for broadcasting messages:

- One node can send data to all other nodes, via a broadcast message.
- Data read from a file or I/O device can be broadcast to all nodes. (See Chapter 15.)
- In host/node programs, the host can broadcast data to all nodes. (Host/node programs are described in Chapter 19.)

This chapter discusses broadcasts in which one node sends a message to all other nodes. This is accomplished by a pair of functions: `CMMD_bc_to_nodes`, by which one node signals its intent to send a broadcast message, and `CMMD_receive_bc_from_nodes`, by which all other nodes signal their readiness to receive the broadcast. Section 9.4 provides reference pages for these functions.

Copyright © 1995 Thinking Machines Corporation

Broadcasts are synchronous, blocking operations. Once a broadcast is signaled, the hardware begins checking for responses. Only when the hardware signals that the entire broadcast is complete can any of the broadcast calls return.

Another way of saying this is that only when all nodes in the partition have made their broadcast call can the broadcast occur.

PLEASE NOTE

1. In every broadcast, all nodes receive data simultaneously, and all receive the same amount of data. For this reason, it is important to ensure that all nodes have sufficient buffer space to hold the broadcast message.

2. Asynchronous nonglobal operations (arrival of messages from the network, etc.) will interrupt broadcasts, as well as other global operations. This generally will not interfere with the global operation in progress.

9.2 Scans, Reduces, and Concatenation

Scans, reduces, and concatenation are global operations. All nodes in the partition take part in these calls: each node provides source data, and each receives the result of the global operation.

Reduces and scans perform arithmetic or logical operations. Given a buffer containing some value (or vector of values) on each node, these global computations operate cumulatively on the buffer set to perform such tasks as

- summing the value across all the nodes
- finding the largest or smallest value
- performing a bitwise AND, OR, or XOR

Copyright © 1995 Thinking Machines Corporation

Reduce operations return the same value to all nodes. Scans return the cumulative results as a running tally across all nodes.

Reduce operations, scans, and segmented scans thus provide three distinct methods of all-to-all communication. See Figure 16.

Section 9.5 provides reference pages for reduce operations.

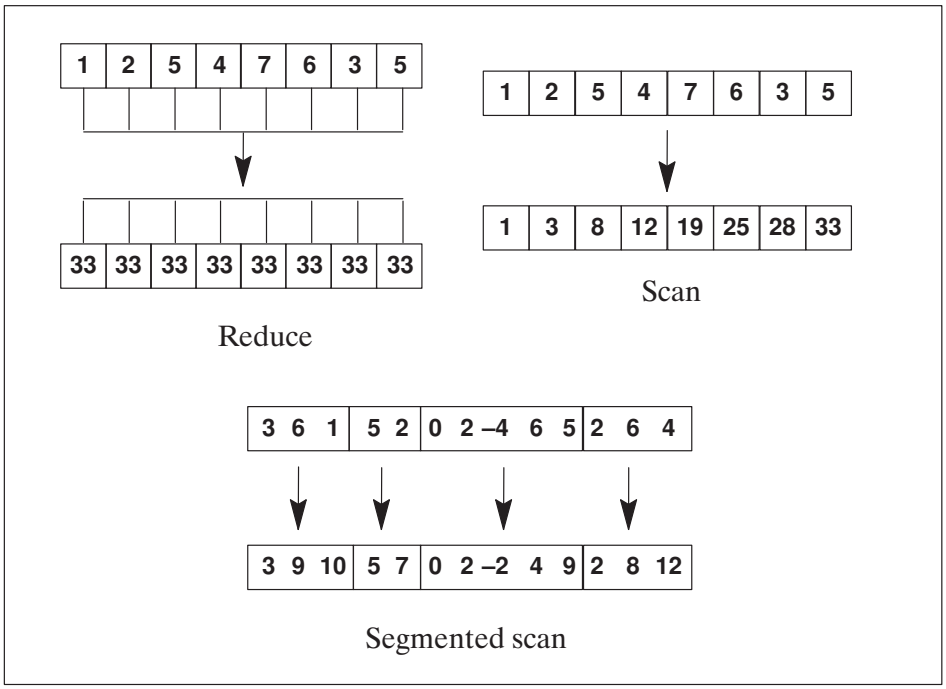

Figure 16. Reduces and scans with an addition operator.

Scan and reduce functions may perform one of a number of operations:

`add`	Summation
`max`	Return the largest value found
`min`	Return the smallest value found
`ior`	Bitwise inclusive OR operation
`xor`	Bitwise exclusive OR operation
`and`	Bitwise logical AND operation

The operation to be performed by a particular function is specified as an argument to the function. For example, the `CMMD_reduce_int` function,

Copyright © 1995 Thinking Machines Corporation

called with the `CMMD_combiner_max` argument, would compare integer values on all nodes and return the largest integer to all nodes.

Reduces

A *reduce* operation starts with values in every processor and ends with a single value in every node. Values may be added, so that the sum of all values is returned, or the largest or smallest value may be chosen, or an OR or XOR may be done across all the values. In each case, the same result is returned to all nodes.

Thus, on four nodes holding these values:

```
4    9    7   6
```

a reduce/add would return

```
26   26   26   26
```

A vector reduce operation starts with a vector of values on each node, and creates (on each node) a new vector in which each specified source element has been replaced by the result of the reduce across all nodes. The new vectors may be in a new location or may overwrite the source vectors.

Thus, if four nodes hold the vectors

```
[1 2 3 4]    [2 3 4 5]    [3 4 5 6]    [4 5 6 7]
```

a reduce/add without striding would create the vectors

```
[10 14 18 22] [10 14 18 22] [10 14 18 22] [10 14 18 22]
```

while a reduce/add with a stride double the element length, overwriting its source buffer, would create the vectors

```
[10 2 18 4]   [10 3 18 5]   [10 4 18 6]   [10 5 18 7]
```

Scans

A *scan* (sometimes called a *parallel prefix* operation) moves from processor to processor, in processor identifier order, creating a running tally of results in each processor. The function call specifies whether the scan proceeds upward (0 to *N*) or downward (*N* to 0), and whether the scan is to be inclusive

Copyright © 1995 Thinking Machines Corporation

or exclusive. (In an inclusive scan, the source value contained in any given node *N* contributes to the result for node *N*; in an exclusive scan, it does not.)

Thus, with our same four values,

```
 4    9    7    6
```

an upward exclusive scan/add would produce

```
 0    4   13   20
```

while a downward inclusive scan/add would produce

```
26   22   13    6
```

Vector scans scan across the nodes for each specified element of the vector. Given the two-element vectors

```
[0 1]   [2 3]   [4 5]   [6 7]
```

an upward inclusive scan/add would produce

```
[0 1]   [2 4]   [6 9]   [12 16]
```

Segmented Scans

In a *segmented scan*, independent scans are run simultaneously on different subgroups (or *segments*) of the nodes. The segments are determined at run time by arguments called the `sbit` and the `smode` (described later in this chapter). For example, given our four values

```
         4    9    7    6
```

and `sbit` values of

```
         1    0    1    0
```

an upward inclusive segmented scan/add would return

```
         4   13    7   13
```

Segmented vector scans follow the same rules as segmented nonvector scans.

