Chapter 19

Host/Node Programming Environment

CMMD provides facilities for creating a host/node environment for those application writers who prefer this style of programming. In the host/node environment, the programmer writes two programs — one that is replicated on all nodes in the partition, and another that resides on the partition manager.

Here is the previous "hello world" example written in the host/node programming style:

Host Program: `hello.host.c`

```
main(argc, argv)
  int argc;
  char *argv[];
{
int nprocs, i;
char buffer[1024];

CMMD_enable();

nprocs = CMMD_partition_size();
for(i=0; i <nprocs; i++) {
  CMMD_receive_block(CMMD_ANY_NODE, CMMD_ANY_TAG,
                     buffer, sizeof(buffer));
  printf("%s\n", buffer);
  }
}
```


Copyright © 1995 Thinking Machines Corporation

Node program: `hello.pn.c`

```
main(argc, argv)
  int argc;
  char *argv[];
{
  char buffer[1024];

  CMMD_enable_host();

  sprintf(buffer, "Node %d says hello\n",
          CMMD_self_address());
  CMMD_send_block(CMMD_host_node(), 0, buffer,
                  strlen(buffer)+2);
}
```

Notice the following:

- `CMMD_enable` is called in the host program. This downloads the node program and initializes CMMD.
- `CMMD_enable_host` is called in the node program. This routine communicates with the host and allows `CMMD_enable` to complete.
- The standard message-passing functions are used to exchange messages between the host and the node.
- `CMMD_host_node` is called to get the ID of the host node.
- Nodes cannot directly perform I/O, either to the user's terminal or to files, without host cooperation (see Section 19.8).

There are several issues the programmer needs to be aware of when using the host/node programming model:

- Asynchronous messages sent to the host do not interrupt the host program. The host program is responsible for polling if asynchronous message passing is used.
- The current CMMD implementation does not allow the host to send a message to itself, by any communication method.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

- Programmers wishing to make use of I/O services on the nodes are responsible for explicitly calling CMMD I/O server routines in their host program.
- Global operations used in host/node programs often differ from those used in hostless programs. For example, a hostless program would call `CMMD_reduce`; the host would not be involved, and the result would be stored on all nodes. A host/node program, in contrast, might want to store the result of the reduce on the host; it would have the host call `CMMD_reduce_from_nodes`, while the nodes called `CMMD_reduce_to_host`.
- The host can never participate in scans.
- Host/node global operations, like the hostless global operations, currently do not accept parallel arrays as arguments.
- The host can transfer only serial data to and from the nodes. Parallel data must be transposed to or from serial data on the nodes themselves as a separate step. (See Section 19.3.)
- Calling `exit` on the nodes causes the nodes to "busy wait" in a dispatch loop. A host/node program does not terminate until the host program exits. The exit status of the entire program is simply the exit status of the host part. (See Section 19.7.)

This chapter describes the CMMD functions that allow you to

- enable the host/node programming environment
- get the host ID
- explicitly synchronize the host with all nodes
- perform global operations that include the host: broadcasts from the host, concatenations and reductions to the host
- perform node I/O

Copyright © 1995 Thinking Machines Corporation

19.1 Enabling the Host/Node Programming Environment

Host/node programs require the application to call two functions, in order to initialize the programming environment. `CMMD_enable` must be the first CMMD function called on the host, while `CMMD_enable_host` must be the first CMMD function called on the nodes.

Reference pages for `CMMD_enable` and `CMMD_enable_host` are provided in Section 19.10.

19.2 Obtaining the Host ID

In the host/node environment, nodes communicate with the host using the usual CMMD message-passing functions. Nodes therefore need to know the host ID, so that they can address the host. They obtain this host ID by calling `CMMD_host_node`.

The host addresses each of its nodes simply by using their usual node ID. Attempts by the host program to send to a processor outside the range of valid node IDs in the current partition results in an error.

The reference page for `CMMD_host_node` is provided in Section 19.10.

19.3 Sending and Receiving Messages

For the most part, the host uses the same functions as the nodes do to send and receive synchronous and asynchronous messages. There are, however, some differences.

