Appendix A

Sample Programs

This appendix contains a number of examples of CMMD programs. These programs, plus their makefiles, are also available on-line.

Examples provided in C are as follows:

- **round_trip:** Round-trip message (host/node).

  The host and all nodes synchronize clocks. Then the host sends a message (zero) to node 0, which receives it and adds one to it; then sends it to node 1, etc., until the message travels around all the nodes and comes back to the host. The host checks if the message is correct (equal to the number of nodes) and prints the result. In addition the host prints the time at which each node received the message.

- **ring:** One-dimensional ring (hostless).

  The nodes communicate in a ring pattern using the synchronous **CMMD_send_and_receive** function. At the end the communications rate is computed and printed.

- **redist1:** Global redistribution using synchronous functions (hostless).

  Node 0 (master) generates n = (**num_nodes**–1)*100 random integers in the range (1, **num_nodes**–1) and redistributes them in such a way that a given node receives only the numbers that are equal to its node address. The global asynchronous "or" flag is used to test for completion.

- **redist2:** Global redistribution using asynchronous functions (hostless).

  Each node generates n = 100 random integers in the range (0, **num_nodes**–1) and redistributes them in such way that a given node receives only the numbers that are equal to its node address. The global asynchronous "or" flag is used to test for completion.

- **redist3:** Global redistribution using asynchronous functions and a handler (hostless).

  Node 0 (master) generates n = (**num_nodes**–1)*100 random integers in the range (1, **num_nodes**–1) and redistributes them in such way that a given node receives only the numbers that are equal to its node address. A handler function counts the number of messages delivered. The global asynchronous "or" flag is used to test for completion.

- **sort:** Enumerate-pack sort (parallel radix) (host/node).

  This sort performs the following operations for each bit of the key: All the elements with a 0 in that bit position are enumerated with a scan with add. Then a second scan does the same for elements with a 1. A scan with max (downward) broadcasts the sum of elements with a 0 to all nodes. This sum is added to the result of the second scan and these values are the new node addresses for elements with a 1 in the bit position. The new node addresses for elements with a 0 is just the enumeration obtained from the first scan. The initial number in each node is randomly generated.

  Two Fortran 77 examples are also provided:

- **simple_io:** A host/node version of a "hello world" program, demonstrating the I/O service loop.

- **redist_block:** Global redistribution using synchronous functions (hostless)

  Node 0 (master) generates n = (**num_nodes**–1)*100 random integers in the range (1, **num_nodes**–1) and redistributes them in such way that a given node receives only the numbers that are equal to its node address. The global asynchronous "or" flag is used to test for completion.

A.1 C Examples

A.1.1 round_trip

Round-trip message (host program)

The host and all nodes synchronize clocks. Then the host sends a message (zero) to node 0, which receives it and adds one to it; then sends it to node 1 etc., etc., until the message travels around all the nodes and comes back to the host. The host checks if the message is correct (equal to the number of nodes) and prints the result. In addition each node prints the time at which it received the message.

```
#include <stdio.h>
#include <sys/types.h>
#include <cm/cmmd.h>
#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif

main ()
{ int msg_len=4, message, tag=0, i, num_nodes, pid = getpid();
  double times[1024];

  printf("\n Executing host program... (pid=%d)", pid);
  CMMD_enable();
  num_nodes=CMMD_partition_size();

  printf("\n Executing node programs...");
  CMMD_sync_host_with_nodes();   /* Global sync. to synchronize clocks */

  /* send message to node 0, receive message from node num_nodes-1 */

  message = 0;
  CMMD_send(0, tag, &message, msg_len);

  CMMD_receive(num_nodes-1, tag, &message, msg_len);

  /* Read timers */

  CMMD_gather_from_nodes(times, sizeof(double));

  if (message == num_nodes) {
    printf("\n All done ok, message = %d\n", message);
    for (i = 0; i < num_nodes; ++i)
      printf("\n Node %d: msg received ok, time= %f secs", i, times[i]);
  }
  else
    printf("\n Error: number of nodes = message = %d\n", message);

}
```