Section 9.6 provides reference pages for CMMD scan functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

Concatenation

The concatenation operation takes the contents of all nodes' source buffers and appends them to each other, in processor order, placing the result in each node's destination buffer. Concatenation always proceeds from the lowest to the highest node, and is never segmented.

For example, given the four values

```
a     b     c     d
```

concatenation would produce

```
abcd     abcd     abcd     abcd
```

Section 9.7 provides the reference pages for the concatenation function.

9.3 Host/Node Global Functions

CMMD also supplies host/node versions of many of these global functions; they are presented in Chapter 19.

9.4 Broadcast Functions

These broadcast functions allow one node to send an identical message to all other nodes in the partition.

All nodes must take part in these broadcasts. One node must signal its intent to send the broadcast message; all other nodes must signal their readiness to receive the message.

Note that, in every broadcast, all nodes receive data simultaneously, and all receive the same amount of data. For this reason, it is important to ensure that all nodes have sufficient buffer space to hold the broadcast message.

A broadcast is a blocking operation. Once a broadcast is signaled, the hardware begins checking for responses. Only when the hardware signals that the entire broadcast is complete can any of the broadcast calls return.

NOTE: Asynchronous operations (arrival of messages from the Data Network, etc.) will interrupt broadcasts, as well as other global operations.

Copyright © 1995 Thinking Machines Corporation

CMMD_bc_to_nodes
CMMD_receive_bc_from_node

`CMMD_bc_to_nodes` is called by a *single* node to send a message to all other nodes. `CMMD_receive_bc_from_node` is called by *all other* nodes to receive a broadcast message from the node that called.

SYNOPSIS

C Syntax

```
void  CMMD_bc_to_nodes(void *buffer, int buffer_desc)
void  CMMD_receive_bc_from_node(void *buffer, int buffer_desc)
```

Fortran Syntax

```
SUBROUTINE  CMMD_bc_to_nodes(buffer, buffer_desc)
   <ARRAY>   buffer
   INTEGER   buffer_desc

SUBROUTINE  CMMD_receive_bc_from_node(buffer, buffer_desc)
   <ARRAY>   buffer
   INTEGER   buffer_desc
```

ARGUMENTS

buffer — The buffer that holds the message being broadcast (from the sending node) and received (on all other nodes).

buffer_desc — The length, in bytes, of the buffer being broadcast (from the sending node) and received (on all other nodes).

RETURNS

The broadcast functions do not return until the broadcast is complete, that is, until all the nodes have made their calls and the data transfer has taken place. There is no return value.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

DESCRIPTION

Any node can call `CMMD_bc_to_node` to broadcast a buffer of the specified length (in bytes) to all other nodes. All other nodes must then call `CMMD_receive_bc_from_node`, with the same length argument, to receive the buffer. At the completion of the operation, all nodes contain identical information in the specified buffers.

IMPORTANT

1. There is no protection against having more than one broadcaster. The result of multiple broadcasts is undefined.
2. If length arguments are not identical across all nodes, a segmentation fault may result.
3. All processors within the partition must take part in this operation. Otherwise, the operation will fail and the program may hang.
4. Broadcasts cannot call or be called from callback or handler functions.

Copyright © 1995 Thinking Machines Corporation

9.5 Reduce Functions

There are three sorts of reduce functions:

- `CMMD_reduce_`*`type`* provides a set of *blocking functions* (one per data type) that operate on block serial arrays.
- `CMMD_reduce_`*`type`*`_start`, `CMMD_reduce_done`, and `CMMD_reduce_`*`type`*`_finish` provide the same operation as a partly asynchronous *split reduce.*
- `CMMD_reduce_v` is a blocking function that operates on vector serial arrays. The combiner specifies the data type for this function.

Note that the nonvector reduce function returns a value. The vector reduce functions write their return values into a buffer supplied as an argument to the function; they do not return a value.

For all these functions, single-precision reduces produce double-precision results. For all other cases, the result data type matches the input data type.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_reduce_*type*

Performs the specified reduce across all nodes.

SYNOPSIS

C Syntax

```
int CMMD_reduce_int(int value, CMMD_combiner_t combiner)
uint CMMD_reduce_uint(uint value, CMMD_combiner_t combiner)
lint CMMD_reduce_lint(int value, CMMD_combiner_t combiner)
luint CMMD_reduce_luint(luint value, CMMD_combiner_t combiner)
double CMMD_reduce_float(float value, CMMD_combiner_t combiner)
double CMMD_reduce_double(double value,
         CMMD_combiner_t combiner)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_reduce_int(value, combiner)
    INTEGER   value
    INTEGER   combiner

LONG INTEGER FUNCTION CMMD_reduce_int(value, combiner)
    LONG INTEGER     value
    INTEGER          combiner

REAL*8 FUNCTION CMMD_reduce_float(value, combiner)
    REAL*4    value
    INTEGER   combiner

REAL*8 FUNCTION CMMD_reduce_double(value, combiner)
    REAL*8    value
    INTEGER combiner
```

ARGUMENTS

value	The value to be contributed to the operation. Its type must match that specified by the function name.
combiner	One of the symbols listed below, specifying the type of operation to be performed.

Copyright © 1995 Thinking Machines Corporation

For signed integer operands, allowable combiners are

```
CMMD_combiner_add CMMD_combiner_ior
CMMD_combiner_max CMMD_combiner_xor
CMMD_combiner_min CMMD_combiner_and
```

For unsigned integer operands, allowable combiners are

```
CMMD_combiner_uadd CMMD_combiner_ior
CMMD_combiner_umax CMMD_combiner_xor
CMMD_combiner_umin CMMD_combiner_and
```

For signed long integer operands, allowable combiners are

```
CMMD_combiner_ladd CMMD_combiner_ior
CMMD_combiner_lmax CMMD_combiner_xor
CMMD_combiner_lmin CMMD_combiner_and
```

For unsigned integer operands, allowable combiners are

```
CMMD_combiner_luadd CMMD_combiner_ior
CMMD_combiner_lumax CMMD_combiner_xor
CMMD_combiner_lumin CMMD_combiner_and
```

For float operands, allowable combiners (see note, below) are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands, allowable combiners (see note, below) are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

NOTE: For backward compatibility, float and double operands may also use the combiners `CMMD_combiner_add`, `CMMD_combiner_max`, and `CMMD_combiner_min`. The use of these combiners for such operands is deprecated, however, and its continued support is not ensured.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

RETURNS

The reduce functions return the value of the specified reduce operation over all the nodes. Every node thus receives the same return value. The functions return only if all nodes have called `CMMD_reduce_`*`type`*.

Callback and Handler Functions. The global functions cannot call or be called from callback or handler functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_reduce_*type*_start

Initiates a reduce, but does not block pending completion.