19.3.1 Serial Data Only

The host, being a serial computer, handles serial data only. Thus, it can send serial arrays and vector serial arrays to any node, and can receive such arrays from any node, using the standard message functions. It cannot, however, participate in the sending or receiving of parallel arrays.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

In particular, serial data from the host cannot be sent directly into a parallel array on the nodes via CMMD functions; if such a move is required, the data should first be brought into the node's serial (SPARC) memory as a serial array, and then transposed into a parallel array by the CM Fortran routine `CMF_FE_ARRAY_TO_CM`. For transposing data from a parallel array into a serial array, the routine is `CMF_FE_ARRAY_FROM_CM`. (Consult the CM Fortran utility library, documented in the *CM Fortran Libraries Reference Manual*, or the manual pages for `CMF_FE_ARRAY_TO_CM` or `CMF_FE_ARRAY_FROM_CM`, for further details.)

19.3.2 Active Messages

The host can send and receive active messages, to or from the nodes. It is illegal, however, to attempt a pcopy, either from the host or from a node to the host. The reason is that the host supports serial data only.

The host uses the same functions that nodes do to send active messages to the nodes. The nodes, however, use special functions to send active messages to the host: for example,

- `CMAML_request_tohost`
- `CMAML_reply_tohost`
- `CMAML_rpc_tohost`

For information on these functions, see Chapter 12.

19.4 Synchronizing the Host with the Nodes

Like the nodes, the host may call `CMMD_set_global_or` and `CMMD_get_global_or`. These functions may thus be used for host/node synchronization as well as for node synchronization.

In addition, a pair of global synchronization functions, `CMMD_sync_host_with_nodes` and `CMMD_sync_with_host`, can be used to synchronize the host with the nodes. The reference page for these functions may be found in Section 19.10.

Copyright © 1995 Thinking Machines Corporation

19.5 Broadcasts and Concatenation

CMMD supplies two forms of host-to-node broadcasts:

- `CMMD_bc_from_host[_v]`
  `CMMD_receive_bc_from_host[_v]`

  Host broadcasts a serial array or vector strided array to all nodes. All nodes receive identical data.

- `CMMD_distrib_to_nodes`
  `CMMD_receive_element_from_host`

  Host distributes or "scatters" a serial array to all the nodes. Each node receives different data, with data being distributed to nodes in processor order.

CMMD also supplies a concatenation function that is the reverse of this last broadcasting function:

- `CMMD_gather_from_nodes`
  `CMMD_concat_elements_to_host`

  All nodes contribute serial data to be concatenated, in processor order, into a buffer on the host.

Reference pages for these functions are supplied in Section 19.10.

19.6 Reduces from Nodes to Host

Ordinarily, the host is not involved in reduce operations. CMMD, however, provides matching host and node functions for including the host in these operations. As with other global operations, these routines do not return until all nodes and the host issue the call. These routines are

- `CMMD_reduce_from_nodes_`*`type`*
  `CMMD_reduce_to_host_`*`type`*

- `CMMD_reduce_from_nodes_v`
  `CMMD_reduce_to_host_v`

Reference pages for these routines are supplied in Section 19.10.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

NOTE: It is an error to call the node-only routine `CMMD_reduce` on the nodes when `CMMD_reduce_from_nodes` is being called on the host.

19.7 Terminating a Host/Node Program

A host/node program terminates normally when the host program calls `exit`. Calls to `exit` made on the nodes do not terminate the node program; rather, they cause the nodes to "busy wait" in a dispatch loop until the host program terminates execution.

NOTE: The exit status returned by a host/node program is the exit status returned by the *host* program. The exit statuses of individual nodes are not accessible.

19.8 Invoking an I/O Server on the Host

Programmers wishing to make UNIX I/O calls on the nodes must ensure that their host program invokes the CMMD I/O server routines appropriately. I/O calls on the nodes result in *CMMD system messages* being sent to the host, and the programmer is responsible for scheduling the I/O server to handle these requests.

The most common way is to call `CMMD_service_loop` on the host. This function starts an I/O server that remains in effect until cancelled by a call to `CMMD_global_suspend_servers` on the nodes. Alternatively, the program may simply poll for services from time to time, using the host function `CMMD_poll_for_services`.

