MONASH University

MPI VIRTUAL TOPOLOGY AND MASTER & SLAVE

[Sample Solutions]

OBJECTIVES

- The purpose of this lab is to explore MPI Virtual Topologies and Master/Slave programs by splitting a communicator.

INSTRUCTIONS

- Download and set up the Linux environment on Docker [Refer to Week 0]
- Lab materials of Week 09 will be assessed in the form of Moodle Quiz on Week 10. Please review the materials and prepare well for the Lab Quiz on Week 10.

TASK

DESCRIPTION:

- Practice inter-process communication using MPI virtual topology functions.
- Design and implement a simple virtual topology.
- Splitting a communicator for master and slave operations.

MONASH University

LAB ACTIVITIES

Task 1 - Creating a 2D Cartesian grid using MPI Worked Example

	0	1	2	3	4
0	Rank0 (0,0)	Rank1 (0,1)	Rank2 (0,2)	Rank3 (0,3)	Rank4 (0,4)
1	Rank5 (1,0)	Rank6 (1,1)	Rank7 (1,2)	Rank8 (1,3)	Rank9 (1,4)
2	Rank10 (2,0)	Rank11 (2,1)	Rank12 (2,2)	Rank13 (2,3)	Rank14 (2,4)
3	Rank15 (3,0)	Rank16 (3,1)	Rank17 (3,2)	Rank18 (3,3)	Rank19 (3,4)

Figure 1: Cartesian grid layout

With reference to Figure 1, create a 2D grid using MPI Cartesian topology functions. Each MPI process in the grid is to print out the following:

a) Current rank
b) Cartesian rank
c) Coordinates
d) List of immediate adjacent processes (left, right, top and bottom) The user has an option to specify the grid size as a command line argument.

Sample solution:

```
/* Gets the neighbors in a cartesian communicator
*    Orginally written by Mary Thomas
*    - Updated Mar, 2015
*    Link: https://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-Cart-Comms-and-Topos.pdf
*    Minor modifications to fix bugs and to revise print output
*/
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <mpi.h>

#define SHIFT_ROW 0
#define SHIFT_COL 1
#define DISP 1
```

```
int main(int argc, char *argv[]) {

    int ndims=2, size, my_rank, reorder, my_cart_rank, ierr;
    int nrows, ncols;
    int nbr_i_lo, nbr_i_hi;
    int nbr_j_lo, nbr_j_hi;
    MPI_Comm comm2D;
    int dims[ndims],coord[ndims];
    int wrap_around[ndims];

    /* start up initial MPI environment */
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* process command line arguments*/
    if (argc == 3) {
        nrows = atoi (argv[1]);
        ncols = atoi (argv[2]);
        dims[0] = nrows; /* number of rows */
        dims[1] = ncols; /* number of columns */
        if( (nrows*ncols) != size) {
            if( my_rank ==0) printf("ERROR: nrows*ncols)=%d * %d =  %d
!= %d\n", nrows, ncols, nrows*ncols,size);
            MPI_Finalize();
            return 0;
        }
    } else {
        nrows=ncols=(int)sqrt(size);
        dims[0]=dims[1]=0;
    }

    /*************************************************************
    */
    /* create cartesian topology for processes */
    /*************************************************************
    */
    MPI_Dims_create(size, ndims, dims);
    if(my_rank==0)
        printf("Root Rank: %d. Comm Size: %d: Grid Dimension = [%d x
%d] \n",my_rank,size,dims[0],dims[1]);

    /* create cartesian mapping */
    wrap_around[0] = wrap_around[1] = 0; /* periodic shift is
.false. */
    reorder = 1;
    ierr =0;
    ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, wrap_around,
reorder, &comm2D);
```

```
    if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

    /* find my coordinates in the cartesian communicator group */
    MPI_Cart_coords(comm2D, my_rank, ndims, coord);
    /* use my cartesian coordinates to find my rank in cartesian
group*/
    MPI_Cart_rank(comm2D, coord, &my_cart_rank);
    /* get my neighbors; axis is coordinate dimension of shift */
    /* axis=0 ==> shift along the rows: P[my_row-1]: P[me] :
P[my_row+1]*/
    /* axis=1 ==> shift along the columns P[my_col-1]: P[me] :
P[my_col+1] */

    MPI_Cart_shift( comm2D, SHIFT_ROW, DISP, &nbr_i_lo, &nbr_i_hi);
    MPI_Cart_shift( comm2D, SHIFT_COL, DISP, &nbr_j_lo, &nbr_j_hi);

    printf("Global rank: %d. Cart rank: %d. Coord: (%d, %d).
Left: %d. Right: %d. Top: %d. Bottom: %d\n", my_rank, my_cart_rank,
coord[0], coord[1], nbr_j_lo, nbr_j_hi, nbr_i_lo, nbr_i_hi);
    fflush(stdout);

    MPI_Comm_free( &comm2D );
    MPI_Finalize();
    return 0;
}
```