SYNOPSIS

C Syntax

```
void CMMD_reduce_int_start(int value, CMMD_combiner_t combiner)
void CMMD_reduce_uint_start(unsigned value,
        CMMD_combiner_t combiner)
void CMMD_reduce_lint_start(int value,
        CMMD_combiner_t combiner)
void CMMD_reduce_luint_start(unsigned value,
        CMMD_combiner_t combiner)
void CMMD_reduce_float_start(float value,
        CMMD_combiner_t combiner)
void CMMD_reduce_double_start(double value,
        CMMD_combiner_t combiner)
```

Fortran Syntax

```
SUBROUTINE CMMD_reduce_int_start(value, combiner)
    INTEGER   value
    INTEGER   combiner

SUBROUTINE CMMD_reduce_float_start(value, combiner)
    REAL*4    value
    INTEGER   combiner

SUBROUTINE CMMD_reduce_double_start(value, combiner)
    REAL*8    value
    INTEGER   combiner
```

ARGUMENTS

value	The value to be contributed to the operation. Its type must match that specified by the function name.
combiner	One of the symbols listed below, specifying the type of operation to be performed.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

For signed integer operands, allowable combiners are

```
CMMD_combiner_add     CMMD_combiner_ior
CMMD_combiner_max     CMMD_combiner_xor
CMMD_combiner_min     CMMD_combiner_and
```

For unsigned integer operands, allowable combiners are

```
CMMD_combiner_uadd    CMMD_combiner_ior
CMMD_combiner_umax    CMMD_combiner_xor
CMMD_combiner_umin    CMMD_combiner_and
```

For signed long integer operands, allowable combiners are

```
CMMD_combiner_ladd    CMMD_combiner_ior
CMMD_combiner_lmax    CMMD_combiner_xor
CMMD_combiner_lmin    CMMD_combiner_and
```

For unsigned long integer operands, allowable combiners are

```
CMMD_combiner_luadd   CMMD_combiner_ior
CMMD_combiner_lumax   CMMD_combiner_xor
CMMD_combiner_lumin   CMMD_combiner_and
```

For float operands, allowable combiners (see note, below) are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands, allowable combiners (see note, below) are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

NOTE: For back compatibility, float and double operands may also use the combiners `CMMD_combiner_add`, `CMMD_combiner_max`, and `CMMD_combiner_min`. The use of these combiners for such operands is deprecated, however, and its continued support is not ensured.

Copyright © 1995 Thinking Machines Corporation

RETURNS

`CMMD_reduce_`*`type`*`_start` has no return value. It is an asynchronous operation, and returns immediately.

`CMMD_reduce_done` returns 1 if all nodes have called `CMMD_reduce_`*`type`*`_start`. It returns 0 otherwise.

`CMMD_reduce_`*`type`*`_finish` returns the value of the specified reduce operation over all the nodes. Every node thus receives the same return value. This function does not return until all nodes have called `CMMD_reduce_`*`type`*`_finish`.

DESCRIPTION

The mechanics of a reduce are as follows: First, each node sends its contribution to the reduce into the Control Network. Second, the Control Network computes the results of the reduce. Finally, each node receives the results of the reduce from the Control Network. The three functions involved in a split reduce cover these three activities.

`CMMD_reduce_`*`type`*`_start` sends the calling node's contribution to a global reduce operation. It does not block; rather, it returns immediately after sending the data.

`CMMD_reduce_done` tells whether the Control Network has received data from all nodes and has performed the reduce operation. It returns 1 if the reduce is complete, 0 otherwise.

`CMMD_reduce_`*`type`*`_finish` moves the reduce results from the Control Network to the nodes. Thus, it returns the value of the specified reduce operation over all the nodes.

Callback and Handler Functions. The global functions cannot be called from callback or handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_reduce_done

Tells whether the result of a reduce operation is available.

SYNOPSIS

C Syntax

```
int CMMD_reduce_done(void)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_reduce_done()
```

RETURNS

`CMMD_reduce_done` returns 1 only if all nodes have called `CMMD_reduce_`*`type`*`_start`. It returns 0 otherwise.

DESCRIPTION

The mechanics of a reduce are as follows: First, each node sends its contribution to the reduce into the Control Network; second, the Control Network computes the results of the reduce; finally, each node receives the results of the reduce from the Control Network. The three functions involved in a split reduce cover these three activities.

`CMMD_reduce_`*`type`*`_start` sends the calling node's contribution to a global reduce operation. It does not block; rather, it returns immediately after sending the data.

`CMMD_reduce_done` reports whether the Control Network has received data from all nodes and has performed the reduce operation. It returns 1 if the reduce is complete, 0 otherwise.

`CMMD_reduce_`*`type`*`_finish` moves the reduce results from the Control Network to the nodes. Thus, it returns the value of the specified reduce operation over all the nodes.

Copyright © 1995 Thinking Machines Corporation

Callback and Handler Functions. The global functions cannot be called from callback or handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_reduce_*type*_finish

Blocks until the reduce finishes, and returns its value.

SYNOPSIS

C Syntax

```
int CMMD_reduce_int_finish(int value, CMMD_combiner_t combiner)
uint CMMD_reduce_uint_finish(unsigned value,
        CMMD_combiner_t combiner)
lint CMMD_reduce_lint_finish(int value,
        CMMD_combiner_t combiner)
luint CMMD_reduce_luint_finish(unsigned value,
        CMMD_combiner_t combiner)
double CMMD_reduce_float_finish(float value,
        CMMD_combiner_t combiner)
double CMMD_reduce_double_finish(double value,
        CMMD_combiner_t combiner)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_reduce_int_finish(value, combiner)
    INTEGER   value
    INTEGER   combiner

REAL*8 FUNCTION CMMD_reduce_float_finish(value, combiner)
    REAL*4    value
    INTEGER   combiner

REAL*8 FUNCTION CMMD_reduce_double_finish(value, combiner)
    REAL*8    value
    INTEGER   combiner
```

ARGUMENTS

value	The value to be contributed to the operation. Its type must match that specified by the function name.
combiner	One of the symbols listed below, specifying the type of operation to be performed.

Copyright © 1995 Thinking Machines Corporation

For signed integer operands, allowable combiners are

```
CMMD_combiner_add     CMMD_combiner_ior
CMMD_combiner_max     CMMD_combiner_xor
CMMD_combiner_min     CMMD_combiner_and
```

For unsigned integer operands, allowable combiners are

```
CMMD_combiner_uadd    CMMD_combiner_ior
CMMD_combiner_umax    CMMD_combiner_xor
CMMD_combiner_umin    CMMD_combiner_and
```

For signed long integer operands, allowable combiners are

```
CMMD_combiner_ladd    CMMD_combiner_ior
CMMD_combiner_lmax    CMMD_combiner_xor
CMMD_combiner_lmin    CMMD_combiner_and
```

For unsigned long integer operands, allowable combiners are

```
CMMD_combiner_luadd   CMMD_combiner_ior
CMMD_combiner_lumax   CMMD_combiner_xor
CMMD_combiner_lumin   CMMD_combiner_and
```

For float operands, allowable combiners (see note, below) are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands, allowable combiners (see note, below) are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

NOTE: For back compatibility, float and double operands may also use the combiners `CMMD_combiner_add`, `CMMD_combiner_max`, and `CMMD_combiner_min`. The use of these combiners for such operands is deprecated, however, and its continued support is not ensured.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

RETURNS

`CMMD_reduce_`*`type`*`_finish` returns the value of the specified reduce operation over all the nodes. Every node thus receives the same return value. This function does not return until all nodes have called `CMMD_reduce_`*`type`*`_finish`.