Reference pages for the I/O server functions are provided in Section 19.10.

Copyright © 1995 Thinking Machines Corporation

An Example

The example below shows proper usage of the I/O server functions in a host/node program using I/O.

```
HOST program: hello_io.host.c

main(argc, argv)
  int argc;
  char *argv[];
{
  int nprocs, msgs_received;
  char buffer[1024];

  CMMD_enable();

  nprocs = CMMD_partition_size();

  msgs_received = 0;

  while ( msgs_received < nprocs ) {

    if (CMMD_msg_pending(CMMD_ANY_NODE, CMMD_ANY_TAG)){
      CMMD_receive_block(CMMD_ANY_NODE, CMMD_ANY_TAG,
                         buffer, sizeof(buffer));
      printf("%s\n", buffer);
      msgs_received++;
    }
    CMMD_poll_for_services();
  }
}
```


Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

```
NODE program: hello_io.pn.c

main(argc, argv)
  int argc;
  char *argv[];
{
  char buffer[1024];

  CMMD_enable_host();

  sprintf(buffer, "Node %d says hello via CMMD_send\n",
          CMMD_self_address());

  CMMD_send_block(CMMD_host_node(), 0, buffer, strlen(buffer)+2);

  printf("Node %d all done\n", CMMD_self_address());
}
```

19.9 Further Server Functions

CMMD allows programmers to alter the default host server program, or even to replace it with a custom server program to perform application-specific operations. However, the issues involved are beyond the scope of this document. If your application requires you to modify the default server program or write your own, please consult with your applications engineer or with Thinking Machines Corporation customer support for more information.

19.10 Reference Pages

This section contains reference pages for the following host/node functions:

- `CMMD_enable` and `CMMD_enable host`
- `CMMD_host_node`
- `CMMD_sync_host_with_nodes` and `CMMD_sync_with_host`
- `CMMD_bc_from_host[_v]` and `CMMD_receive_bc_from_host[_v]`
- `CMMD_distrib_to_nodes` and `CMMD_receive_element_from_host`

Copyright © 1995 Thinking Machines Corporation

- `CMMD_gather_from_nodes` and `CMMD_concat_elements_to_host`
- `CMMD_reduce_from_nodes_`*`type`* and `CMMD_reduce_to_host_`*`type`*
- `CMMD_reduce_from_nodes_v` and `CMMD_reduce_to_host_v`
- `CMMD_service_loop`
  `CMMD_global_suspend_servers`
  `CMMD_poll_for_services`

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_enable
CMMD_enable_host

These must be the first CMMD functions called by the host and all nodes in a host/node program.

SYNOPSIS

C Syntax

```
void CMMD_enable(void)
void CMMD_enable_host(void)
```

Fortran Syntax

```
SUBROUTINE CMMD_enable()
SUBROUTINE CMMD_enable_host()
```

DESCRIPTION

`CMMD_enable` must be called on the host only. This function downloads the node program and initializes the message-passing system. It can be called only once, at the beginning of a program. Calling it twice in the same program is an error.

`CMMD_enable_host` must be called on all nodes. This function synchronizes the nodes and causes the matching `CMMD_enable` routine on the host to return.

Copyright © 1995 Thinking Machines Corporation

CMMD_host_node

Returns the logical identifier for the host processor.

SYNOPSIS

C Syntax

```
int CMMD_host_node(void)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_host_node()
```

RETURNS

`CMMD_host_node` returns the host identifier.

DESCRIPTION

`CMMD_host_node` returns the host identifier (a positive integer always greater than the largest node identifier). It can be called from the host itself or from any node.

NOTE: The value returned by this function remains constant throughout the execution of an application, whether or not the "logical partition" is resized by `CMMD_reset_partition_size`.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_sync_host_with_nodes
CMMD_sync_with_host

Global function; synchronizes the host and all nodes.

SYNOPSIS

C Syntax

```
void CMMD_sync_host_with_nodes(void)
void CMMD_sync_with_host(void)
```

Fortran Syntax

```
SUBROUTINE CMMD_sync_host_with_nodes()
SUBROUTINE CMMD_sync_with_host()
```

RETURNS

These functions do not return until the host and all the nodes have made the function call. They have no return value.

