Chapter 8

Virtual Channels

When a message-passing program has a pattern of communication between nodes that is repeatedly used and is relatively static in form, the handshaking overhead involved in point-to-point message passing can become an unnecessary burden. This is particularly true for programs that transmit many small messages, where the handshaking is a significant part of the overall transmission time of each message.

An example of this kind of program is one that uses a *stencil*, a fixed pattern of data exchange that is the same for every node. As an example, Conway's Game of Life uses a stencil consisting of a node and its eight nearest neighbors on a two-dimensional grid:

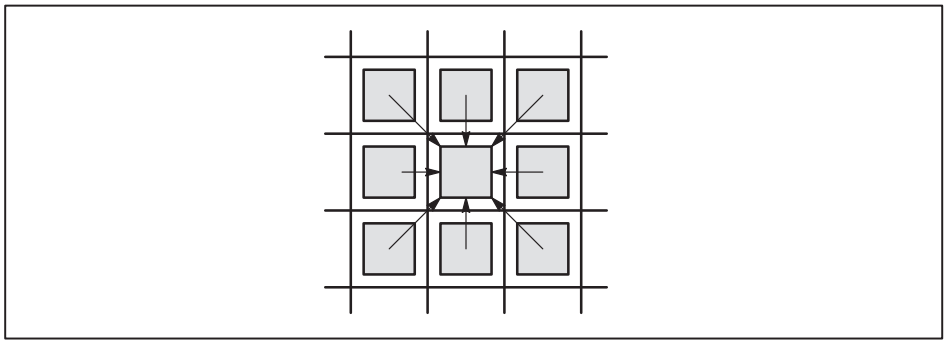

Figure 12. The stencil used by Conway's Game of Life.

To minimize handshaking overhead in these situations, CMMD provides *virtual channel* functions, which allow two nodes to establish a direct, one-way transmission link for data. (If two-way transmission is required, the nodes can set up a second channel going the other way.)

Copyright © 1995 Thinking Machines Corporation

When a virtual channel is established, the handshaking between the nodes is done only once. The nodes exchange information about the type, size, and shape of the array to be passed. The channel can then remain active over many uses, with no further handshake overhead. So long as the nodes always exchange messages with the same source array, destination array, array size, data type, and (for parallel arrays) shape, they no longer need to incur the handshaking overhead.

NOTE: These are called *virtual* channels because they are implemented entirely in software. They do not restrict the actual paths of the messages through the Data Network.

8.1 Using Virtual Channels

There is a specific protocol for establishing and using a virtual channel:

- The source node opens a *send channel*, and the destination node opens a *receive channel*. These send and receive channels represent the two ends of the channel pipeline:

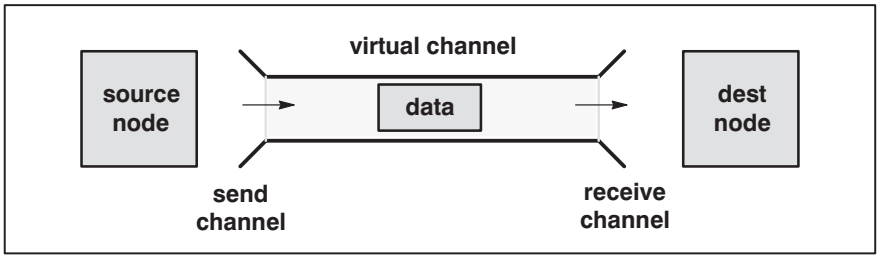

Figure 13. A virtual channel is a one-way pipeline.

- When the source node needs to send a message, it writes data to the source array of the channel, and then performs a *write* operation. This delivers the data to the destination node, and marks the channel as *readable*.

- When the destination node detects that the channel is readable, it either copies the data from the destination array to some other location or makes use of it in place. It then *resets* the channel, marking

Copyright © 1995 Thinking Machines Corporation

it as *writable*. This notifies the sender that the data has been received, and that the channel can be reused.

NOTE: A virtual channel can transmit only one message at a time; each message must be received and the channel must be reset before another message can be sent.

- When the source node is finished sending data, it can mark the send channel as *closed*. Likewise, when the destination node is finished receiving data, it can mark the receive channel as closed. Only when both ends of the channel have been marked as closed is the channel actually deallocated and freed for further use.

Figure 14 and Figure 15 show the steps by which this process is implemented.