DESCRIPTION

The mechanics of a reduce are as follows: first, each node sends its contribution to the reduce into the Control Network; second, the Control Network computes the results of the reduce; finally, each node receives the results of the reduce from the Control Network. The three functions involved in a split reduce cover these three activities.

`CMMD_reduce_`*`type`*`_start` sends the calling node's contribution to a global reduce operation. It does not block; rather, it returns immediately after sending the data.

`CMMD_reduce_done` reports whether the Control Network has received data from all nodes and has performed the reduce operation. It returns 1 if the reduce is complete, 0 otherwise.

`CMMD_reduce_`*`type`*`_finish` moves the reduce results from the Control Network to the nodes. Thus, it returns the value of the specified reduce operation over all the nodes.

Callback and Handler Functions. The global functions cannot be called from callback or handler functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_reduce_v

Computes the value of the specified reduce operation for each specified source buffer element over all the nodes.

SYNOPSIS

C Syntax

```
void CMMD_reduce_v(void *dest, void *source,
         CMMD_combiner_t combiner, int stride, int elem_count)
```

Fortran Syntax

```
SUBROUTINE CMMD_reduce_v(dest, source, combiner, stride,
         elem_count)
    <ARRAY>   dest, source
    INTEGER   combiner, stride, elem_count
```

ARGUMENTS

dest — The buffer that will hold the vector of reduced values. It must be a serial array, not a parallel one.

source — The local buffer that holds the vector being reduced. It must be a serial array, not a parallel one.

combiner — One of the symbols listed below, specifying the type of operation to be performed.

For signed integer operands, allowable combiners are

```
CMMD_combiner_add     CMMD_combiner_ior
CMMD_combiner_max     CMMD_combiner_xor
CMMD_combiner_min     CMMD_combiner_and
```

For unsigned integer operands, allowable combiners are

```
CMMD_combiner_uadd    CMMD_combiner_ior
CMMD_combiner_umax    CMMD_combiner_xor
CMMD_combiner_umin    CMMD_combiner_and
```

For signed long integer operands, allowable combiners are

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

```
CMMD_combiner_ladd    CMMD_combiner_ior
CMMD_combiner_lmax    CMMD_combiner_xor
CMMD_combiner_lmin    CMMD_combiner_and
```

For unsigned long integer operands, allowable combiners are

```
CMMD_combiner_luadd   CMMD_combiner_ior
CMMD_combiner_lumax   CMMD_combiner_xor
CMMD_combiner_lumin   CMMD_combiner_and
```

For float operands, allowable combiners are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands, allowable combiners are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

stride An integer specifying the distance in bytes between the starting addresses of vector elements. Must be a positive integer multiple of the relevant data type.

elem_count A positive integer specifying the number of elements in the vector.

RETURNS

This function does not return until all nodes have called it, and it does not return a value.

DESCRIPTION

The vector reduce function computes the value of the specified reduce operation for each specified source buffer element over all the nodes. Every node thus computes the same set of values. The function does not return until all nodes have called `CMMD_reduce_v`.

Copyright © 1995 Thinking Machines Corporation

The vector reduce functions do not return values. Rather, each value computed from a *source* buffer element is stored in the corresponding *dest* buffer element. The *source* and *dest* arguments may be identical, in which case the results overwrite the *source* buffer. If *source* and *dest* overlap in any other way, the results are undefined.

Note that the data type for vector reduces is not specified by function name, but only by the combiner.

If *elem_count* is 1, you may mix CMMD_reduce_*type* operations with vector operations; If *elem_count* is more than one, however, mixing vector and nonvector operations will cause the system to hang.

An Example. As an example of vector reduce, let us consider the case of a 4 x 6 matrix A, filled with unsigned integers from 0 to 23, laid out in memory as follows:

```
          0  1  2  3  4  5
          6  7  8  9 10 11
A =      12 13 14 15 16 17
         18 19 20 21 22 23
```

To sum the first column of the matrix across all nodes, you would do a vector reduce with combiner CMMD_combiner_uadd, stating that you were reducing four elements (*elem_count*), each located six times the number of bytes in a *uint* apart (*stride*):

```
CMMD_reduce_v
     (&A, &A, CMMD_combiner_uadd, 6*sizeof(uint), 4)
```

If there were *P* processing nodes in the partition and array A contained the same data in each, then the first element in the second row of A would be replaced by 6**P* in every node, the first element in the third row would be replaced by 12**P*, and the first element in the last row would be replaced by 18**P*. All other elements would remain unchanged.

Callback and Handler Functions. The global functions cannot be called from callback or handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

9.6 Scan Functions

CMMD provides three sorts of scan functions:

- `CMMD_scan_`*`type`* offers a set of *blocking functions* (one per data type) that operate on block serial arrays.
- `CMMD_scan_`*`type`*`_start`, `CMMD_scan_done`, and `CMMD_scan_`*`type`*`_finish` provide the same operation as a partly asynchronous *split scan*.
- `CMMD_scan_v` is a blocking function that operates on vector serial arrays. The combiner specifies the data type for this function.

Note that the nonvector scan function returns a value. The vector scan functions write their return values into a buffer supplied as an argument to the function; they do not return a value.

For all these functions, single-precision scans produce double-precision results. For all other cases, the result data type matches the input data type.

Copyright © 1995 Thinking Machines Corporation

CMMD_scan_*type*

Performs the specified scan on data in source across all nodes, stores result in dest.

SYNOPSIS

C Syntax

```
int CMMD_scan_int(int value, CMMD_combiner_t combiner,
          CMMD_scan_direction_t direction,
          CMMD_segment_mode_t smode, int sbit,
          CMMD_scan_inclusion_t inclusion)

uint CMMD_scan_uint(uint value,
          CMMD_combiner_t combiner,
          CMMD_scan_direction_t direction,
          CMMD_segment_mode_t smode, int sbit,
          CMMD_scan_inclusion_t inclusion)

lint CMMD_scan_lint(lint value,
          CMMD_combiner_t combiner,
          CMMD_scan_direction_t direction,
          CMMD_segment_mode_t smode, int sbit,
          CMMD_scan_inclusion_t inclusion)

luint CMMD_scan_luint(luint value,
          CMMD_combiner_t combiner,
          CMMD_scan_direction_t direction,
          CMMD_segment_mode_t smode, int sbit,
          CMMD_scan_inclusion_t inclusion)

double CMMD_scan_float(float value,
          CMMD_combiner_t combiner,
          CMMD_scan_direction_t direction,
          CMMD_segment_mode_t smode, int sbit,
          CMMD_scan_inclusion_t inclusion)

double CMMD_scan_double(double value,
          CMMD_combiner_t combiner,
          CMMD_scan_direction_t direction,
          CMMD_segment_mode_t smode, int sbit,
          CMMD_scan_inclusion_t inclusion)
```


Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

Fortran Syntax

```
INTEGER FUNCTION CMMD_scan_int(value, combiner, direction,
         smode, sbit, inclusion)
    INTEGER   value
    INTEGER   combiner, direction, smode, sbit, inclusion

LONG INTEGER FUNCTION CMMD_scan_lint(value, combiner,
         direction, smode, sbit, inclusion)
    INTEGER   value
    INTEGER   combiner, direction, smode, sbit, inclusion

REAL*8 FUNCTION CMMD_scan_float(value, combiner, direction,
         smode, sbit, inclusion)
    REAL*4    value
    INTEGER   combiner, direction, smode, sbit, inclusion

REAL*8 FUNCTION CMMD_scan_double(value, combiner,
         direction, smode, sbit, inclusion)
    REAL*8    value
    INTEGER   combiner, direction, smode, sbit, inclusion
```

ARGUMENTS

value — The value to be contributed by this processor to the operation. Its type must match that specified by the function name.

combiner — One of the symbols listed below, specifying the type of operation to be performed.

For signed integer operands, allowable combiners are

CMMD_combiner_add	CMMD_combiner_ior
CMMD_combiner_max	CMMD_combiner_xor
CMMD_combiner_min	CMMD_combiner_and

For unsigned integer operands, allowable combiners are

CMMD_combiner_uadd	CMMD_combiner_ior
CMMD_combiner_umax	CMMD_combiner_xor
CMMD_combiner_umin	CMMD_combiner_and

Copyright © 1995 Thinking Machines Corporation

For long signed integer operands, allowable combiners are

```
CMMD_combiner_ladd    CMMD_combiner_ior
CMMD_combiner_lmax    CMMD_combiner_xor
CMMD_combiner_lmin    CMMD_combiner_and
```

For long unsigned integer operands, allowable combiners are

```
CMMD_combiner_luadd   CMMD_combiner_ior
CMMD_combiner_lumax   CMMD_combiner_xor
CMMD_combiner_lumin   CMMD_combiner_and
```

For float operands (see note, below), allowable combiners are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands (see note, below), allowable combiners are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

NOTE: For back compatibility, float and double operands may also use the combiners `CMMD_combiner_add`, `CMMD_combiner_max`, and `CMMD_combiner_min`. The use of these combiners for such operands is deprecated, however, and its continued support is not ensured.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

direction	CMMD_upward	The scan starts at node 0 and proceeds to the highest-numbered node.
	CMMD_downward	The scan starts at the highest-numbered node and proceeds downward to node 0.
smode	CMMD_none	The scan proceeds across all nodes.
	CMMD_segment_bit	The scan is a segmented scan, with the *sbit* acting as a segment bit.
	CMMD_start_bit	The scan is a segmented scan, with the *sbit* acting as a start bit.
sbit	If *sbit* is nonzero, the node marks the boundary (usually the beginning) of a segment; if *sbit* is 0 the node is not a boundary marker. (If *smode* is set to CMMD_none, *sbit* is ignored.)	
inclusion	CMMD_inclusive	The scan is inclusive.
	CMMD_exclusive	The scan is exclusive.

RETURNS

CMMD_scan_*type* returns the value of the specified scan operation over all the nodes. Single-precision scans produce double-precision results. For all other cases, the result data type matches the input data type.

This function does not return until all nodes have called it.

DESCRIPTION

CMMD_scan_*type* returns the value of the specified scan operation over all the nodes.

Copyright © 1995 Thinking Machines Corporation

PLEASE NOTE

1. Values for `direction`, `smode`, and `inclusion` *must* be identical across all nodes. Otherwise, results are unpredictable and the program may crash.

2. For `CMMD_scan_float` and `CMMD_scan_double`, the combination of `smode` = `CMMD_start_bit` and `inclusion` = `CMMD_exclusive` is currently illegal and causes the nodes to exit.

***direction* and *inclusion*.** The `direction` argument determines the direction of the scan, either upward (from 0 to the highest-numbered node) or downward (from the highest-numbered node to 0). The `inclusion` argument determines whether a given node contributes *its own value* to the scan result. When `smode` is `CMMD_none`, the scan result is determined as follows:

- In inclusive upward scans, the value returned for a given node N is the combination of the source values in all nodes $\leq N$.
- In inclusive downward scans, the value returned for a given node N is the combination of the source values in all nodes $\geq N$.
- In exclusive upward scans, the value returned for a given node N is the combination of source values in all processors $< N$. The first (lowest-numbered) node receives the identity element for the combiner.
- In exclusive downward scans, the value returned for a given node N is the combination of source values in all nodes $> N$. The highest-numbered node receives the identity element for the combiner.

If a scan is a segmented scan, these rules apply on a per-segment basis, as explained below.

***smode* and *sbit*.** The `smode` and `sbit` arguments define *segmented scans*. These are scans that tally their results across subgroups of the nodes. Every node belongs to one group, or *segment*, with the group to which it belongs determined by `smode` and `sbit` as follows:

***smode* = CMMD_segment_bit**: If `smode` is `CMMD_segment_bit`, then `sbit` is considered a *segment bit*. A nonzero segment bit starts a new segment for an upward

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

scan, but ends a segment for a downward scan. Imagine 8 nodes with the following segment bits:

```
0  0  1  0  0  1  0  0
```

Both upward and downward scans would have 3 segments: one would include nodes 0 and 1, another would include nodes 2−4, and a third would include nodes 5−7.

When *sbit* is a segment bit, operations in one segment never affect the values of elements in another segment. Thus, given segment bits of

```
0  0  1  0
```

and unsigned values of

```
4  1  5  2
```

an upward exclusive unsigned max would produce

```
0  4  0  5
```

***smode* = CMMD_start_bit**. If *smode* is CMMD_start_bit, then *sbit* is considered a *start bit*. A nonzero start bit always starts a new segment, whether the direction of the scan is upward or downward. Thus, given 8 nodes with the following start bits:

```
0  0  1  0  0  1  0  0
```

an upward scan would have the same segments as the segmented scan shown above (0−1, 2−4, 5−7); but a downward scan would have segments of 7−6, 5−3, and 2−0.

In addition, if the operation is exclusive, a node with a nonzero start bit does receive a value from the preceding segment. The value received is the reduce of the previous segment, with the same combiner. Thus, given start bits of

```
0  0  1  0
```

and values of

```
4  1  5  2
```

an upward exclusive unsigned max would produce

```
0  4  4  5
```


Copyright © 1995 Thinking Machines Corporation

but a downward exclusive unsigned max would produce

```
5   5   2   0
```

However, an upward inclusive unsigned max would produce

```
4   4   5   5
```

and a downward inclusive unsigned max would produce

```
5   5   5   2
```

Callback and Handler Functions. The global functions cannot be called from callback or handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_scan_*type*_start

Sends the calling node's contribution to a global scan operation; does not block pending completion, but returns immediately after sending the data.