DESCRIPTION

This pair of calls serves as a synchronization point for host and nodes together. The host calls `CMMD_sync_host_with_nodes`; all nodes call `CMMD_sync_with_host`.

All processors in the partition must join in these calls. Once the host or any node has begun one of these synchronization calls, the CM hardware keeps track of responses, and allows none of the calls to return until all nodes plus host have made their calls.

Copyright © 1995 Thinking Machines Corporation

CMMD_bc_from_host
CMMD_receive_bc_from_host

The host calls `CMMD_bc_from_host` to broadcast a message to all nodes. Each node calls `CMMD_receive_bc_from_host` to receive the broadcast message.

SYNOPSIS

C Syntax

```
void CMMD_bc_from_host(void *buffer, int buffer_desc)
void CMMD_receive_bc_from_host(void *buffer, int buffer_desc)
```

Fortran Syntax

```
SUBROUTINE CMMD_bc_from_host(buffer, buffer_desc)
SUBROUTINE CMMD_receive_bc_from_host(buffer, buffer_desc)
    <ARRAY> buffer
    INTEGER buffer_desc
```

ARGUMENTS

buffer	A serial buffer that holds the message being broadcast and received.
buffer_desc	The length, in bytes, of the buffer being broadcast and received.

RETURNS

These functions do not return until the broadcast is complete, that is, until the host and all the nodes have made their calls. They do not return any values.

DESCRIPTION

These functions constitute a global broadcast operation, with the host included. The host program calls `CMMD_bc_from_host` to broadcast a buffer of the specified length (in bytes) to all nodes. All nodes must call `CMMD_receive_bc_from_host`, with the same length argument, to receive the buffer.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

NOTE

If length arguments are not identical across all nodes, a segmentation fault may result.

All processors within the partition must take part in this operation. Otherwise, the operation will fail and the program may hang.

Copyright © 1995 Thinking Machines Corporation

CMMD_bc_from_host_v
CMMD_receive_bc_from_host_v

Vector serial version of broadcast from the host to all nodes.

SYNOPSIS

C Syntax

```
void CMMD_bc_from_host_v(void *vector, int elem_bytes,
        int stride, int elem_count)

void CMMD_receive_bc_from_host_v(void *vector, int elem_bytes,
        int stride, int elem_count)
```

Fortran Syntax

```
SUBROUTINE CMMD_bc_from_host_v(vector, elem_bytes, stride,
        elem_count)

SUBROUTINE CMMD_receive_bc_from_host_v(vector, elem_bytes,
        stride, elem_count)

    <ARRAY> vector
    INTEGER elem_bytes, stride, elem_count
```

ARGUMENTS

vector	A serial buffer that holds the vector of message elements being broadcast and received.
elem_bytes	An integer specifying the length of each element in the vector.
stride	An integer specifying the distance in bytes between the starting addresses of vector elements.
elem_count	An integer specifying the number of elements in the vector.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

RETURNS

These functions do not return until the broadcast is complete, that is, until the host and all the nodes have made their calls.

DESCRIPTION

The host program makes a single call to `CMMD_bc_from_host` or `CMMD_bc_from_host_v` in order to broadcast message elements from the vector to all nodes. All nodes must make a single call to `CMMD_receive_bc_from_host` or `CMMD_receive_bc_from_host_v`, with arguments specifying the same total length, in order to receive the vector. All nodes receive all elements of the vector.

NOTE

The total length of an elementary broadcast message is given by the *len* argument, whereas the total length of a vector broadcast message is the product of the *elem_bytes* and *elem_count* arguments.

If total lengths are not identical across all nodes, a segmentation fault may result.

Please note also that all processors within the partition must take part in this operation. Otherwise, the operation will fail and the program may hang.

`CMMD_bc_from_host_v` and `CMMD_receive_bc_from_host_v` are frequently paired, so that data are received in the same geometry in which they were sent. It is possible, however, to receive data in a geometry different from that in which they were sent: for instance, sequential data may be broken into a vector (thus "scattering" the data), or a vector may be received in sequence (thus "gathering" it). The four possible combinations are the same as for the send and receive functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_distrib_to_nodes CMMD_receive_element_from_host

Host broadcasts message, each node receives one portion.

