In this second article on MPI (Message Passing Interface), we look at a group of important commands that enable communication via messages between the nodes. There are two groups of communication commands or operations: point-to-point and collective (broadcast). MPI point-to-point operations typically involve message passing between only two different MPI processes or tasks. One task is performing a send operation and the other task is performing a matching receive operation. Collective communication must involve all processes in the scope of a communicator. All processes are by default members in the communicator MPI_COMM_WORLD.

In this article, we look at the point-to-point operations. MPI has a rich set of send and receive commands and we will take a closer look at the most commonly used ones. The difference between them lies in buffering (where the data is kept until it is received) and synchronisation (when an operation completes).

Blocking Send & Receive

MPI_Send (buf, count, datatype, dest, tag, comm)
This is the basic blocking send operation. The routine returns only after the send buffer is free for reuse by the sending task. Note that this routine may be implemented differently on different systems. The MPI standard permits the use of a system buffer but does not require it. Some implementations may actually use a synchronous send (another mode not discussed here) to implement the basic blocking send.

MPI_Recv (buf, count, datatype, source, tag, comm, status)
This is the basic blocking receive operation. The routine blocks until the requested message data is stored in the receive buffer.

As a result, messages are non-overtaking. Messages are received in order of sends if processes are single-threaded and the wildcard MPI_ANY_SOURCE is not used in receives. The following example illustrates:

/* ------------------------------------------------------------------
* ring.c
* This MPI program sends a series of messages in a ring.
*/
* The master task sends a message, waits for message to complete
* ring before repeating process.
* 
* *------------------------------------------------------------------
* 
* /

#include <mpi.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
  float x=3.34,y=999;
  int tag=1,size,rank,d;
  MPI_Status status;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  printf("size %d rank %d\n",size, rank);

  if ( rank == 0 ) {
    for ( d = 0; d < size ; d++ ) {
      x=x+(float)d;
      MPI_Send(&x,1,MPI_FLOAT,1,tag,MPI_COMM_WORLD);
      printf("master sending %f \n",x);

      MPI_Recv(&y,1,MPI_FLOAT,size-1,tag,MPI_COMM_WORLD,&status);
      printf("master receiving %f \n",y);
    }
  } else {
    MPI_Recv(&y,1,MPI_FLOAT,rank-1,tag,MPI_COMM_WORLD,&status);
    printf("rank %d receiving, x = %f \n",rank,y);

    MPI_Send(&y,1,MPI_FLOAT,(rank+1)%size,tag,MPI_COMM_WORLD);
  }

  MPI_Finalize();
  return 0;
}

Sample output:

rank 3 size 4
rank 1 size 4
rank 1 receiving, x = 3.340000
rank 2 size 4
rank 2 receiving, x = 3.340000
rank 0 size 4
master sending 3.340000
rank 3 receiving, x = 3.340000
master receiving 3.340000
master sending 4.340000
rank 1 receiving, x = 4.340000
rank 2 receiving, x = 4.340000
rank 3 receiving, x = 4.340000
Non-Blocking Send & Receive

MPI_Isend (buf, count, datatype, dest, tag, comm, request)
Basic non-blocking send identifies an area in memory to serve as a send buffer. Processing continues immediately without waiting for the message to be copied out from the buffer. A communication request handle is returned for handling the pending message status. The program should not modify the application buffer until subsequent calls to MPI_Wait or MPI_Test indicate that the non-blocking send has been completed.

MPI_Irecv (buf, count, datatype, source, tag, comm, request)
Basic non-blocking receive identifies an area in memory to serve as a receive buffer. Processing continues immediately without actually waiting for the message to be received and copied into the buffer. A communication request handle is returned for handling the pending message status. The program must use calls to MPI_Wait or MPI_Test to determine when the non-blocking receive operation is completed and the requested message is available in the buffer.

Using MPI_Isend and MPI_Irecv without MPI_Wait and MPI_Test in the example above, the sample output shows evidence of overtaking messages where sends and receives are not matched:

```
rank 1 receiving, x = 6.340000
rank 2 receiving, x = 6.340000
master receiving 4.340000
master sending 6.340000
rank 3 receiving, x = 6.340000
master receiving 6.340000
master sending 9.340000
rank 1 receiving, x = 9.340000
rank 2 receiving, x = 9.340000
rank 3 receiving, x = 9.340000
master receiving 9.340000
```

```
Summary
These are the main send and receive modes but there are many variations. For details on other modes not covered here, please refer to the online guides: http://www.llnl.gov/computing/tutorials/mpi/

For details on submitting MPI jobs to SVU clusters through the LSF scheduler, please refer to SVU technical info page.

The previous article, Using MPI on Atlas, introduces the basic MPI commands.

For questions or feedback, please contact us at ccsvuhelp@nus.edu.sg.