Virtual channel read and write operations return immediately, reporting either success or failure. This reduces the latency of the channel operations and simplifies internal mechanisms. However, it also implies the following restrictions:

- Channel transmissions are unbuffered. If the write operation does not succeed, the sending node must retry the operation.

- Channels are either in a *writable* state or a *readable* state. The receiving node must explicitly reset the channel, or the sender will not be able to use it to transmit more data.

Channels are a limited resource. Each uses either a send port or a receive port. To check whether ports are available, an application can call `CMMD_available_sends` or `CMMD_available_receives`, described in Section 5.4.

Copyright © 1995 Thinking Machines Corporation

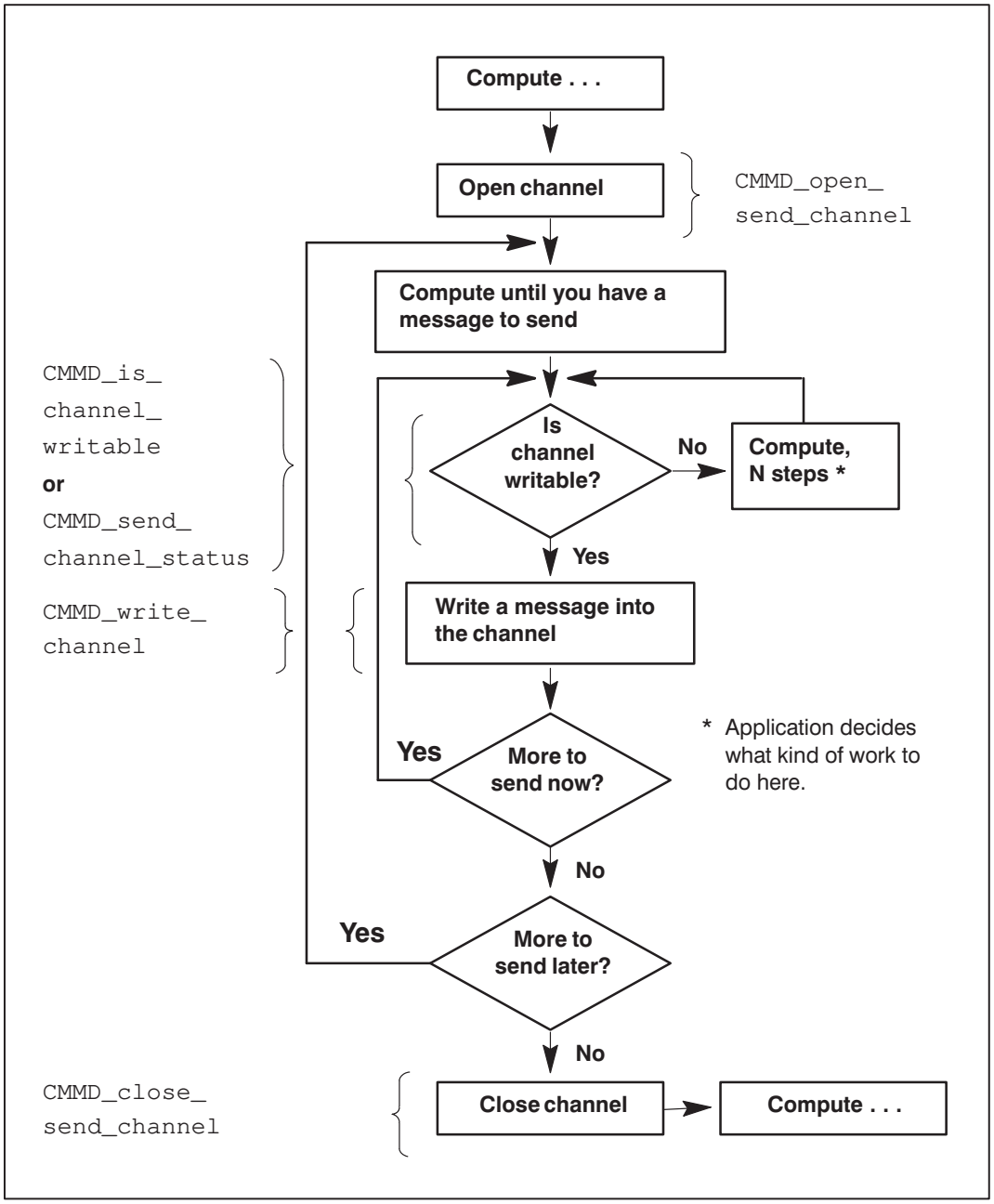

Figure 14. Virtual channel for sending data.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