SYNOPSIS

C Syntax

```
void CMMD_distrib_to_nodes(void *buffer, int elem_length)
void CMMD_receive_element_from_host(void *buffer, int length)
```

Fortran Syntax

```
SUBROUTINE CMMD_distrib_to_nodes(buffer, elem_length)
SUBROUTINE CMMD_receive_element_from_host(buffer, length)
    <ARRAY> buffer
    INTEGER length
```

ARGUMENTS

buffer	The serial buffer that holds the messages being sent and received. For the host, the length of the buffer (in bytes) must be at least (CMMD_partition_size() * *elem_length*). For a node, the length must be at least *elem_length*.
elem_length	The length (in bytes) of each element to be sent.
length	The length (in bytes) of the buffer that is to receive the element being sent.

RETURNS

Neither the host call nor any of the node calls return until all have been made and completed. There are no return values.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

DESCRIPTION

The host program calls `CMMD_distrib_to_nodes` in order to distribute elements of the given length from the specified buffer to each node in processor order. The length (in bytes) of the buffer on the host must be at least `(CMMD_partition_size * elem_length)`. Only the first `(CMMD_partition_size * elem_length)` bytes are sent; each node receives one element.

In response to the host call, all nodes must call `CMMD_receive_element_from_host`, specifying a buffer of the appropriate size to receive the element.

Copyright © 1995 Thinking Machines Corporation

CMMD_gather_from_nodes CMMD_concat_elements_to_host

Global concatenation, with host concatenating elements from all nodes.

SYNOPSIS

C Syntax

```
void CMMD_gather_from_nodes(void *buffer, int elem_length)
void CMMD_concat_elements_to_host(void *element, int elem_length)
```

Fortran Syntax

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

ARGUMENTS

buffer	(Host only.) The buffer in which the returned value will be stored. Its length in bytes must be at least (CMMD_partition_size * *elem_length*).
element	(Nodes only.) The element this node contributes to the concatenation process.
elem_length	The length in bytes of the element to be concatenated. (Must be identical for all processors).

RETURNS

None of the calls returns until host and all nodes have made the call. There are no return values.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

DESCRIPTION

This pair of functions concatenates elements from each node into a buffer on the host. The element length must be identical for all processors, and the host must specify enough space to store the result. The function returns after all nodes have called `CMMD_concat_elements_to_host` and the host has called `CMMD_gather_from_nodes`.

Note that these functions are essentially the opposite of the functions `CMMD_distrib_to_nodes` and `CMMD_receive_element_from_host`. That pair distributes the contents of a buffer element-wise from the host to the nodes; this pair gathers the elements from the nodes into a buffer on the host.

Copyright © 1995 Thinking Machines Corporation

CMMD_reduce_from_nodes_*type*
CMMD_reduce_to_host_*type*

Global serial reduction from all nodes to host and all nodes.

SYNOPSIS

C Syntax

```
int CMMD_reduce_from_nodes_int(int value,
        CMMD_combiner_t combiner)
uint CMMD_reduce_from_nodes_uint(uint value,
         CMMD_combiner_t combiner)
lint CMMD_reduce_from_nodes_lint(lint value,
        CMMD_combiner_t combiner)
luint CMMD_reduce_from_nodes_luint(luint value,
        CMMD_combiner_t combiner)
double CMMD_reduce_from_nodes_float(float value,
        CMMD_combiner_t combiner)
double CMMD_reduce_from_nodes_double(double value,
        CMMD_combiner_t combiner)

int CMMD_reduce_to_host_int(int value,
        CMMD_combiner_t combiner)
uint CMMD_reduce_to_host_uint(uint value,
        CMMD_combiner_t combiner)
lint CMMD_reduce_to_host_lint(lint value,
        CMMD_combiner_t combiner)
luint CMMD_reduce_to_host_luint(luint value,
        CMMD_combiner_t combiner)
double CMMD_reduce_to_host_float(float value,
        CMMD_combiner_t combiner)
double CMMD_reduce_to_host_double(double value,
        CMMD_combiner_t combiner)
```

Fortran Syntax

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

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


Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

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

INTEGER FUNCTION CMMD_reduce_to_host_int(value, combiner)
    INTEGER   value, combiner

REAL*8 FUNCTION CMMD_reduce_to_host_float(value,combiner)
    REAL*8    value
    INTEGER   combiner

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

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
```

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
```


Copyright © 1995 Thinking Machines Corporation

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.