MONASH University

Task 2 – Inter process communication between adjacent processes in a Cartesian grid topology

Using the aforementioned sample solution and with reference to Figure 1, implement the following:

a) Each MPI process in the grid generates a random prime number and exchanges the random prime value with its adjacent processes. You can use any type of MPI send and receive function (asynchronous or synchronous here) or other MPI collective communication functions.

b) Upon exchanging the random numbers, each process compares the received prime numbers with its own prime number. **If there is a match between any one of the received prime numbers with its own prime number,** the process logs this information into a text file. This means that if you are running a MPI topology using a 4 × 5 grid, you will have 20 unique log files. Each log file will either contain entries for matching prime numbers or no content (depending on the randomness of the generated prime numbers).

c) Repeat part (b) above using a loop based on a fixed number of iterations.

Sample solution:

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <stdbool.h>
#include "mpi.h"

#define NBRILO 10
#define NBRIHI 11
#define NBRJLO 12
#define NBRJHI 13
#define SHIFT_ROW 0
#define SHIFT_COL 1
#define DISP 1

#define MAX_ITERATION 50
#define randomUB 50

// ----------------------------------------------------------------------
/// Is Prime Function Definition
/// Input: integer number
/// Returns: 1 if input argument is a prime number, 0 otherwise
//----------------------------------------------------------------------
```

```
int IsPrime(int input);

int main(int argc, char *argv[]) {
    int ndims=2, size, my_rank, reorder, my_cart_rank, ierr, nrows, ncols,
nbr_i_lo, nbr_i_hi;
    int nbr_j_lo, nbr_j_hi;
    MPI_Comm comm2D;
    int dims[ndims],coord[ndims];
    int wrap_around[ndims];
    int i;
    char buf[256] = {0};
    FILE *pFile;

    int iteration_count = 1;
    int num;
    int matching_count = 0;

    MPI_Request send_request[4];
    MPI_Request receive_request[4];
    MPI_Status send_status[4];
    MPI_Status receive_status[4];

    /* start up initial MPI environment */
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    unsigned int seed = time(NULL) * my_rank;

    /* process command line arguments*/
    if (argc == 3) {
        nrows = atoi (argv[1]);
        ncols = atoi (argv[2]);
        dims[0] = nrows; /* number of rows */
        dims[1] = ncols; /* number of columns */
        if( (nrows*ncols) != size) {
            if( my_rank ==0) printf("ERROR: nrows*ncols)=%d * %d = %d !=
%d\n", nrows, ncols, nrows*ncols,size);
            MPI_Finalize();
            return 0;
        }
    } else {
        nrows=ncols=(int)sqrt(size);
        dims[0]=dims[1]=0;
    }

    /*************************************************************/
    /* create cartesian topology for processes */
    /*************************************************************/
    MPI_Dims_create(size, ndims, dims);
    if(my_rank==0)
        printf("PW[%d], CommSz[%d]: PEdims = [%d x %d]
\n",my_rank,size,dims[0],dims[1]);

    /* create cartesian mapping */
    wrap_around[0] = wrap_around[1] = 0; /* periodic shift is .false.
*/
    reorder = 1;
```

```
    ierr =0;

    ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
wrap_around,reorder, &comm2D);
    if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