Round-trip message (node program)

```
#include <stdio.h>
#include <cm/cmmd.h>
#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif

main()
{ int src, dest, msg_len=4, message, tag=0, num_nodes, my_address;
  double time;

  CMMD_enable_host();

  /* Compute self address, source and destination */

  my_address = CMMD_self_address();
  num_nodes = CMMD_partition_size();
  src = (my_address == 0) ? CMMD_host_node() : my_address - 1;
  dest = (my_address == num_nodes-1) ? CMMD_host_node() : my_address + 1;

  /* Synchronize clocks */

  CMMD_sync_with_host();
  CMMD_node_timer_clear(0);
  CMMD_node_timer_start(0);

  /* Receive message  */

  CMMD_receive(src, tag, &message, msg_len);

  /* Get time */

  CMMD_node_timer_stop(0);
  time = CMMD_node_timer_elapsed(0);

  /* Add one to the message and send it to the next guy  */

  message++;
  CMMD_send(dest, tag, &message, msg_len);

  /* Host accumulates all times */

  CMMD_concat_elements_to_host(&time, sizeof(double));
}
```

A.1.2 ring

One-dimensional ring

The nodes communicate in a ring pattern using the synchronous **CMMD_send_and_receive** function. At the end the communications rate is computed and printed.

```
#include <stdio.h>
#include <cm/cmmd.h>
#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif

main()
{ int src, dest, msg_len=24000, tag=0, num_nodes, i, niter=10, my_address;
  char *msg_in, *msg_out;
  /* Get self address and number of nodes. Allocate msg. buffers */
  my_address = CMMD_self_address();
  num_nodes = CMMD_partition_size();
  msg_in = (char *) malloc(msg_len);
  msg_out = (char *) malloc(msg_len);
  /* set independent IO mode */
  CMMD_fset_io_mode(stdout, CMMD_independent);
  CMMD_fset_io_mode(stderr, CMMD_independent);
  if (my_address == 0) printf("\n Executing ring ...\n");
  /* Compute source and destination */
  src = my_address - 1;
  dest = my_address + 1;
  if (src < 0) src += num_nodes;
  if (dest >= num_nodes) dest -= num_nodes;
  /* Synchronous send and receive loop (repeat niter times) */
  CMMD_node_timer_clear(0);
  CMMD_node_timer_start(0);
  for (i = 0; i < niter; i++) {
    CMMD_send_and_receive(src, tag, msg_in, msg_len,
                          dest, tag, msg_out, msg_len);
    CMMD_sync_with_nodes();
  }
  CMMD_node_timer_stop(0);
  if (my_address == 0)  /* comm. rate = (total bytes in plus out)/time */
    printf("Time=%f [secs], Comm. Rate=%f [Mb/sec/node] \n",
           CMMD_node_timer_busy(0)/niter,
           2.0*niter*msg_len*1.e-6/CMMD_node_timer_busy(0));

}
```

A.1.3 redist_block

Global redistribution using synchronous (blocking) functions

Node 0 (master) generates n = (**num_nodes**–1)*100 random integers in the range (1, **num_nodes**–1) and redistributes them in such way that a given node receives only the numbers which are equal to its node address. The global asynchronous "or" flag is used to test for completion.

```
#include <stdio.h>
#include <cm/cmmd.h>
#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif

main(){ int msg_len=4, tag=0, i, n, rand_num, bufferin, recv_count=0, total;
  int my_address=CMMD_self_address(), num_nodes=CMMD_partition_size();

  /* set independent IO mode */

  CMMD_fset_io_mode(stdout, CMMD_independent);
  CMMD_fset_io_mode(stderr, CMMD_independent);

  if (my_address == 0) {          /* Node 0: sender node */
    printf("\n Executing redist_block ...\n");

    CMMD_set_global_or(1);        /* set global "or" to 1 (not completion) */
    while(!CMMD_get_global_or()); /* wait for setting */

    CMMD_sync_with_nodes();       /* ready to start */

    srand(1234);                  /* initialize RNG */

    n = (num_nodes-1)*100;        /* each node will get 100 msgs on average */

    for (i = 0; i < n; i++) {     /* send n random numbers to nodes */

     rand_num = 1 + (rand() % (num_nodes-1)); /* generate random number */

      CMMD_send(rand_num, tag, &rand_num, msg_len);
    }

    CMMD_set_global_or(0);        /* set global "or" to 0 (completion) */
    printf("\n Total msgs generated (node 0)= %8d\n", n);
    total = CMMD_reduce_int(recv_count, CMMD_combiner_add);
    printf("\n Total msgs received by nodes (1-- num_nodes-1)= %8d\n", total);

  }
  else {                          /* Nodes (1 -- num_nodes-1): recipients */

    CMMD_set_global_or(0);        /* node 0 will control completion */

    CMMD_sync_with_nodes();       /* ready to start */
```

```
    while(CMMD_get_global_or()) { /* loop until all done  */

      if (CMMD_msg_pending(CMMD_ANY_NODE, tag)) { /* poll for incoming msgs */
    CMMD_receive(0, tag, &bufferin, msg_len);
    recv_count++;
      }
    }

    /* print results */

    printf("PN %4d, msgs received = %6d \n", my_address, recv_count);
    total = CMMD_reduce_int(recv_count, CMMD_combiner_add);
  }
}
```

