Chapter 10

Synchronization

Synchronization occurs whenever two or more nodes wait for one another. This may be stated more precisely by using *A* and *B* to refer to programs running on two different nodes, and *X* and *Y* to refer to particular execution points in *A* and *B*, respectively. *A* and *B* are said to be *synchronized* at *X* and *Y* if *A* cannot proceed beyond *X* until *B* reaches *Y*, and *B* cannot proceed beyond *Y* until *A* reaches *X*. Some CMMD functions (for example, the cooperative send and receive functions) automatically synchronize two or more nodes. Other CMMD functions (for example, certain I/O routines, certain global functions) automatically synchronize *all* nodes. Sometimes it is possible for the application program to achieve synchronization indirectly, by using functions that are not inherently synchronizing (for example, by using active messages to build a synchronizing protocol; see Chapter 12).

Sometimes nodes *must* synchronize; other times they *should* synchronize; still other times they should *not* synchronize:

- Nodes must synchronize whenever program logic demands it. For example, if no node can logically (or safely) proceed to step 2 until all have completed step 1, then the nodes must synchronize at the end of step 1.

- Nodes should synchronize whenever performance is improved by operating in a synchronous, rather than asynchronous, fashion. If all nodes perform the same operation at the same time, performance can frequently be improved by having the nodes synchronize between operations.

- Nodes should synchronize when synchronization would contribute to ease of debugging a program that is under development. Adding extra synchronization to early versions of a program can greatly ease the debugging effort.

Copyright © 1995 Thinking Machines Corporation

- Nodes should not synchronize when efficiency can be gained by avoiding it, i.e., by *overlapping communication with computation* (see Section 2.1.1).

Programmers should remember that, in a message-passing program, the programmer is responsible for ensuring that synchronization happens when it is needed. Insufficient synchronization can cause program errors that are hard to track.

This chapter briefly reviews those operators that cause synchronization and offers full descriptions of three groups of functions used for explicit synchronization:

```
CMMD_sync_with_nodes

CMMD_sync_with_nodes_start
CMMD_sync_done
CMMD_sync_with_nodes_finish

CMMD_set_global_or
CMMD_get_global_or
```

Interrupts. If interrupts are enabled on a node that is waiting for a synchronization (such as a `sync_with_nodes` or a global operation), that node can be interrupted to receive a message. If interrupts are not enabled, all pending messages must wait until after the synchronization occurs.

10.1 Automatic Synchronization

Two or more nodes synchronize automatically when they participate in one of the blocking message-passing functions described in Chapter 3.

All nodes synchronize automatically when

- Your application begins or ends.
- Your application performs one of the global operations described in Chapter 9 (broadcasts, scans, reductions, concatenation).

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

- Your application performs global I/O (that is, it opens or closes a global file, or performs any I/O on files that are in synchronous broadcast or synchronous sequential mode), as described in Chapter 15.
- Your application performs certain system-level functions, such as `CMMD_reset_partition_size` (see Section 18.2) and `CMMD_global_suspend_servers` (see Section 19.8).

10.2 Explicit Synchronization

In addition to the automatic synchronization mentioned above, you can explicitly synchronize nodes in the following ways:

- At any reasonable point in your application, you can have the nodes call either the global function `CMMD_sync_with_nodes` or the pair of functions `CMMD_sync_with_nodes_start` and `CMMD_sync_with_nodes_finish`. Both are described below, in Section 10.3.

  Host/node programs use a variant of this call to include the host in the synchronization; see Section 19.4 for details.

- To synchronize the nodes at the completion of some sequence of computation or communications, you can use CMMD's global OR facility. Each node contributes a value asynchronously to this logical OR; any node can then read the value of the OR as a test for synchronization purposes. Section 10.3, below, describes the global OR facility and the two functions that access it.

  (Synchronization that uses the global OR, or any of the `_start` or `_finish` functions, is sometimes called a "soft" synchronization, since it allows the nodes to do useful things while "waiting" for the synchronization point.

- In order to force nodes to synchronize at the completion of an asynchronous communications pattern — that is, to synchronize on the emptying of the Data Network after some series of asynchronous send and receive calls — each node can use the functions `CMMD_all_msgs_done` (described in Section 7.5) or `CMMD_all_msgs_wait` (described in Section 7.6).

  To synchronize at the completion of a communications pattern that uses channels, the equivalent functions are `CMMD_all_channels_`

Copyright © 1995 Thinking Machines Corporation

`free` and `CMMD_all_channels_free_wait`. (See Section 8.2.5 for details.)

Note that these routines do not, themselves, synchronize the nodes. Rather, they are prerequisites to synchronization, either via an explicit synchronization call or via some other action that forces synchronization. For example, a node should never consider its share of a program finished until it knows it has received all messages intended for it.

10.3 Global Synchronization Functions

Global synchronization functions, as their name implies, serve to synchronize all nodes at a given point in a program.

CMMD offers several ways for nodes to synchronize. The first is an explicit synchronization function, to be called by all nodes at any point in a program.

`CMMD_sync_with_nodes`	Allows all nodes to synchronize themselves.

This function must be called by all nodes in the partition. Once any node has begun this synchronization call, the CM hardware keeps track of responses, and allows none of the calls to return until all nodes have made their call.

Next is a trio of functions that perform the same synchronization, but do so in a "split" timeframe.

`CMMD_sync_with_nodes_start`	Initiates a synchronization, but does not block pending completion.
`CMMD_sync_done`	Returns 1 if the synchronization is ready to complete, 0 otherwise.
`CMMD_sync_with_nodes_finish`	Blocks until the synchronization finishes, then returns.

In addition, two asynchronous global logical OR routines allow nodes to signal to each other by contributing to a global logical OR (*not* a bitwise OR) and reading its results.