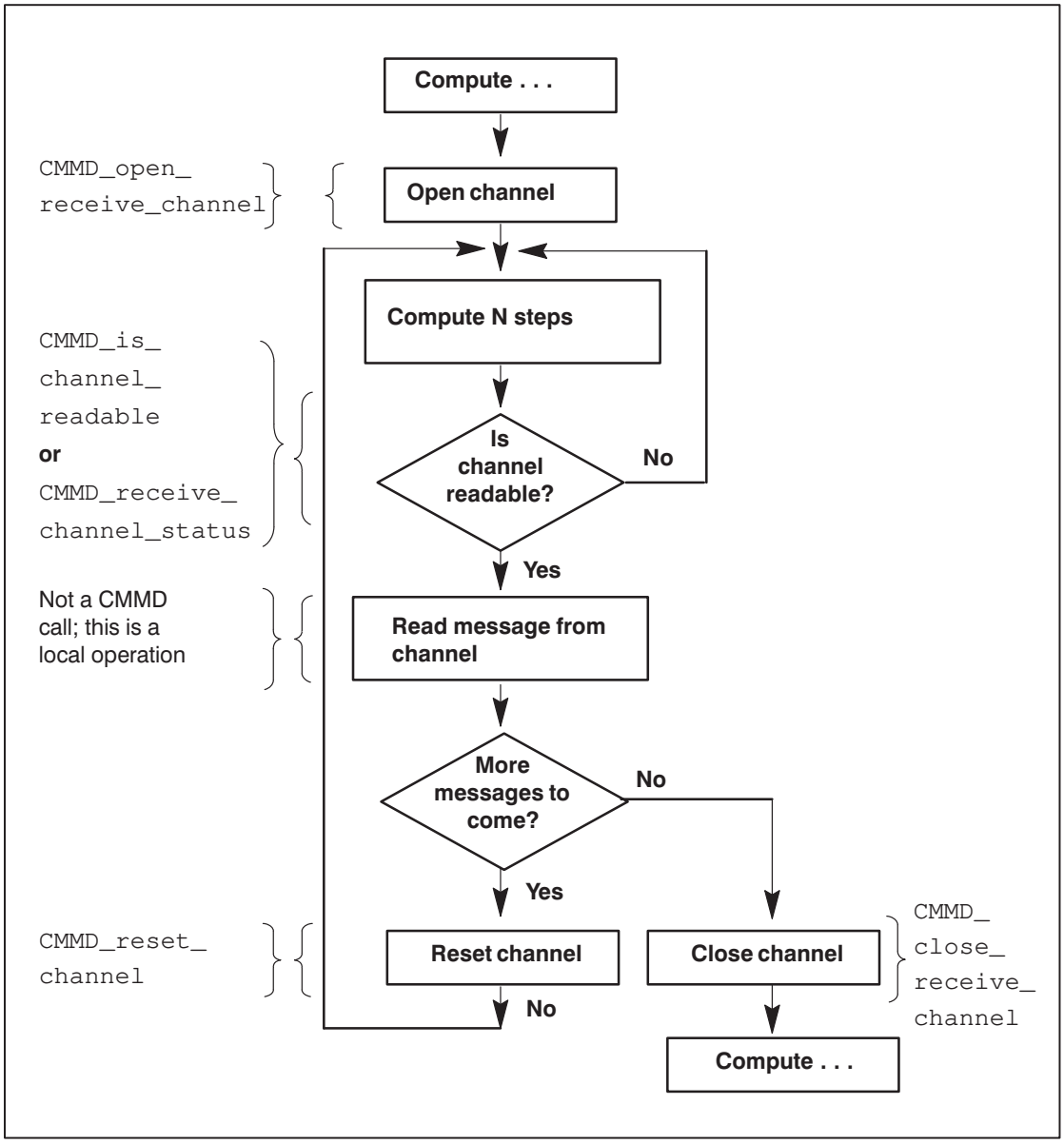

Figure 15. Virtual channel for receiving data.

Copyright © 1995 Thinking Machines Corporation

8.2 Virtual Channel Functions

CMMD provides the following functions for creating and using send channels:

`CMMD_open_send_channel`	Opens a channel for sending.
`CMMD_send_channel_status`	Returns current send channel state.
`CMMD_is_channel_writable`	Polls network and tests whether send channel is writable.
`CMMD_write_channel`	Sends message, marks channel as readable.
`CMMD_close_send_channel`	Marks a send channel as closable.