A.1.4 redist_nonblock

Global redistribution using asynchronous (nonblocking) functions

Each node generates n = 100 random integers in the range (0, **num_nodes**–1) and redistributes them in such way that a given node receives only the numbers which are equal to its node address. The global asynchronous "or" flag is used to test for completion.

```
#include <stdio.h>
#include <cm/cmmd.h>
#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif

main()
{ int msg_len=4, tag=0, rand_num, send_ok, n, recv_ok, bufferin;
  int send_count=0, recv_count=0, send_done=0, total;
  int my_address=CMMD_self_address(), num_nodes=CMMD_partition_size();

  /* set independent IO mode */

  CMMD_fset_io_mode(stdout, CMMD_independent);
  CMMD_fset_io_mode(stderr, CMMD_independent);
  if (my_address == 0) printf("\n Executing redist_nonblock ...\n");

  n = 100;                        /* each node will get 100 msgs. on average */

  srand(my_address);              /* initialize RNG */
```

```
  CMMD_set_global_or(1);          /* set global "or" to 1 (not completion) */
  while(!CMMD_get_global_or());   /* wait for setting to take place */

  /* send and receive loop */

  while(CMMD_get_global_or()) {   /* is everybody done ? */

    if (send_count < n) {         /* more sending ? */
      rand_num = (rand() >> 10) % num_nodes; /* generate random number */

      send_ok = CMMD_send_noblock(rand_num, tag, &rand_num, msg_len, NULL);
      if (send_ok == 0) send_count++;
    }
    send_done = (send_count == n);     /* sending done */

    if (CMMD_msg_pending(CMMD_ANY_NODE, tag)) {  /* poll for incoming msgs */

      recv_ok = CMMD_receive(CMMD_ANY_NODE, tag, &bufferin, msg_len);
      if (recv_ok == 0) {
    recv_count++;
    if (bufferin != my_address) {
      printf(" Error: received wrong msg: %d instead of %d\n",
          bufferin, my_address);
    }
      }
    }
    if (send_done && CMMD_all_msgs_done()) /* am I done ? */
      CMMD_set_global_or(0);
  }

  /* print results */

  printf("PN %4d, msgs received = %6d \n", my_address, recv_count);
  total = CMMD_reduce_int(recv_count, CMMD_combiner_add);
  if (my_address == 0) {
    printf("\n Total msgs generated (all nodes) = %8d\n", n*num_nodes);
    printf("\n Total msgs received (all nodes) = %8d\n", total);
  }
}
```