SYNOPSIS

C Syntax

```
void CMMD_scan_int_start(int value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

void CMMD_scan_uint_start(uint value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

void CMMD_scan_lint_start(lint value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

void CMMD_scan_luint_start(luint value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

void CMMD_scan_float_start(float value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

void CMMD_scan_double_start(double value,
         CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)
```


Copyright © 1995 Thinking Machines Corporation

Fortran Syntax

```
SUBROUTINE CMMD_scan_int_start(value, combiner, direction,
          smode, sbit, inclusion)
    INTEGER   value
    INTEGER   combiner, direction, smode, sbit, inclusion

SUBROUTINE CMMD_scan_lint_start(value, combiner, direction,
          smode, sbit, inclusion)
    LONG INTEGER value
    INTEGER   combiner, direction, smode, sbit, inclusion

SUBROUTINE CMMD_scan_float_start(value, combiner, direction,
          smode, sbit, inclusion)
    REAL*4    value
    INTEGER   combiner, direction, smode, sbit, inclusion

SUBROUTINE CMMD_scan_double_start(value, combiner, direction,
          smode, sbit, inclusion)
    REAL*8 value
    INTEGER combiner, direction, smode, sbit, inclusion
```

ARGUMENTS

value — The value to be contributed by this processor to the operation. Its type must match that specified by the function name.

combiner — One of the symbols listed below, specifying the type of operation to be performed.

For signed integer operands, allowable combiners are

CMMD_combiner_add	CMMD_combiner_ior
CMMD_combiner_max	CMMD_combiner_xor
CMMD_combiner_min	CMMD_combiner_and

For unsigned integer operands, allowable combiners are

CMMD_combiner_uadd	CMMD_combiner_ior
CMMD_combiner_umax	CMMD_combiner_xor
CMMD_combiner_umin	CMMD_combiner_and

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

For signed long integer operands, allowable combiners are

```
CMMD_combiner_ladd    CMMD_combiner_ior
CMMD_combiner_lmax    CMMD_combiner_xor
CMMD_combiner_lmin    CMMD_combiner_and
```

For unsigned long integer operands, allowable combiners are

```
CMMD_combiner_luadd   CMMD_combiner_ior
CMMD_combiner_lumax   CMMD_combiner_xor
CMMD_combiner_lumin   CMMD_combiner_and
```

For float operands (see note, below), allowable combiners are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands (see note, below), allowable combiners are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

NOTE: For back compatibility, float and double operands may also use the combiners `CMMD_combiner_add`, `CMMD_combiner_max`, and `CMMD_combiner_min`. The use of these combiners for such operands is deprecated however, and its continued support is not ensured.

direction	`CMMD_upward`	The scan starts at node 0 and proceeds to the highest-numbered node.
	`CMMD_downward`	The scan starts at the highest-numbered node and proceeds downward to node 0.

Copyright © 1995 Thinking Machines Corporation

smode	CMMD_none	The scan proceeds across all nodes.
	CMMD_segment_bit	The scan is a segmented scan, with *sbit* acting as a segment bit.
	CMMD_start_bit	The scan is a segmented scan, with *sbit* acting as a start bit.
sbit	If *sbit* is nonzero, the node marks the boundary (usually the beginning) of a segment; if *sbit* is 0, the node is not a boundary marker. (If *smode* is set to CMMD_none, *sbit* is ignored.)	
inclusion	CMMD_inclusive	The scan is inclusive.
	CMMD_exclusive	The scan is exclusive.

RETURNS

CMMD_scan_*type*_start has no return value. It returns immediately.

CMMD_scan_done returns 1 if all nodes have called CMMD_scan_*type*_start, because this means that all nodes are now ready to participate in the scan operation. It returns 0 otherwise.

DESCRIPTION

The mechanics of a scan are as follows: first, each node sends its contribution to the scan into the Control Network; second, the Control Network computes the results of the scan; finally, each node receives the results of the scan from the Control Network. The three functions involved in a split scan cover these three activities.

CMMD_scan_*type*_start sends the calling node's contribution to a global scan operation. It does not block; rather, it returns immediately after sending the data.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

PLEASE NOTE

1. Values for `direction`, `smode`, and `inclusion` *must* be identical across all nodes. Otherwise, results are unpredictable and the program may crash.

2. For `CMMD_scan_float` and `CMMD_scan_double`, the combination of `smode` = `CMMD_start_bit` and `inclusion` = `CMMD_exclusive` is currently illegal and causes the nodes to exit.

***direction* and *inclusion*.** The `direction` argument determines the direction of the scan, either upward (from 0 to the highest-numbered node) or downward (from the highest-numbered node to 0). The `inclusion` argument determines whether a given node contributes *its own value* to the scan result. When `smode` is `CMMD_none`, the scan result is determined as follows:

- In inclusive upward scans, the value returned for a given node N is the combination of the source values in all nodes $\leq N$.
- In inclusive downward scans, the value returned for a given node N is the combination of the source values in all nodes $\geq N$.
- In exclusive upward scans, the value returned for a given node N is the combination of source values in all processors $< N$. The first (lowest-numbered) node receives the identity element for the combiner.
- In exclusive downward scans, the value returned for a given node N is the combination of source values in all nodes $> N$. The highest-numbered node receives the identity element for the combiner.

If a scan is a segmented scan, these rules apply on a per-segment basis, as explained below.

***smode* and *sbit*.** The `smode` and `sbit` arguments define *segmented scans*. These are scans that tally their results across subgroups of the nodes. Every node belongs

Copyright © 1995 Thinking Machines Corporation

to one group, or *segment*, with the group to which it belongs determined by *smode* and *sbit* as follows:

***smode* = CMMD_segment_bit**: If *smode* is CMMD_segment_bit, then *sbit* is considered a *segment bit*. A nonzero segment bit starts a new segment for an upward scan, but ends a segment for a downward scan. Imagine 8 nodes with the following segment bits:

```
0  0  1  0  0  1  0  0
```

Both upward and downward scans would have 3 segments: one would include nodes 0 and 1, another would include nodes 2−4, and a third would include nodes 5−7.

When *sbit* is a segment bit, operations in one segment never affect the values of elements in another segment. Thus, given segment bits of

```
0  0  1  0
```

and unsigned values of

```
4  1  5  2
```

an upward exclusive unsigned max would produce

```
0  4  0  5
```

***smode* = CMMD_start_bit**. If *smode* is CMMD_start_bit, then *sbit* is considered a *start bit*. A nonzero start bit always starts a new segment, whether the direction of the scan is upward or downward. Thus, given 8 nodes with the following start bits:

```
0  0  1  0  0  1  0  0
```

an upward scan would have the same segments as the segmented scan shown above (0−1, 2−4, 5−7); but a downward scan would have segments of 7−6, 5−3, and 2−0.