CMMD provides the following functions for creating and using receive channels:

`CMMD_open_receive_channel`	Opens a channel for receiving.
`CMMD_receive_channel_status`	Returns current receive channel state.
`CMMD_is_channel_readable`	Polls network and then tests whether receive channel is readable.
`CMMD_reset_channel`	Marks receive channel as writable.
`CMMD_close_receive_channel`	Closes a receive channel.

CMMD also provides the following two functions for testing that all channels have been closed:

`CMMD_all_channels_free`	Tests whether all channels are free.
`CMMD_all_channels_free_wait`	Waits until all channels have been freed.

The sections below describe how all these functions are used. Section 8.3 provides reference pages for all these functions.

8.2.1 Establishing a Channel

To establish a channel,

- The source node calls `CMMD_open_send_channel` to specify the source array for the channel; this function returns a *send channel descriptor* (*scd*).

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

- The destination node calls `CMMD_open_receive_channel` to specify the destination array; this function returns a *receive channel descriptor* (*rcd*).

When both calls have been made, the channel is open, and is writable. The send and receive status functions have the following values:

```
CMMD_send_channel_status(scd)      ==   CMMD_Sendable
CMMD_receive_channel_status(rcd)   ==   CMMD_Unreadable
```

When either the source or the destination node has opened its end of the channel, and the other node has not yet followed suit, the channel is regarded as *half-open*, and cannot yet be used. The status values for this state are

```
CMMD_send_channel_status(scd)      ==   CMMD_O_send
CMMD_receive_channel_status(rcd)   ==   CMMD_O_receive
```

8.2.2 Sending Data through a Channel

To send data through a channel:

- The source node writes whatever data it wishes to send to the source array of the channel, and then calls `CMMD_write_channel(scd)`. (If the write succeeds, this function returns the integer 1.)

At this point, the data is sent through the channel and stored in the destination array. The channel now becomes *readable.* The send and receive status functions have the following values:

```
CMMD_send_channel_status(scd)      ==   CMMD_Sent
CMMD_receive_channel_status(rcd)   ==   CMMD_Readable
```

Optionally, before writing the channel, the source node can call `CMMD_is_channel_writable(scd)` to check whether it is writable; that is, that

```
CMMD_send_channel_status(scd)      ==   CMMD_Sendable
```


Copyright © 1995 Thinking Machines Corporation

8.2.3 Reading from and Resetting a Channel

To read from and reset a channel:

- The destination node calls `CMMD_is_channel_readable(`*rcd*`)` to confirm that the channel is readable; that is, that

  ```
  CMMD_receive_channel_status(rcd) == CMMD_Readable
  ```

- The destination node can then copy the data from the destination array, or simply use it in place.

- In either case, if the destination node expects to receive more data from the source node, it calls `CMMD_reset_channel(`*rcd*`)` to indicate that the channel is available for sending.

The channel now becomes writable again. The send and receive status functions have the following values:

```
CMMD_send_channel_status(scd)      ==   CMMD_Sendable
CMMD_receive_channel_status(rcd)   ==   CMMD_Unreadable
```

NOTE: If the destination node does not expect or intend to receive more data, it should *not* reset the channel. The channel should be reset only when more data is expected.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

8.2.4 Closing a Channel

To close a channel:

- The source node calls `CMMD_close_send_channel(`*scd*`)`.

  The send channel is now marked as *closed*; that is,

  ```
  CMMD_send_channel_status(scd) ==  CMMD_C_send
  ```

- The destination node calls `CMMD_close_receive_channel(`*rcd*`)`.

  The receive channel is now marked as *closed*; that is,

  ```
  CMMD_receive_channel_status(rcd) ==  CMMD_C_receive
  ```

- When both calls have been made, the channel is deallocated and freed for further use.

8.2.5 Testing for Completion of Channel Operations

To test whether all channels have been freed, any node can call

- `CMMD_all_channels_free()`

  This function returns 1 if all channels (including those in use by other nodes) have been freed, and returns 0 otherwise.

To wait until all channels have been freed, call

- `CMMD_all_channels_free_wait()`

  This function does not return until all channels have been freed. Thus, a node calling this function will block until *all* channels have been freed.

