1 MPI Hands-On; Sending and Receiving Messages II

1. The code3.c consists of one receiver process and N-1 sender processes.

   - The sender processes send a message consisting of their process identifier (id) and the total number of processes (ntasks) to the receiver.
   - The receiver process prints out the values it receives in the messeges from the senders.

```c
/* A simple SPMD example program using MPI                          */

/* The program consists of on receiver process and N-1 sender       */
/* processes. The sender processes send a message consisting        */
/* of their process identifier (id) and the total number of         */
/* processes (ntasks) to the receiver. The receiver process         */
/* prints out the values it receives in the messeges from the       */
/* senders.                                                         */

/* Compile the program with 'mpicc code3.c -o code3'                */
/* To run the program, using four of the computers specified in     */
/* your hostfile, do 'mpirun -machinefile mf.txt -np 4 code3        */
/* An example mf.txt is just containing the following lines         */
/* lecture.ikcu.edu.tr                                              */
/* lecture.ikcu.edu.tr                                              */
/* lecture.ikcu.edu.tr                                              */
/* lecture.ikcu.edu.tr                                              */

#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  const int tag = 42;          /* Message tag */
  int id, ntasks, source_id, dest_id, err, i;
  MPI_Status status;
  int msg[2];      /* Message array */

  err = MPI_Init(&argc, &argv); /* Initialize MPI */
  if (err != MPI_SUCCESS) {
    printf("MPI initialization failed!\n");
    exit(1);
  }
  err = MPI_Comm_size(MPI_COMM_WORLD, &ntasks); /* Get nr of tasks */
  err = MPI_Comm_rank(MPI_COMM_WORLD, &id); /* Get id of this process
    */
  if (ntasks < 2) {
    printf("You have to use at least 2 processors to run this program\
    n");
    MPI_Finalize();       /* Quit if there is only one processor */
    exit(0);
  }

  if (id == 0) {    /* Process 0 (the receiver) does this */
    for (i=1; i<ntasks; i++) {
```

```
44      err = MPI_Recv(msg, 2, MPI_INT, MPI_ANY_SOURCE, tag,
    MPI_COMM_WORLD, &status);          /* Receive a message */
45      source_id = status.MPI_SOURCE;  /* Get id of sender */
46      printf("Received message %d of %d from process %d\n", msg[0],
    msg[1], source_id);
47    }
48  }
49  else {      /* Processes 1 to N-1 (the senders) do this */
50    msg[0] = id;     /* Put own identifier in the message */
51    msg[1] = ntasks;          /* and total number of processes */
52    dest_id = 0;     /* Destination address */
53    err = MPI_Send(msg, 2, MPI_INT, dest_id, tag, MPI_COMM_WORLD);
54  }
55
56  err = MPI_Finalize();          /* Terminate MPI */
57  if (id==0) printf("Ready\n");
58  exit(0);
59 }
```

2. **Sending in a ring**. A code4.c that takes data from process zero and sends it to all of the other processes by sending it in a ring.

   - That is, process **i** should receive the data and send it to process **i+1**, until the last process is reached.
   - Assume that the data consists of a single integer. Process zero reads the data from the user.

```
1  #include <stdio.h>
2  #include "mpi.h"
3
4  int main(int argc, char **argv)
5  {
6      int rank, value, size;
7      MPI_Status status;
8
9      MPI_Init( &argc, &argv );
10
11     MPI_Comm_rank( MPI_COMM_WORLD, &rank );
12     MPI_Comm_size( MPI_COMM_WORLD, &size );
13     do {
14   if (rank == 0) {
15       scanf( "%d", &value );
16       MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
17   }
18   else {
19       MPI_Recv( &value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD, &
     status );
20       if (rank < size - 1)
21     MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
22   }
23   printf( "Process %d got %d\n", rank, value );
24     } while (value >= 0);
25
26     MPI_Finalize( );
27     return 0;
28 }
```

