Chapter 16

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);
        }
}
```

```
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 itself.
- **`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 do not directly perform any I/O, either to the user's terminal or to files.

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

- 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.

- 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`**.
- Also, 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 16.3.)
- Calling **`exit()`** on the nodes causes the nodes to "busy wait" in a dispatch loop. The program does not terminate until the host program calls **`exit()`**. (See Section 16.7.)

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
- handle node I/O

16.1 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.

FUNCTIONS **CMMD_enable**
CMMD_enable_host

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.

16.2 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 address in the current partition results in an error.

FUNCTION **CMMD_host_node**

C SYNTAX

int **CMMD_host_node(**void**)**

FORTRAN SYNTAX

INTEGER FUNCTION **CMMD_host_node(**)

DESCRIPTION

CMMD_host_node returns the host identifier (an integer not in the partition set). 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**.

16.3 Sending and Receiving Messages

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.

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 Reference Manual*, or the manual pages for **CMF_FE_ARRAY_TO_CM** or **CMF_FE_ARRAY_FROM_CM**, for further details.)

The host can participate in sending and receiving active messages, with the same constraints: the data transferred must be serial data, not parallel data. The host uses the same functions that nodes do to send active messages to the nodes.

The nodes use special functions to send active messages to the host:

- **CMAML_request_tohost**
- **CMAML_reply_tohost**
- **CMAML_rpc_tohost**

For information on these functions, see Section 11.6.1 in the chapter on Active Messages.

Two additional constraints apply when having the host program send or receive messages:

- The host cannot send messages to itself, nor receive messages from itself.
- Interrupts cannot be enabled on the host. If asynchronous message functions or active messages are used, the host program must poll regularly in order to send or receive the messages.

16.4 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.

FUNCTIONS **CMMD_sync_host_with_nodes**
CMMD_sync_with_host

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()**

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.

16.5 Broadcasts and Concatenation

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

- The host can broadcast a serial array or vector strided array to all nodes. As the name implies, all nodes receive identical data.
- The host can distribute a serial array among all nodes. In this case, each node receives different data, with data being distributed to nodes in processor order.

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

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

16.5.1 Host Broadcasts an Entire Buffer to All Nodes

FUNCTIONS **CMMD_bc_from_host**
CMMD_receive_bc_from_host

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.

DESCRIPTION

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.

NOTE

If length arguments 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.

16.5.2 Host Broadcasts a Vector to the Nodes

FUNCTIONS **`CMMD_bc_from_host_v`**
`CMMD_receive_bc_from_host_v`

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.

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.

16.5.3 Host Distributes a Buffer among the Nodes

FUNCTIONS **`CMMD_distrib_to_nodes`**
`CMMD_receive_element_from_host`

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.

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.

16.5.4 Host Receives Data Concatenated from Nodes

Concatenation appends the value from each processor to the values of all preceeding processors (in processor identifier order). In the host/node environment CMMD provides a function for concatenating values from every node into a buffer on the host. Concatenation always proceeds from the lowest to the highest node; it is never segmented.

FUNCTIONS **CMMD_gather_from_nodes**
CMMD_concat_elements_to_host

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***)**

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.

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.

16.6 Reduction from Nodes to Host

Ordinarily, the host is not involved in reduction operatations. 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.

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.

16.6.1 Reduction from Nodes to Host

FUNCTIONS **CMMD_reduce_from_nodes_***type*
CMMD_reduce_to_host_*type*

C SYNTAX

int **CMMD_reduce_from_nodes_int(**int *value*,
CMMD_combiner_t *combiner***)**
uint **CMMD_reduce_from_nodes_uint(**unsigned *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(**unsigned *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 *value*
integer *combiner*

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 *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 float operands (see note, below), allowable operands are

```
CMMD_combiner_fadd
CMMD_combiner_fmax
CMMD_combiner_fmin
```

For double operands (see note, below), allowable operands 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.

16.6.2 Vector Reduction from Nodes to Host

FUNCTIONS **`CMMD_reduce_from_nodes_v`**
`CMMD_reduce_to_host_v`

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 that holds 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`

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 float operands, allowable operands are

`CMMD_combiner_fadd`
`CMMD_combiner_fmax`
`CMMD_combiner_fmin`

For double operands, allowable operands 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.

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

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.

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`**.

16.7 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 value returned by the nodes is not accessible.

16.8 Invoking an I/O Server on the Host

Programmers wishing to make UNIX I/O calls on the nodes must ensure that their host programs invoke the CMMD I/O server routines appropriately. I/O calls on the nodes result in 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()**.

FUNCTIONS **CMMD_service_loop**
CMMD_global_suspend_servers
CMMD_poll_for_services

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 performs 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_terminate_servers** will hang.

DESCRIPTION

Called by the host program, **CMMD_service_loop** 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.

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 were no I/O services to perform. If there were any I/O services pending, it performs a single one, returning –1 if that service was a suspend request and 1 otherwise.

PLEASE NOTE

(1) If a call to **CMMD_poll_for_services** returns –1, you must follow it with a call to the function **CMMD_sync_host_with_nodes** to synchronize the host with the nodes, or otherwise the node calls to **CMMD_terminate_servers** will hang.

(2) Messages intended for the I/O server have a special message tag. User-written host programs may interfere with these messages by not polling often enough or by not receiving a user message that blocks a waiting I/O message. For this reason, construction of host/node programs that use node-level I/O requires careful programming practice.

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();
        }
}

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());
}
```

16.9 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.