RETURNS

These functions return to the host and all nodes the value of the specified reduce operation over all processors, including the host. Host and all nodes must call this function before the call returns on any processor.

DESCRIPTION

In this host/node pair of functions, the host calls `CMMD_reduce_from_nodes_`*`type`* and all nodes call `CMMD_reduce_to_host_`*`type`*. The functions return to the host processor and to each node the value of the specified reduce operation over all the nodes including the host processor. The functions do not return until all nodes have called `CMMD_reduce_to_host_`*`type`* and the host has called `CMMD_reduce_from_nodes_`*`type`*.

Note that reductions on both single- and double-precision data return double-precision values.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_reduce_from_nodes_v
CMMD_reduce_to_host_v

Global vector reduction from all nodes to host and all nodes.

SYNOPSIS

C Syntax

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

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

Fortran Syntax

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

ARGUMENTS

dest — A buffer to hold the vector of reduced values.

source — A serial buffer that holds the vector being reduced.

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 (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.

`stride` An integer specifying the distance in bytes between the starting addresses of vector elements.

`elem_count` An integer specifying the number of elements in the vector.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

RETURNS

The functions do not return until all nodes have called `CMMD_reduce_to_host_v` and the host has called `CMMD_reduce_from_nodes_v`.

DESCRIPTION

In this host/node pair of functions, the host calls `CMMD_reduce_from_nodes_v` and all nodes call `CMMD_reduce_to_host_v`. The functions compute in the host processor and in each node the values of the specified reduce operation over all the nodes including the host processor. The element size used by these functions is equal to the data type of the element and is specified by the combiner.

Copyright © 1995 Thinking Machines Corporation

CMMD_service_loop
CMMD_global_suspend_servers
CMMD_poll_for_services

CMMD I/O server functions, for use in host/node programs.

SYNOPSIS

C Syntax

```
void CMMD_service_loop(void)
void CMMD_global_suspend_servers(void)
int CMMD_poll_for_services(void)
```

Fortran Syntax

```
SUBROUTINE CMMD_service_loop()
SUBROUTINE CMMD_global_suspend_servers()
INTEGER FUNCTION CMMD_poll_for_services()
```

RETURNS

`CMMD_service_loop` does not return until `CMMD_global_suspend_servers` is called on the nodes. Neither of these routines has any return value.

`CMMD_poll_for_services` returns 0 if there were no I/O services to perform. If there were any I/O services pending, it handles a single one, returning −1 if that service was a suspend request and 1 otherwise.

Important: If a call to `CMMD_poll_for_services` returns −1, you *must* follow it with a call to `CMMD_sync_host_with_nodes` to synchronize the host with the nodes, or the node calls to `CMMD_global_suspend_servers` will hang.

DESCRIPTION

`CMMD_service_loop`, which is called by the host program, starts an I/O server on the host that handles all I/O services requested by the nodes. The function does not return until all the nodes call `CMMD_global_suspend_servers`, thus suspending the server.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

`CMMD_global_suspend_servers` does a `sync_with_nodes`, kills the server on the host, synchronizes the nodes again, and returns.

In cases where it is not desirable to have the host program wait in a server loop, the programmer may periodically poll for I/O service requests using the `CMMD_poll_for_services` function. `CMMD_poll_for_services` returns 0 if there are no I/O services to perform. If there are any I/O services pending, it handles a single one, returning −1 if that service was a suspend request and 1 otherwise.

PLEASE NOTE

CMMD sends its own internal messages (*system messages*) to the I/O server using a special message tag. User-written host programs may interfere with system messages by not polling often enough or by not receiving a user message that blocks a waiting system message. For this reason, construction of host/node programs that use node-level I/O requires careful use of polling.

Copyright © 1995 Thinking Machines Corporation