3. Analyse the example code5.c for sending/receiving.

```c
/*
   ******************************************************************************
 * FILE: mpl.ex1.c
 * DESCRIPTION:
 *   In this simple example, the master task initiates numtasks-1 number of
 *   worker tasks.  It then distributes an equal portion of an array to each
 *   worker task.  Each worker task receives its portion of the array, and
 *   performs a simple value assignment to each of its elements. The value
 *   assigned to each element is simply that element's index in the array+1.
 *   Each worker task then sends its portion of the array back to the master
 *   task.  As the master receives back each portion of the array, selected
 *   elements are displayed.
 * AUTHOR: Blaise Barney
 * LAST REVISED:  09/14/93 for latest API changes  Blaise Barney
 * LAST REVISED:  01/10/94 changed API to MPL      Stacy Pendell
 * CONVERTED TO MPI: 11/12/94 by                   Xianneng Shen

   ******************************************************************************
   */

#include <stdio.h>
#include "mpi.h"
#define ARRAYSIZE       60000
#define MASTER          0       /* taskid of first process */

MPI_Status status;
main(int argc, char **argv)
{
  int   numtasks,               /* total number of MPI process in
    partitiion */
    numworkers,         /* number of worker tasks */
    taskid,                     /* task identifier */
    dest,                       /* destination task id to send message
     */
    index,                      /* index into the array */
    i,                  /* loop variable */
    arraymsg = 1,               /* setting a message type */
    indexmsg = 2,               /* setting a message type */
    source,                     /* origin task id of message */
    chunksize;          /* for partitioning the array */
  float data[ARRAYSIZE],        /* the intial array */
    result[ARRAYSIZE];  /* for holding results of array operations */

  /************************ initializations
    ***********************************
   * Find out how many tasks are in this partition and what my task id
    is.  Then
   * define the number of worker tasks and the array partition size as
    chunksize.
   * Note:  For this example, the MP_PROCS environment variable should
    be set
   * to an odd number...to insure even distribution of the array to
```

```
    numtasks-1
44   * worker tasks.
45
     ************************************************************************
     */
46  MPI_Init(&argc, &argv);
47  MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
48  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
49  numworkers = numtasks-1;
50  chunksize = (ARRAYSIZE / numworkers);
51
52  /*************************** master task
      ************************************/
53  if (taskid == MASTER) {
54    printf("\n************ Starting MPI Example 1 ************\n");
55    printf("MASTER: number of worker tasks will be= %d\n",numworkers);
56    fflush(stdout);
57
58    /* Initialize the array */
59    for(i=0; i<ARRAYSIZE; i++)
60      data[i] =  0.0;
61    index = 0;
62
63    /* Send each worker task its portion of the array */
64    for (dest=1; dest<= numworkers; dest++) {
65      printf("Sending to worker task= %d\n",dest);
66      fflush(stdout);
67      MPI_Send(&index, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
68      MPI_Send(&data[index], chunksize, MPI_FLOAT, dest, 0,
    MPI_COMM_WORLD);
69      index = index + chunksize;
70    }
71
72    /* Now wait to receive back the results from each worker task and
      print */
73    /* a few sample values */
74    for (i=1; i<= numworkers; i++) {
75      source = i;
76      MPI_Recv(&index, 1, MPI_INT, source, 1, MPI_COMM_WORLD, &status)
    ;
77      MPI_Recv(&result[index], chunksize, MPI_FLOAT, source, 1,
    MPI_COMM_WORLD,
78               &status);
79
80      printf("-------------------------------------------------------\n");
81      printf("MASTER: Sample results from worker task = %d\n",source);
82      printf("    result[%d]=%f\n", index, result[index]);
83      printf("    result[%d]=%f\n", index+100, result[index+100]);
84      printf("    result[%d]=%f\n\n", index+1000, result[index+1000]);
85      fflush(stdout);
86    }
87
88    printf("MASTER: All Done! \n");
89  }
90
91
92  /*************************** worker task
      ************************************/
93  if (taskid > MASTER) {
94    /* Receive my portion of array from the master task */
95    source = MASTER;
96    MPI_Recv(&index, 1, MPI_INT, source, 0, MPI_COMM_WORLD, &status);
```

```
97      MPI_Recv(&result[index], chunksize, MPI_FLOAT, source, 0,
98               MPI_COMM_WORLD, &status);
99      /* Do a simple value assignment to each of my array elements */
100     for(i=index; i < index + chunksize; i++)
101       result[i] = i + 1;
102
103     /* Send my results back to the master task */
104
105     MPI_Send(&index, 1, MPI_INT, MASTER, 1, MPI_COMM_WORLD);
106     MPI_Send(&result[index], chunksize, MPI_FLOAT, MASTER, 1,
        MPI_COMM_WORLD);
107
108   }
109   MPI_Finalize();
110 }
```