    /* find my coordinates in the cartesian communicator group */
    MPI_Cart_coords(comm2D, my_rank, ndims, coord);
    /* use my cartesian coordinates to find my rank in cartesian group*/
    MPI_Cart_rank(comm2D, coord, &my_cart_rank);
    /* get my neighbors; axis is coordinate dimension of shift */
    /* axis=0 ==> shift along the rows: P[my_row-1]: P[me] : P[my_row+1] */
    /* axis=1 ==> shift along the columns P[my_col-1]: P[me] : P[my_col+1]
*/

    MPI_Cart_shift( comm2D, SHIFT_ROW, DISP, &nbr_i_lo, &nbr_i_hi );
    MPI_Cart_shift( comm2D, SHIFT_COL, DISP, &nbr_j_lo, &nbr_j_hi );
    int action_list[4] = {nbr_j_lo, nbr_j_hi, nbr_i_lo, nbr_i_hi};
    int recv_data[4] = {-1, -1, -1, -1};

    sprintf(buf, "log_%d.txt", my_rank);
    pFile = fopen(buf, "w");

    fprintf(pFile, "Global rank: %d. Cart rank: %d. Coord: (%d, %d). Left:
%d. Right: %d. Top: %d. Bottom: %d\n\n", my_rank, my_cart_rank, coord[0],
coord[1], nbr_j_lo, nbr_j_hi, nbr_i_lo, nbr_i_hi);


    while(iteration_count < MAX_ITERATION){

          matching_count = 0; //reset the matching count

          while(true){
                num = rand_r(&seed) % randomUB;
                if(IsPrime(num) == 0){
                      break;
                }
          }


          for (i = 0; i < 4; i++){
              MPI_Isend(&num, 1, MPI_INT, action_list[i], 0, comm2D,
&send_request[i]);
              MPI_Irecv(&recv_data[i], 1, MPI_INT, action_list[i], 0,
comm2D, &receive_request[i]);
          }
          MPI_Waitall(4, send_request, send_status);
          MPI_Waitall(4, receive_request, receive_status);

          // printf("Global rank: %d. Cart rank: %d. Coord: (%d, %d). Rand:
%d. Recv Vals - ", my_rank, my_cart_rank, coord[0], coord[1], num);
          for (i = 0; i < 4; i++){
              //printf("recv_data[%d] = %d\t", i, recv_data[i]);
              if(num==recv_data[i]){
                    matching_count++;
              }
          }
```

```
            // record the informatioo if there is any matches
            if(matching_count>=1){
                fprintf(pFile, "[ITERATION %d] Random value: %d; Recv Left:
%d; Recv Right: %d; Recv Top: %d; Recv Bottom: %d; Num of matches:%d\n",
iteration_count, num, recv_data[0], recv_data[1], recv_data[2], recv_data[3],
matching_count);
            }

            iteration_count++;
      }

      fclose(pFile);
      MPI_Comm_free(&comm2D);
      MPI_Finalize();
      return 0;
}

int IsPrime(int input)
{
      int j;
      int limit = (int)sqrt((double)input);

      for(j = 2; j <= limit; j++)
      {
            if (input % j == 0)
            {
                return 0;
            }
      }
      return 1;
}
```

MONASH University

Task 3 – Master/Slaves program in MPI Worked Example

Message passing is well-suited to handling computations where a task is divided up into subtasks, with most of the processes used to compute the subtasks and a few processes (often just one process) managing the tasks.

The manager is called the "master" and the others the "workers" or the "slaves".

In this task, you will begin to build an Input/Output master/slave system. This will allow you to relatively easily arrange for different kinds of input and output from your program, including:

- Ordered output (process 2 after process 1)
- Duplicate removal (a single instance of "Hello world" instead of one from each process)
- Input to all processes from a terminal

This will be accomplished by dividing the processes in MPI_COMM_WORLD into two sets - the master (who will do all of the I/O) and the slaves (who will do all of their I/O by contacting the master). The slaves will also do any other computation that they might desire.

For this task, divide the processors into two communicators, with one processor the master and the others the slave. The master should accept messages from the slaves (of type MPI_CHAR) and print them in rank order (that is, first from slave 0, then from slave 1, etc.). The slaves should each send 2 messages to the master. For simplicity, have the slaves send the messages.