NOTE: This function is useful only on the send side of a channel, during the period after the sender has closed all its send channels and is waiting for the receiver to close all its. Calling it on the receive side of a channel may actually block the receiving node from being able to close the channel.

Copyright © 1995 Thinking Machines Corporation

WARNING

If any node calls `CMMD_all_channels_free_wait` and there is something the calling node has not yet done to allow a channel to be freed, the channel will never be freed, and so the call to `CMMD_all_channels_free_wait` will hang.

8.2.6 Restrictions on Channels

It is essential for the source and destination nodes of a channel to agree on the number of messages to be transmitted. An error (possibly fatal, crashing the program) may occur if the source sends more messages than the destination is prepared to receive, or if the destination node expects more messages than are sent by the source node.

A channel may not be closed by either the source or the destination node until at least one message has been sent through it.

Both the source and the destination nodes must cooperate in closing a channel; the source node must call the function `CMMD_close_send_channel` and the destination node must call `CMMD_close_receive_channel`. Once either of these functions has been called, the channel is no longer usable, but the channel is not actually reclaimed until both sender and receiver have closed it.

The destination node may not reset the channel after the source node has closed it. (Doing so has no harmful effect; it merely causes the channel status functions to report an error state.) If the destination node resets the channel, it is committed to receiving another message from the sender. When the channel is in a writable state, it is an error for the destination node to close its end of the channel. (Again, doing so has no harmful effect; it merely causes the channel status functions to report an error state.)

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

8.3 Channel Function Reference Pages

This section includes reference pages for the following functions:

```
CMMD_open_send_channel
CMMD_open_receive_channel
CMMD_send_channel_status
CMMD_receive_channel_status
CMMD_is_channel_writable
CMMD_is_channel_readable
CMMD_write_channel
CMMD_reset_channel
CMMD_close_send_channel
CMMD_close_receive_channel
CMMD_all_channels_free
CMMD_all_channels_free_wait
```


Copyright © 1995 Thinking Machines Corporation

CMMD_open_send_channel
CMMD_open_receive_channel

These functions open a virtual channel for sending or receiving.

SYNOPSIS

C Syntax

```
int CMMD_open_send_channel(int dest_node, int tag,
        void *buffer, int buffer_desc)
int CMMD_open_receive_channel(int source_node, int tag,
        void *buffer, int buffer_desc)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_open_send_channel(dest_node, tag,
        buffer, buffer_desc)

    INTEGER dest_node, tag, buffer_desc
    <ARRAY> buffer

INTEGER FUNCTION CMMD_open_receive_channel(source_node,
        tag, buffer, buffer_desc)

    INTEGER source_node, tag, buffer_desc
    <ARRAY> buffer
```

C* Syntax, for Parallel Arrays Only

```
int CMMD_open_send_channel(int dest_node, int tag,
        void:void *buffer, int array_element_length)

int CMMD_open_receive_channel(int dest_node, int tag,
        void:void *buffer, int array_element_length)
```

ARGUMENTS

dest_node	An integer identifying the node to which messages will be sent over this channel.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

source_node	An integer identifying the node from which messages will be sent over this channel.
tag	A non-negative integer (or CMMD_DEFAULT_TAG) that is the label for the messages to be sent (or received) over this channel.
buffer	The buffer that will contain messages sent or received.
buffer_desc	For serial arrays, a positive integer: the number of bytes in the buffer that are to be sent or received. For a CM Fortran parallel array, this should be the keyword CMMD_PARALLEL_ARRAY. For C* parallel arrays, this should be *array_element_length*, the positive length, in bytes, of a single data element.

RETURNS

These functions return immediately, with the following return values:

Nonnegative integer	ID of the send or receive channel.
CMMD_ERRVAL	Error; no channel resources are available.

DESCRIPTION

These functions open a many-use send or receive channel with the specified partner node. Array size, shape, and layout information is provided by the *buffer* and *buffer_desc* arguments. These *must* match for sender and receiver. Arrays may be either serial or parallel; *vector serial arrays are not supported*.

The following restrictions apply to buffers for channel operations:

- The receive buffer must be of the same size and shape as the send buffer. An attempt to establish a channel between two non-identical arrays will not produce a writable channel, and may produce an error.
- Buffers must have a positive length. It is an error to attempt to open a channel for zero-length messages.