A.1.5 redist_nonblk_hndlr

Global redistribution using asynchronous (nonblocking) functions and a handler

Node 0 (master) generates n = (**num_nodes**–1)*100 random integers in the range (1, **num_nodes**–1) and redistributes them in such way that a given node receives only the numbers which are equal to its node address. A handler function counts the number of messages delivered. The global asynchronous "or" flag is used to test for completion.

```
#include <stdio.h>
#include <cm/cmmd.h>

#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif

#ifdef __STDC__
volatile
#endif
  int msgs_delivered = 0;

main()
{
  int tag=0, i, n, bufferin, recv_count=0, total;
  int my_address=CMMD_self_address(), num_nodes=CMMD_partition_size();
  CMMD_mcb mcb;
  int *rand_num;        /* Where to put the random nums.  */

  void my_handler();

  /* set independent IO mode */
  CMMD_fset_io_mode(stdout, CMMD_independent);
  CMMD_fset_io_mode(stderr, CMMD_independent);

  if (my_address == 0) {        /* Node 0: sender node */
    printf("\n Executing redist_nonblk_hndlr ...\n");
    CMMD_set_global_or(1);      /* set global "or" to 1 (not completion) */
    while(!CMMD_get_global_or()); /* wait for setting */
    CMMD_sync_with_nodes();     /* ready to start */
    srand(63924);               /* initialize RNG */
    n = (num_nodes-1)*100;      /* each node will get 100 msgs on average */

    rand_num = (int *) malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {   /* send n random numbers to nodes */
     rand_num[i] = 1 + (rand() % (num_nodes-1)); /* generate random number */
     CMMD_send_async(rand_num[i], tag, &rand_num[i], sizeof(rand_num[i]),
              my_handler, 0);
    }

    /* the handler takes care of counting the number of msgs. being
       delivered */
    while(msgs_delivered != n);  /* wait until all msgs have been delivered */

    CMMD_set_global_or(0);      /* set global "or" to 0 (completion) */
    printf("\n Total msgs generated (node 0)= %8d\n", n);
    total = CMMD_reduce_int(recv_count, CMMD_combiner_add);
    printf("\n Total msgs received by nodes (1-- num_nodes-1)= %8d\n", total);
  }
```

```
  else {                         /* Nodes (1 -- num_nodes-1): recipients */
    CMMD_set_global_or(0);       /* node 0 will control completion */
    CMMD_sync_with_nodes();      /* ready to start */
    while(CMMD_get_global_or()) { /* loop until all done  */
      if (CMMD_msg_pending(CMMD_ANY_NODE, tag)) { /* poll for incoming msgs */
        CMMD_receive_block(0, tag, &bufferin, sizeof(bufferin));
        recv_count++;
      }
    }
    /* print results */

    printf(”PN %4d, msgs received = %6d \n”, my_address, recv_count);
    total = CMMD_reduce_int(recv_count, CMMD_combiner_add);
  }
}

void my_handler(mcb, arg)
CMMD_mcb *mcb;
int arg;
{
  /* free the mcb and add one to the counter of msgs. delivered */

  CMMD_free_mcb(*mcb);
  msgs_delivered++;
}
```

A.1.6 sort

Enumerate-pack sort (parallel radix)

Here’s an example of a CMMD program written in the host/node programming style. This sort example performs the following operations for each bit of the key: All the elements with a 0 in that bit position are enumerated with a scan with add. Then a second scan does the same for elements with a 1. A scan with max (downward) broadcasts the sum of elements with a 0 to all nodes. This sum is added to the result of the second scan and these values are the new node addresses for elements with a 1 in the bit position. The new node addresses for elements with a 0 is just the enumeration obtained from the first scan. The initial number in each node is randomly generated.

Host Program:

```
#include <stdio.h>
#include <sys/types.h>
#include <cm/cmmd.h>

#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif#

main ()
{ int *numbers, *sorted, i, num_nodes, pid = getpid();

  printf("\n Executing host program... (pid=%d)", pid);

  CMMD_enable();
  num_nodes = CMMD_partition_size();

  numbers = (int *) malloc (num_nodes*sizeof(int));
  sorted = (int *) malloc (num_nodes*sizeof(int));

  CMMD_gather_from_nodes(numbers, sizeof(int));

  CMMD_gather_from_nodes(sorted, sizeof(int));

  printf("\n\n %8s   %11s\n", "numbers:", "after sort:");

  for (i = 0; i < num_nodes; i++)
      printf("\n %8d   %11d", numbers[i], sorted[i]);

  printf("\n All done \n");

}
```

Node Program

```
#include <stdio.h>
#include <cm/cmmd.h>

#if CMMD_VERSION <= 20
#include <cm/cmmd-io.h>
#endif

main()
{ int seed = 1234, msg_len=4, tag=0, tempo, number, k=0;
  int my_address=CMMD_self_address(), num_nodes=CMMD_partition_size();
  int mask_zero, mask_one, enum_one, enum_zero, sum, new_pos;

  CMMD_enable_host();

  srand(my_address*seed);                   /* initialize RNG */
  number = (rand() >> 10) % num_nodes;      /* generate random number */

  CMMD_concat_elements_to_host(&number, sizeof(int));
```

```
for (k = 0; 1 << k < num_nodes; k++) {
  mask_one = number >> k & 1;
  mask_zero = ~(number >> k) & 1;
  enum_one = CMMD_scan_int(mask_one, CMMD_combiner_add, CMMD_upward,
                 CMMD_none, NULL, CMMD_inclusive) - 1;

  enum_zero = CMMD_scan_int(mask_zero, CMMD_combiner_add, CMMD_upward,
                 CMMD_none, NULL, CMMD_inclusive) - 1;

  sum = CMMD_scan_int(enum_zero + 1, CMMD_combiner_max, CMMD_downward,
                 CMMD_none, NULL, CMMD_inclusive);

  enum_one += sum;
  new_pos = enum_one*mask_one + enum_zero*mask_zero;

  CMMD_send_and_receive(CMMD_ANY_NODE, tag, &tempo, msg_len,
              new_pos, tag, &number, msg_len);
  number = tempo;
}

CMMD_concat_elements_to_host(&number, sizeof(int));
```