In addition, if the operation is exclusive, a node with a nonzero start bit does receive a value from the preceding segment. The value received is the reduce of the previous segment, with the same combiner. Thus, given start bits of

```
0  0  1  0
```

and values of

```
4  1  5  2
```


Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

an upward exclusive unsigned max would produce

```
0   4   4   5
```

but a downward exclusive unsigned max would produce

```
5   5   2   0
```

However, an upward inclusive unsigned max would produce

```
4   4   5   5
```

and a downward inclusive unsigned max would produce

```
5   5   5   2
```

Callback and Handler Functions. The global functions cannot be called from callback or handler functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_scan_done

Reports whether the Control Network has received data from all nodes and has performed the scan operation; returns 1 if the scan is complete, 0 otherwise.

SYNOPSIS

C Syntax

```
int CMMD_scan_done(void)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_scan_done()
```

RETURNS

`CMMD_scan_done` returns 1 if all nodes have called `CMMD_scan_`*`type`*`_start`, because this means that all nodes are now ready to participate in the scan operation. It returns 0 otherwise.

DESCRIPTION

The mechanics of a scan are as follows: first, each node sends its contribution to the scan into the Control Network; second, the Control Network computes the results of the scan; finally, each node receives the results of the scan from the Control Network. The three functions involved in a split scan cover these three activities.

`CMMD_scan_done` reports whether the Control Network has received data from all nodes and has performed the scan operation. It returns 1 if the scan is complete, 0 otherwise.

Callback and Handler Functions. The global functions cannot be called from within callback or handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_scan_*type*_finish

Moves the scan results from the Control Network to the nodes. Thus, it returns the value of the specified scan operation over all the nodes.

SYNOPSIS

C Syntax

```
int CMMD_scan_int_finish(int value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

uint CMMD_scan_uint_finish(uint value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

lint CMMD_scan_lint_finish(lint value, CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

luint CMMD_scan_luint_finish(luint value,
         CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

double CMMD_scan_float_finish(float value,
         CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)

double CMMD_scan_double_finish(double value,
         CMMD_combiner_t combiner, CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion)
```


Copyright © 1995 Thinking Machines Corporation

Fortran Syntax

```
INTEGER FUNCTION CMMD_scan_int_finish(value, combiner,
         direction, smode, sbit, inclusion)
    INTEGER   value
    INTEGER   combiner, direction, smode, sbit, inclusion

REAL*8 FUNCTION CMMD_scan_float_finish(value, combiner,
         direction, smode, sbit, inclusion)
    REAL*4    value
    INTEGER   combiner, direction, smode, sbit, inclusion

REAL*8 FUNCTION CMMD_scan_double_finish(value, combiner,
         direction, smode, sbit, inclusion)
    REAL*8    value
    INTEGER   combiner, direction, smode, sbit, inclusion
```

ARGUMENTS

value — The value to be contributed by this processor to the operation. Its type must match that specified by the function name.

combiner — One of the symbols listed below, specifying the type of operation to be performed.

For signed integer operands, allowable combiners are

```
CMMD_combiner_add   CMMD_combiner_ior
CMMD_combiner_max   CMMD_combiner_xor
CMMD_combiner_min   CMMD_combiner_and
```

For unsigned integer operands, allowable combiners are

```
CMMD_combiner_uadd CMMD_combiner_ior
CMMD_combiner_umax CMMD_combiner_xor
CMMD_combiner_umin CMMD_combiner_and
```

For signed long integer operands, allowable combiners are

```
CMMD_combiner_ladd CMMD_combiner_ior
CMMD_combiner_lmax CMMD_combiner_xor
CMMD_combiner_lmin CMMD_combiner_and
```


Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

For unsigned long integer operands, allowable combiners are

```
CMMD_combiner_luadd CMMD_combiner_ior
CMMD_combiner_lumax CMMD_combiner_xor
CMMD_combiner_lumin CMMD_combiner_and
```

For float operands (see note below), allowable combiners are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands (see note below), allowable combiners are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

NOTE: For back compatibility, float and double operands may also use the combiners `CMMD_combiner_add`, `CMMD_combiner_max`, and `CMMD_combiner_min`. The use of these combiners for such operands is deprecated, however, and its continued support is not ensured.

direction	`CMMD_upward`	The scan starts at node 0 and proceeds to the highest-numbered node.
	`CMMD_downward`	The scan starts at the highest-numbered node and proceeds downward to node 0.
smode	`CMMD_none`	The scan proceeds across all nodes.
	`CMMD_segment_bit`	The scan is a segmented scan, with *sbit* acting as a segment bit.
	`CMMD_start_bit`	The scan is a segmented Scan, with *sbit* acting as a start bit.

sbit — If *sbit* is nonzero, the node marks the boundary (usually the beginning) of a segment; if *sbit* is 0, the node is not a boundary marker.

(If *smode* is set to `CMMD_none`, *sbit* is ignored.)

Copyright © 1995 Thinking Machines Corporation

inclusion	CMMD_inclusive	The scan is inclusive.
	CMMD_exclusive	The scan is exclusive.

RETURNS

CMMD_scan_*type*_finish returns the value of the specified scan operation over all the nodes. This function does not return until all nodes have called the function.

DESCRIPTION

The mechanics of a scan are as follows: first, each node sends its contribution to the scan into the Control Network; second, the Control Network computes the results of the scan; finally, each node receives the results of the scan from the Control Network. The three functions involved in a split scan cover these three activities.

CMMD_scan_*type*_finish moves the scan results from the Control Network to the nodes. Thus, it returns the value of the specified scan operation over all the nodes.

PLEASE NOTE

1. Values for *direction*, *smode*, and *inclusion* ***must*** be identical across all nodes. Otherwise, results are unpredictable and the program may crash.
2. For CMMD_scan_float and CMMD_scan_double, the combination of *smode* = CMMD_start_bit and *inclusion* = CMMD_exclusive is currently illegal and causes the nodes to exit.

***direction* and *Inclusion*.** The *direction* argument determines the direction of the scan, either upward (from 0 to the highest-numbered node) or downward (from the highest-numbered node to 0). The *inclusion* argument determines whether

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

a given node contributes *its own value* to the scan result. When *smode* is CMMD_none, the scan result is determined as follows:

- In inclusive upward scans, the value returned for a given node *N* is the combination of the source values in all nodes ≤ *N*.
- In inclusive downward scans, the value returned for a given node *N* is the combination of the source values in all nodes ≥ *N*.
- In exclusive upward scans, the value returned for a given node *N* is the combination of source values in all processors < *N*. The first (lowest-numbered) node receives the identity element for the combiner.
- In exclusive downward scans, the value returned for a given node *N* is the combination of source values in all nodes > *N*. The highest-numbered node receives the identity element for the combiner.

If a scan is a segmented scan, these rules apply on a per-segment basis, as explained below.

***smode* and *sbit*.** The *smode* and *sbit* arguments define *segmented scans*. These are scans that tally their results across subgroups of the nodes. Every node belongs to one group, or *segment*, with the group to which it belongs determined by *smode* and *sbit* as follows:

***smode* = CMMD_segment_bit**: If *smode* is CMMD_segment_bit, then *sbit* is considered a *segment bit*. A nonzero segment bit starts a new segment for an upward scan, but ends a segment for a downward scan. Imagine 8 nodes with the following segment bits:

```
0  0  1  0  0  1  0  0
```

Both upward and downward scans would have 3 segments: one would include nodes 0 and 1, another would include nodes 2−4, and a third would include nodes 5−7.