Copyright © 1995 Thinking Machines Corporation

- For DPEAC parallel arrays, the buffer must be a doubleword-aligned memory address, and the number of elements in the array must be a positive integer that is a multiple of 4 bytes. (CM Fortran and C* parallel arrays will obey these rules automatically.)

  (If you want to tune for optimal performance, serial buffers should be either word-aligned or doubleword-aligned, with a length that is a multiple of 4 bytes. Note, however, that this is *not* a requirement.)

Callback and Handler Functions. Channel functions can be called from within callback and handler functions.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_send_channel_status
CMMD_receive_channel_status

These functions return the current status of the channel.

SYNOPSIS

C Syntax

```
int CMMD_send_channel_status(int scd)
int CMMD_receive_channel_status(int rcd)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_send_channel_status(scd)
    INTEGER   scd

INTEGER FUNCTION CMMD_receive_channel_status(rcd)
    INTEGER   rcd
```

ARGUMENTS

scd	Send channel descriptor.
rcd	Receive channel descriptor.

RETURNS

These routines return status values describing the status of the send channel and the receive channel, respectively. Possible return values for a send channel are

CMMD_O_send	The send channel has been opened by the sender, but the receive channel is not open on the receiver.
CMMD_Sendable	The send channel may be written to.
CMMD_Sent	The send channel may not be written to (because the receiving node has not yet reset the channel.)

Copyright © 1995 Thinking Machines Corporation

`CMMD_C_send`	The sender has marked the send channel for closing, but the receiver has not yet closed the channel.
`CMMD_C_receive`	The receiver has closed the receive channel, but the sender has not yet closed the send channel.
`CMMD_Free`	The channel is not allocated.
`CMMD_ERRVAL`	An error occurred.

Possible return values for a receive channel are

`CMMD_O_receive`	The receive channel has been opened by the receiver, but the send channel is not yet open on the sender.
`CMMD_Readable`	The receive channel buffer is ready to be read.
`CMMD_Unreadable`	The receive channel buffer is not ready to be read.
`CMMD_Close_error`	The sender has closed the send channel, and the receiver has tried to reset the channel.
`CMMD_Free`	The channel is not allocated.
`CMMD_ERRVAL`	An error occurred.

DESCRIPTION

These routines tell the nodes what status the channel is in. Sending nodes use them to find out whether the channel is ready for the next message to be sent; receiving nodes use them to find out whether a message has arrived.

- When a channel is newly created, the *`scd`* is `CMMD_Sendable` and the *`rcd`* is `CMMD_Unreadable`.
- When a channel has been written, the *`scd`* is `CMMD_Sent` and the *`rcd`* is `CMMD_Readable`.
- After the receiving node has (presumably) done useful work with the received message and has explicitly reset the channel, the *`scd`* is `CMMD_Sendable` and the *`rcd`* is `CMMD_Unreadable` again.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

- After both nodes have closed the channel, the channel is `CMMD_Free`.

NOTE: Status values other than those above can be returned, particularly while the channel is being established. However, these status values are transitory and implementation-dependent, and therefore of little use to programmers.

Callback and Handler Functions. Channel functions can be called from within callback and handler functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_is_channel_writable
CMMD_is_channel_readable

These functions tell whether a channel is readable or writable.

SYNOPSIS

C Syntax

```
int CMMD_is_channel_writable(int scd)
int CMMD_is_channel_readable(int rcd)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_is_channel_writable(scd)
    INTEGER   scd

INTEGER FUNCTION CMMD_is_channel_readable(rcd)
    INTEGER   rcd
```

ARGUMENTS

scd	Send channel ID.
rcd	Receive channel ID.

RETURNS

These functions return 1 if the send (or receive) channel is writable (or readable); they return 0 if it is not.

DESCRIPTION

These functions return true or false (1 or 0), depending on whether the channel is or is not writable or readable. Essentially,

`CMMD_is_channel_writable(`*scd*`)` is equivalent to
`CMMD_send_channel_status(`*scd*`) == CMMD_Sendable`

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

`CMMD_is_channel_readable(`*scd*`)` is equivalent to
`CMMD_send_channel_status(`*scd*`) == CMMD_Readable`

These functions do not return any specific error conditions: If the channel ID is out of range, or there is some error condition connected with the channel, the function only announces that the channel is not writable or readable.

Callback and Handler Functions. Channel functions can be called from within callback and handler functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_write_channel

Writes a message into a send channel.