```
Hello from slave 3


Goodbye from slave 3
```

(with appropriate values for each slave). You may assume a maximum length message of 256 characters.

For this first task, keep the code simple. Do not use `intercommunicators`. Also, you'll find that you use the new communicator for the slaves only to get the rank of the slave in its communicator. Note that if the slaves were also computing, they would use that new communicator instead of MPI_COMM_WORLD. You may want to use these MPI routines in your solution:

```
MPI_Comm_split MPI_Send MPI_Recv
```

Sample solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
```

```
int master_io(MPI_Comm master_comm, MPI_Comm comm);
int slave_io(MPI_Comm master_comm, MPI_Comm comm);

int main(int argc, char **argv)
{
    int rank;
    MPI_Comm new_comm; MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_split( MPI_COMM_WORLD,rank == 0, 0, &new_comm);
    if (rank == 0)
       master_io( MPI_COMM_WORLD, new_comm );
    else
       slave_io( MPI_COMM_WORLD, new_comm );
    MPI_Finalize();
    return 0;
 }

 /* This is the master */
 int master_io(MPI_Comm master_comm, MPI_Comm comm)
 {
    int i,j, size; char buf[256];
    MPI_Status status;
    MPI_Comm_size( master_comm, &size );
    for (j=1; j<=2; j++) {
       for (i=1; i<size; i++) {
           MPI_Recv( buf, 256, MPI_CHAR, i, 0, master_comm, &status);
           fputs( buf, stdout );
        }
    }
    return 0;
 }

 /* This is the slave */
 int slave_io(MPI_Comm master_comm, MPI_Comm comm)
 {
    char buf[256];
    int rank;

    MPI_Comm_rank(comm, &rank);
    sprintf(buf, "Hello from slave %d\n", rank);
    MPI_Send( buf, strlen(buf) + 1, MPI_CHAR, 0, 0, master_comm );

    sprintf( buf, "Goodbye from slave %d\n", rank );
    MPI_Send( buf, strlen(buf) + 1, MPI_CHAR, 0, 0, master_comm );
    return 0;
 }
```

MONASH University

Task 4 – A Simple Output Server

Modify the output master to accept three types of messages from the slaves. These types are

- Ordered output (just like the previous exercise)
- Unordered output (as if each slave printed directly)
- Exit notification (see below)

The master continues to receive messages until it has received an exit message from each slave. For simplicity in programming, have each slave send the messages

```
Hello from slave 3
```

```
Goodbye from slave 3
```

with the ordered output mode, and

```
I'm exiting (3)
```

with the unordered output mode.

You may want to use these MPI routines in your solution:

```
MPI_Comm_split MPI_Send MPI_Recv
```

Sample solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
#include<time.h>

#define MSG_EXIT 1
#define MSG_PRINT_ORDERED 2
#define MSG_PRINT_UNORDERED 3

int master_io(MPI_Comm master_comm, MPI_Comm comm);
int slave_io(MPI_Comm master_comm, MPI_Comm comm);

int main(int argc, char **argv)
{
    int rank;
    MPI_Comm new_comm;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_split( MPI_COMM_WORLD,rank == 0, 0, &new_comm);
    if (rank == 0)
       master_io( MPI_COMM_WORLD, new_comm );
    else
       slave_io( MPI_COMM_WORLD, new_comm );
    MPI_Finalize();
    return 0;
}
```

```
/* This is the master */
int master_io(MPI_Comm master_comm, MPI_Comm comm)
{
    int i, size, nslave, firstmsg;
    char buf[256], buf2[256];
    MPI_Status status;
    MPI_Comm_size( master_comm, &size ); nslave = size - 1;
    while (nslave > 0) {
       MPI_Recv( buf, 256, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG,
master_comm, &status );
       switch (status.MPI_TAG) {
            case MSG_EXIT:
                // fputs( buf, stdout ); // not requires to print the exit
notification message
                nslave--;
                break;
            case MSG_PRINT_UNORDERED:
                fputs( buf, stdout );
                break;
            case MSG_PRINT_ORDERED:
                firstmsg = status.MPI_SOURCE;
                for (i=1; i<size; i++) {
                    if (i == firstmsg)
                        fputs( buf, stdout );
                    else {
                        MPI_Recv( buf2, 256, MPI_CHAR, i, MSG_PRINT_ORDERED,
master_comm, &status);
                        fputs( buf2, stdout);
                    }
                }
                break;
       }
    }
    return 0;
}
/* This is the slave */
int slave_io(MPI_Comm master_comm, MPI_Comm comm)
{
    char buf[256];
    int rank;

