Chapter 4

Cooperative Message-Passing Auxiliary Functions

This chapter describes the auxiliary CMMD functions that can be used with the cooperative message-passing functions. These include functions for:

- getting the self-address of a node and the size of its partition
- determining when there is a message waiting to be received
- getting information about the most recent message sent or received
- setting the internal message-matching function

4.1 Node Information Functions

Processors must address each other explicitly during message passing. They do so through node identifiers, which may be considered to be the logical addresses of the nodes. For each partition, the set of node identifiers consists of the integers from 0 to the number of nodes in the partition minus 1, inclusive.

A node can call **`CMMD_partition_size`** to determine the size of the current partition, and can call **`CMMD_self_address`** to determine its own position within the partition.

In host/node programming, a node can obtain the address of the host processor by calling **`CMMD_host_node`**. See Section 16.2 for information on this function.

FUNCTION **CMMD_self_address**

C SYNTAX

int **CMMD_self_address(**void**)**

FORTRAN SYNTAX

INTEGER FUNCTION **CMMD_self_address()**

RETURNS

CMMD_self_address returns an integer that represents the node identifier (the logical address) for the calling node.

DESCRIPTION

Called from a process running on a given node, **CMMD_self_address** returns the node identifier for that node.

Node identifiers are integers, from zero to the maximum number of processors in the partition, minus one, inclusive. For example, every 128-node partition contains nodes 0–127. Node identifiers are logical identifiers: programs and programmers need never concern themselves with physical processor addresses.

EXAMPLE

For example, a program preparing to do a number of circular shifts on a vector, where the vector spanned all nodes, might contain the following code:

```
if (CMMD_self_address() < (CMMD_partition_size() - 1))
         right_neighbor = CMMD_self_address() + 1;
    else
         right_neighbor = 0;

if (CMMD_self_address > 0)
         left_neighbor = CMMD_self_address() -1;
    else
         left_neighbor = CMMD_partition_size() - 1;
```

The circular shifts themselves might then be performed by having all nodes issue calls in which a node simultaneously receives a 16-byte message from its left-hand neighbor and sends its own 16-byte message to its right-hand neighbor:

```
CMMD_send_and_receive
    (left_neighbor,CMMD_DEFAULT_TAG, &coming, 16,
     right_neighbor, CMMD_DEFAULT_TAG, &going, 16)
```

FUNCTION **CMMD_partition_size**

C SYNTAX

int **CMMD_partition_size(**void**)**

FORTRAN SYNTAX

INTEGER FUNCTION **CMMD_partition_size()**

RETURNS

CMMD_partition_size returns the number of processors in the current partition.

DESCRIPTION

CMMD_partition_size returns the number of nodes in the current partition. By default, this is the size of the physical partition on which the application is running. However, an application can change the size of its "logical partition" by using the function **CMMD_reset_partition_size**. In this case, **CMMD_partition_size** returns the logical, not the physical, partition size.

Note: Some important rules and restrictions apply to using **CMMD_reset_partition_size**. Please read Section 15.1 thoroughly before using this function.

4.2 Message-Pending Function

The function **`CMMD_msg_pending`** tests whether there is a message waiting to be received. It returns 1 (true) if a message is available, and 0 (false) otherwise. Any node can call **`CMMD_msg_pending`** to test for the availability of a message, and then issue a **`receive_block`** or **`receive_async`** call only when it has detected that a message is waiting to be sent. Thus, the node can do some useful task, check for pending messages, do some other task, check again, and so on.

Calling **`CMMD_msg_pending`**, and then **`CMMD_receive_block`** if a pending message is detected, is similar to calling **`CMMD_receive_async`** — the receiving node doesn't have to block for unknown periods of time waiting for the sender. Also, if **`CMMD_msg_pending`** is used with **`CMMD_send_block`**, it allows synchronization of sending and receiving nodes at the time the message is sent.

FUNCTION **`CMMD_msg_pending`**

C SYNTAX

int **`CMMD_msg_pending(`**int *node*, int *tag***`)`**

FORTRAN SYNTAX

INTEGER FUNCTION **`CMMD_msg_pending(`***node, tag*)
    integer *node, tag*

ARGUMENTS

node	Integer identifying a node. (May be **`CMMD_ANY_NODE`**.)
tag	Integer identifying a tag. (May be **`CMMD_ANY_TAG`**.)

RETURNS

Returns 1 if a message is pending, 0 otherwise.

DESCRIPTION

`CMMD_msg_pending` returns 1 (true) if there is a message waiting to be received from the specified node (or from any node if **`CMMD_ANY_NODE`** is supplied as the node argument) with tag *tag* (or any tag if **CMMD_ANY_TAG** is supplied as the tag argument). It returns 0 (false) otherwise.

If **CMMD_ANY_NODE** is used, the function **CMMD_msg_sender()** can then be called to get the node identifier of the pending sender; if **CMMD_ANY_TAG** is used, **CMMD_msg_tag** returns the tag of the pending message.

4.3 Message Accessor Functions

CMMD provides four accessor functions for use with cooperative message-passing functions. These are the routines that tell you what really happened when you sent that message from one node to another: How much was sent or received? By what node was it sent? How was it tagged?

CMMD_bytes_received
CMMD_bytes_sent
CMMD_msg_sender
CMMD_msg_tag

The values these functions return are updated by each call to **CMMD_msg_pending**, and by calls to cooperative message-passing functions.

These accessor routines can be called at almost any point during a program, once message passing has begun. In particular, they can be called after a cooperative message-passing function has returned a positive result (signifying incomplete transmission or reception) to determine how much of the message was actually sent. These functions are also useful for getting information about messages received from **CMMD_ANY_NODE** or messages labeled with **CMMD_ANY_TAG**.

FUNCTIONS **CMMD_bytes_received**
CMMD_bytes_sent

C SYNTAX

int **CMMD_bytes_received(**void**)**
int **CMMD_bytes_sent(**void**)**

FORTRAN SYNTAX

INTEGER FUNCTION **CMMD_bytes_received()**
INTEGER FUNCTION **CMMD_bytes_sent()**

RETURNS

CMMD_bytes_received returns the number of bytes received by the calling node either in its most recent message or in the message detected by a call to **CMMD_msg_pending**.

CMMD_bytes_sent returns the number of bytes sent by the last message that was sent from the calling node.

FUNCTIONS **CMMD_msg_sender**
CMMD_msg_tag

C SYNTAX

int **CMMD_msg_sender(**void**)**
int **CMMD_msg_tag(**void**)**

FORTRAN SYNTAX

INTEGER FUNCTION **CMMD_msg_sender()**
INTEGER FUNCTION **CMMD_msg_tag()**

RETURNS

These functions return the node identifier or tag, respectively, for the last message received by the calling node, except when the calls are issued following a call to **`CMMD_msg_pending`**. In that case:

- If the call to **`CMMD_msg_pending`** returned true, **`CMMD_msg_sender`** returns the ID of the node waiting to send a message, and **`CMMD_msg_tag`** returns the tag of the waiting message.
- If the call to **`CMMD_msg_pending`** returned false, calling **`CMMD_msg_sender`** or **`CMMD_msg_tag`** returns the last node identifier or tag obtained from a real message (if any).

4.4 Message-Matching Function

CMMD uses an internal *matching function* to match up sending and receiving message-passing calls. You have the option of installing your own custom matching function, to allow your program to perform specialized patterns of messaging operations. For more information, see Section 15.1.