SYNOPSIS

C Syntax

```
int CMMD_write_channel(int scd)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_write_channel(scd)
    INTEGER   scd
```

ARGUMENTS

scd	The send channel descriptor.

RETURNS

The function returns immediately, with one of the following values:

0	The write succeeded
1	The write failed — the channel was not ready for writing.
CMMD_ERRVAL	The write failed because the channel was not allocated, or was currently in the process of being opened or closed.

DESCRIPTION

CMMD_write_channel attempts to send an array via the specified channel to a destination node. The array size and shape, and the identity of the destination node, were specified when the channel was opened.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

Callback and Handler Functions. Channel functions can be called from within callback and handler functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_reset_channel

Resets a receive channel so it can receive a new message.

SYNOPSIS

C Syntax

```
int CMMD_reset_channel(int rcd)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_reset_channel(rcd)
    INTEGER   rcd
```

ARGUMENTS

rcd	Receive channel descriptor.

RETURNS

Returns immediately with one of the following values:

0	The specified channel is set up for another transaction.
1	The reset failed (the channel status is CMMD_Unreadable, so the channel cannot be reset).
CMMD_ERRVAL	The reset failed because the channel was not allocated, or has been closed by the source node.

DESCRIPTION

CMMD_reset_channel resets both the receive channel and the send channel associated with it. It marks the send channel's descriptor as CMMD_Sendable (thus indicating that the sending node may write a new message into the channel) and marks the receive channel's descriptor as CMMD_Unreadable (indicating that

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

the receiving node should not try to read the channel again until the sending node has written its next message).

Once the receiver has reset the channel, it must preserve the receive buffer until the sender has sent the next message. It is an error for the receiver to reset the channel and immediately either deallocate the buffer or close the channel.

Callback and Handler Functions. Channel functions can be called from within callback and handler functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_close_send_channel
CMMD_close_receive_channel

These functions close a virtual channel.

SYNOPSIS

C Syntax

```
int CMMD_close_send_channel(int scd)

int CMMD_close_receive_channel(int rcd)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_close_send_channel(scd)
    INTEGER   scd

INTEGER FUNCTION CMMD_close_receive_channel(int rcd)
    INTEGER   rcd
```

ARGUMENTS

scd	Send channel descriptor.
rcd	Receive channel descriptor.

RETURNS

These functions return immediately with one of the following values:

0	The channel has been closed.
CMMD_ERRVAL	The channel was not closed for one of these reasons: ▪ The channel had not been allocated. ▪ The channel was not in a closable state: either it was in the unreadable state that occurs before an array arrives, or it was still in the process of being established.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

DESCRIPTION

`CMMD_close_send_channel` and `CMMD_close_receive_channel` close the specified send and receive channels, respectively.

If the two nodes know in advance how many messages will travel over this channel, and keep count of the number sent and received, then they may close their respective channels independently. This is the preferred method for closing a channel, since it avoids potential program hangs.

NOTE: A channel may not be closed by either the source or the destination node until at least one message has been sent through it.

Callback and Handler Functions. Channel functions can be called from within callback and handler functions..

Copyright © 1995 Thinking Machines Corporation

CMMD_all_channels_free
CMMD_all_channels_free_wait

`CMMD_all_channels_free` tells whether all channels are free. `CMMD_all_channels_free_wait` blocks until all channels are free.

SYNOPSIS

C Syntax

```
int CMMD_all_channels_free(void)

void CMMD_all_channels_free_wait(void)
```

Fortran Syntax

```
INTEGER FUNCTION CMMD_all_channels_free()

SUBROUTINE CMMD_all_channels_free_wait()
```

RETURNS

`CMMD_all_channels_free` returns 1 if all channels have been freed, and 0 otherwise.

`CMMD_all_channels_free_wait` does not return a value.

DESCRIPTION

`CMMD_all_channels_free` returns immediately, and indicates by its value whether all channels have been freed (or, alternatively, whether any are still in use).

`CMMD_all_channels_free_wait` does not return until all channels have been freed; thus, any node that calls this function will block until all channels connected to this node have been freed.

Warning: If any node calls `CMMD_all_channels_free_wait` and there is something the calling node has not yet done to allow a channel to be freed, the channel will never be freed and the call to `CMMD_all_channels_free_wait` will hang.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

Callback and Handler Functions. Channel functions can be called from within callback and handler functions..

Copyright © 1995 Thinking Machines Corporation