When *sbit* is a segment bit, operations in one segment never affect the values of elements in another segment. Thus, given segment bits of

```
0  0  1  0
```

and unsigned values of

```
4  1  5  2
```


Copyright © 1995 Thinking Machines Corporation

an upward exclusive unsigned max would produce

```
0  4  0  5
```

***smode* = CMMD_start_bit**. If *smode* is CMMD_start_bit, then *sbit* is considered a *start bit*. A nonzero start bit always starts a new segment, whether the direction of the scan is upward or downward. Thus, given 8 nodes with the following start bits:

```
0  0  1  0  0  1  0  0
```

an upward scan would have the same segments as the segmented scan shown above (0−1, 2−4, 5−7); but a downward scan would have segments of 7−6, 5−3, and 2−0.

In addition, if the operation is exclusive, a node with a nonzero start bit does receive a value from the preceding segment. The value received is the reduce of the previous segment, with the same combiner. Thus, given start bits of

```
0  0  1  0
```

and values of

```
4  1  5  2
```

an upward exclusive unsigned max would produce

```
0  4  4  5
```

but a downward exclusive unsigned max would produce

```
5  5  2  0
```

However, an upward inclusive unsigned max would produce

```
4  4  5  5
```

and a downward inclusive unsigned max would produce

```
5  5  5  2
```

Callback and Handler Functions. The global functions cannot be called from within callback or handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_scan_v

Computes the value of the specified scan operation for each specified buffer element over all the nodes.

SYNOPSIS

C Syntax

```
void CMMD_scan_v(void *dest, void *source,
         CMMD_combiner_t combiner,
         CMMD_scan_direction_t direction,
         CMMD_segment_mode_t smode, int sbit,
         CMMD_scan_inclusion_t inclusion,
         int stride, int elem_count)
```

Fortran Syntax

```
SUBROUTINE CMMD_scan_v(dest, source, combiner, direction,
         smode, sbit, inclusion, stride, elem_count)
    <ARRAY>   dest, source
    INTEGER   combiner, direction, smode, sbit, inclusion
    INTEGER   stride, elem_count
```

ARGUMENTS

dest — The buffer that will hold the vector of computed values. It must be a serial array, not a parallel one.

source — The buffer that holds the vector being scanned. It must be a serial array, not a parallel one.

combiner — One of the symbols listed below, specifying the type of operation to be performed.

For signed integer operands, allowable combiners are

```
CMMD_combiner_add CMMD_combiner_ior
CMMD_combiner_max CMMD_combiner_xor
CMMD_combiner_min CMMD_combiner_and
```


Copyright © 1995 Thinking Machines Corporation

For unsigned integer operands, allowable combiners are

```
CMMD_combiner_uadd CMMD_combiner_ior
CMMD_combiner_umax CMMD_combiner_xor
CMMD_combiner_umin CMMD_combiner_and
```

For signed long integer operands, allowable combiners are

```
CMMD_combiner_ladd CMMD_combiner_ior
CMMD_combiner_lmax CMMD_combiner_xor
CMMD_combiner_lmin CMMD_combiner_and
```

For unsigned long integer operands, allowable combiners are

```
CMMD_combiner_luadd CMMD_combiner_ior
CMMD_combiner_lumax CMMD_combiner_xor
CMMD_combiner_lumin CMMD_combiner_and
```

For float operands, allowable combiners are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands, allowable combiners are

```
CMMD_combiner_dadd
CMMD_combiner_dmax
CMMD_combiner_dmin
```

Using anything other than an allowable combiner causes a fatal error.

direction	`CMMD_upward`	The scan starts at node 0 and proceeds to the highest-numbered node.
	`CMMD_downward`	The scan starts at the highest-numbered node and proceeds downward to node 0.
smode	`CMMD_none`	The scan proceeds across all nodes.
	`CMMD_segment_bit`	The scan is a segmented scan, with *`sbit`* acting as a segment bit.
	`CMMD_start_bit`	The scan is a segmented scan, with *`sbit`* acting as a start bit.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

sbit	If *sbit* is nonzero, the node marks the boundary (usually the beginning) of a segment; if *sbit* is 0, the node is not a boundary marker. (If *smode* is set to CMMD_none, *sbit* is ignored.)	
inclusion	CMMD_inclusive CMMD_exclusive	The scan is inclusive. The scan is exclusive.
stride	An integer specifying the distance in bytes between the starting addresses of vector elements. Must be a positive integer multiple of the number of bytes in an element.	
elem_count	An integer specifying the number of elements in the vector.	

RETURNS

This function does not return until all nodes have made the function call. It has no return value.

DESCRIPTION

The vector scan function computes the value of the specified scan operation for each specified buffer element over all the nodes. This function does not return until all nodes have called the function.

The rules for *direction*, *smode*, *sbit*, and *inclusion* are the same as for non-vector scans.

This function does not return a value. Instead, each value computed from a *source* buffer element is stored in the corresponding *dest* buffer element. The *source* and *dest* buffers may be the same, in which case the new vector overwrites the previous one. If *source* and *dest* overlap in any other way, the results are not predictable. *source* and *dest* must be serial buffers, not parallel ones.

Note that the type of vector scan is not specified by the function name. The scan type is specified only by the combiner.

If *elem_count* is 1, then you may mix CMMD_scan_*type* operations with vector operations. If *elem_count* is greater than 1, however, mixing vector and nonvector scans causes the system to hang.

Copyright © 1995 Thinking Machines Corporation

Callback and Handler Functions. The global functions cannot be called from within callback or handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

9.7 Concatenation Function

CMMD offers one concatenation function for hostless programming, `CMMD_concat_to_nodes`. Concatenation functions for host/node programming are described in Chapter 19.

Copyright © 1995 Thinking Machines Corporation

CMMD_concat_with_nodes

Concatenates element from each node into a buffer on all nodes.

SYNOPSIS

C Syntax

```
void CMMD_concat_with_nodes(void *element, void *buffer,
          int elem_length)
```

Fortran Syntax

```
SUBROUTINE CMMD_concat_with_nodes(element, buffer, elem_length)
    <ARRAY>   element, buffer
    INTEGER   elem_length
```

ARGUMENTS

element	The element that this node contributes to the concatenation process.
buffer	The buffer in which the returned value will be stored. Its length in bytes must be at least (CMMD_partition_size * *elem_length*). It must be a serial buffer, not a parallel one.
elem_length	The length in bytes of the element to be concatenated. Must be identical across all nodes.

RETURNS

This function does not return until all nodes in the partition have made the function call. It does not return a value.

DESCRIPTION

CMMD_concat_with_nodes concatenates elements of equal length from each node into the given (serial) buffer. The length of the buffer in bytes must be at

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

least (`CMMD_partition_size` * *`elem_length`*). This function does not return until all nodes have called `CMMD_concat_with_nodes`.

Callback and Handler Functions. The global functions cannot be called from within callback or handler functions.

Copyright © 1995 Thinking Machines Corporation