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

`CMMD_set_global_or`	Sent by any node, this call adds a value (0 or nonzero) toward the creation of a global OR.
`CMMD_get_global_or`	Invoked by any node, this call reads the current value of the global OR.

By using the `CMMD_set_global_or` function, each processor contributes to the logical OR at an appropriate time; the hardware checks and updates the global value at frequent intervals; and individual processors read the value when desired. Thus, the global OR mechanism can be used as a nonblocking method of determining when all processors have reached a given state. All processors would start a task, for instance, by sending a 1. As each finished its share of the task, it would send a 0. By checking the value of the global OR (which would change to 0 only when all processors had finished), a processor could determine whether the whole task was complete and thus select its own next action.

Notes:

1. Using the global OR in this manner does require synchronizing at the beginning of the task to ensure that all nodes contribute and see a 1 before any node finishes its task and sends a 0.

2. There is some propagation delay between the time a node contributes a value and the time that contribution is reflected in the new value of the global OR.

3. These asynchronous global OR functions should not be confused with the synchronous global OR reduce operation, which is described in Section 9.2.

The remainder of the chapter contains the reference pages for these explicit synchronization functions.

Copyright © 1995 Thinking Machines Corporation

CMMD_sync_with_nodes

Global blocking function, causes all nodes to synchronize.

SYNOPSIS

C Syntax

```
void CMMD_sync_with_nodes(void)
```

Fortran Syntax

```
SUBROUTINE CMMD_sync_with_nodes()
```

RETURNS

This function does not return until all nodes in the partition have called the function. It has no return value.

DESCRIPTION

`CMMD_sync_with_nodes` synchronizes the calling node with all other nodes. Once any node has made this call, all nodes must eventually make the call; the function does not return until they all do.

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

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_sync_with_nodes_start
CMMD_sync_done
CMMD_sync_with_nodes_finish

Split synchronization function; `_start` is asynchronous, `_finish` blocks.

SYNOPSIS

C Syntax

```
void CMMD_sync_with_nodes_start(void)
int CMMD_sync_done(void)
void CMMD_sync_with_nodes_finish(void)
```

Fortran Syntax

```
SUBROUTINE CMMD_sync_with_nodes_start()
INTEGER FUNCTION CMMD_sync_done()
SUBROUTINE CMMD_sync_with_nodes_finish()
```

RETURNS

`CMMD_sync_with_nodes_start` returns immediately, with no return value.

`CMMD_sync_done` returns 1 if all nodes are ready to perform the synchronization, 0 otherwise.

`CMMD_sync_with_nodes_finish` does not return until all nodes in the partition have called the function. It has no return value.

DESCRIPTION

The global functions `CMMD_sync_with_nodes_start` and `CMMD_sync_with_nodes_finish` are used as a pair to synchronize all nodes.

`CMMD_sync_with_nodes_start` is asynchronous; it does not prevent the calling nodes from doing other work. `CMMD_sync_with_nodes_finish`, however, is a blocking function.

Copyright © 1995 Thinking Machines Corporation

Once one node has initiated the synchronization, all nodes must participate. `CMMD_sync_with_nodes_finish` does not return until all nodes have made the call and the synchronization has occurred.

`CMMD_sync_done` can be called to determine when all nodes have called `CMMD_sync_with_nodes_start`, and thus when it is reasonable to call `CMMD_sync_with_nodes_finish`.

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

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation

CMMD_set_global_or
CMMD_get_global_or

These functions contribute to and read a logical global OR.

SYNOPSIS

C Syntax

```
int CMMD_set_global_or(int value)
int CMMD_get_global_or(void)
```

Fortran Syntax

```
INTEGER FUNCTION  CMMD_set_global_or(value)
    INTEGER value

INTEGER FUNCTION  CMMD_get_global_or()
```

ARGUMENT

value	A logical TRUE or FALSE (i.e., an integer).

RETURN VALUES

Both these functions return the current value of the global OR. (This is a logical OR, *not* a bitwise OR.)

NOTE

If CMMD_set_global_or has not already set a value for the global OR, calls to CMMD_get_global_or return unpredictable results.

Note also that the value returned by CMMD_set_global_or itself is not guaranteed to include the value contributed by the function call.

Copyright © 1995 Thinking Machines Corporation

DESCRIPTION

`CMMD_set_global_or` allows any node to contribute a logical value (any integer) to a global *logical* OR. The new value replaces any previous value sent by the node. The function returns when the value has been sent; it does not wait for participation by any other node. It returns the current value of the global OR (either `TRUE` or `FALSE`).

Notice that since the function returns *immediately* upon contributing its argument to the global OR, its return value might not reflect the contribution of its own argument. For example, if all nodes contribute a `FALSE`, then wait for the Control Network to become quiet, it may be possible for a node to call this function with `TRUE` and still return `FALSE`.

`CMMD_get_global_or` returns the current value of the global OR function. It takes no argument, and thus contributes no value itself. If the node has not already contributed to the global OR using `CMMD_set_global_or`, then the return value of `CMMD_get_global_or` is unpredictable.

It should be emphasized that both of these functions are completely asynchronous; they do not require the participation of any other processors in order to return.

A typical use of these functions is to arrange for *any* node to notify *all* nodes that a certain condition has been met: All nodes initially contribute `FALSE` (and check to see that `FALSE` is returned everywhere), and then proceed with their work. Each node checks the global OR periodically (using `CMMD_get_global_or`), while also checking to see whether it meets the condition itself. If it does, it contributes `TRUE` (using `CMMD_set_global_or`). All nodes stop as soon as `CMMD_get_global_or` returns `TRUE`, for at that point they have a guarantee that at least one node has met the condition.

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

Version 3.3, April 1995
Copyright © 1995 Thinking Machines Corporation