A.2 Fortran 77 Examples

Global redistribution using synchronous (blocking) functions

Node 0 generates *n* random integers in the range (0, **num_modes**–1) and redistributes them in such way that a given node receives only the numbers which are equal to its node address. A global asynchronous “or” flag is used to test for completion.

```
      program redist_block

      include ”/usr/include/cm/cmmd_fort.h”

      integer n, msg_len, tag, i, rand_num, bufferin, seed, total
      parameter(tag = 0, msg_len = 4, n = 1000)
      integer my_address, num_nodes, msg_ok, ret_val, recv_count

      total =  cmmd_set_io_mode(6, cmmd_independent)
      my_address = cmmd_self_address()
      num_nodes = cmmd_partition_size()
      recv_count = 0

c      node 0: sender node

      if (my_address.eq.0) then

c     set global ”or” to 1 (not completion)

         ret_val = cmmd_set_global_or(1)

c        wait for setting
         do while (cmmd_get_global_or().eq.0)
         end do

c        ready to start
         call cmmd_sync_with_nodes()

         seed = my_address

c        distribute (send) random numbers to nodes

         do i = 1, n
c           generate random number
            rand_num = 1 + int((num_nodes-1)*rand(seed))
            msg_ok = cmmd_send(rand_num, tag, rand_num, msg_len)
         end do
```

```
c         set global ”or” to 0 (completion)
         ret_val = cmmd_set_global_or(0)
         print *,” Total msgs generated (node 0)= ”, n
         total = cmmd_reduce_int(0, CMMD_combiner_add)
         print *,”Total msgs received by nodes (1--num_nodes-1)=”,total
         call flush(6)

      else

c      nodes (1 -- num_nodes-1): recipients

c    node 0 will control completion

         ret_val = cmmd_set_global_or(0)

c    ready to start
         call cmmd_sync_with_nodes()

c        loop until all done
         do while (cmmd_get_global_or().ne.0)
c         poll for incoming msgs
            if (cmmd_msg_pending(CMMD_ANY_NODE, tag).ne.0) then
               mesg_ok = cmmd_receive(0, tag, bufferin, msg_len)
               recv_count = recv_count + 1
            end if
         end do
         print *, ”PN”,  my_address, ”  msgs received = ”, recv_count
         total = cmmd_reduce_int(recv_count, CMMD_combiner_add)

      end if

      call flush(6)
      end
```

Simple I/O

Here’s an example of a host/node program written in CM Fortran:

Host Program:

```
 program simple_io
      include ”/usr/include/cm/cmmd_fort.h”

      integer nodes, i, data, msg_ok

      call cmmd_enable()
      nodes = cmmd_partition_size()
      print *, ”start”
```

```
      call cmmd_bc_from_host(nodes, 4)

      call cmmd_service_loop()

      do 100 i = 0, nodes-1
      msg_ok =  cmmd_receive(i,0,data,4)
      print *, "PE ", i, " Sent ", data
 100  continue
      end
```

Node Program:

```
      program simple_io
      integer data, msg_ok, ret_code, dummy
      integer host

      include "/usr/include/cm/cmmd_fort.h"

      call cmmd_enable_host()

      call cmmd_receive_bc_from_host(dummy, 4)

      ret_code = cmmd_set_open_mode(CMMD_independent)
      write (*, *) "hello world"
      call cmmd_global_suspend_servers()

      data = cmmd_self_address() * 2
      host = cmmd_host_node()
      msg_ok =  cmmd_send(host,0,data,4)
      end
```