    MPI_Comm_rank( comm, &rank );
    sprintf( buf, "Hello from slave %d\n", rank );
    MPI_Send( buf, strlen(buf) + 1, MPI_CHAR, 0, MSG_PRINT_ORDERED,
master_comm );

    sprintf(buf, "Goodbye from slave %d\n", rank);
    MPI_Send(buf, strlen(buf) + 1, MPI_CHAR, 0, MSG_PRINT_ORDERED,
master_comm);

    //can use sleep(...) or usleep(...) to delay the process of sending
unordered output message for a random timing to simulate the processes
sends the message in different timing
    sprintf(buf, "I'm exiting (%d)\n", rank);
```

```
    MPI_Send(buf, strlen(buf) + 1, MPI_CHAR, 0, MSG_PRINT_UNORDERED,
master_comm);

    sprintf(buf, "Exit notification from %d\n", rank);
    MPI_Send(buf, strlen(buf) + 1, MPI_CHAR, 0, MSG_EXIT, master_comm);
    return 0;
}
```

TASK 5 – 3D CARTESIAN VIRTUAL TOPOLOGY

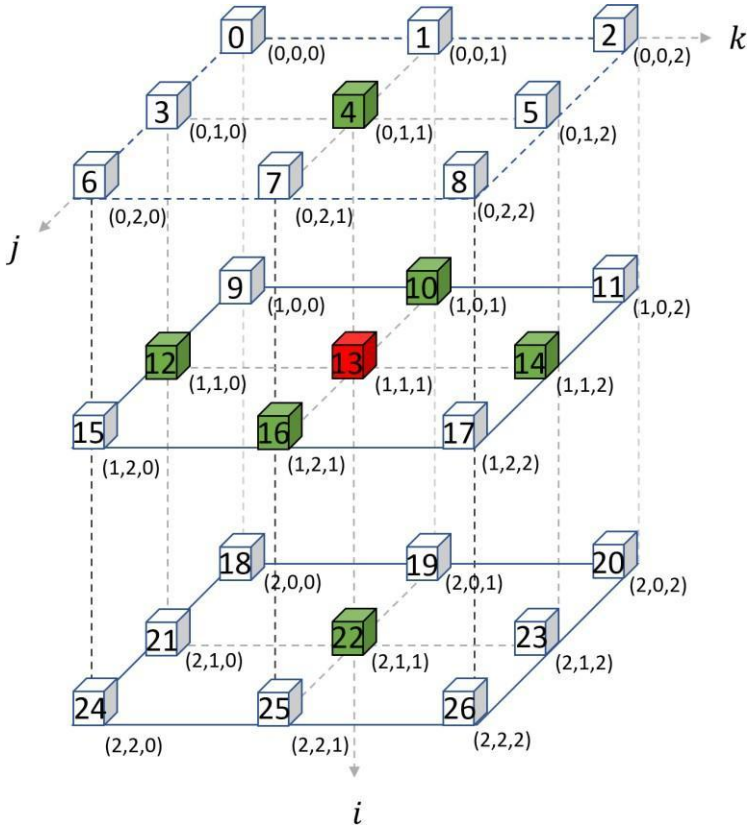

Figure 2: Sample 3D Cartesian (3×3×3). Mapping between a process rank and this Cartesian diagram is based on (i, j, k) indexing.
Image created by Ming Jie Lee.

Using the MPI virtual topology, implement a 3D Cartesian as illustrated in Figure 2.This figure also includes an example of a MPI process with rank 13 (refer to the red-coloured block) and its immediate neighbourhood nodes (refer to the green-coloured blocks). Implement the following:

a) Each MPI process prints out its immediate neighborhood nodes.
- Current rank
- Cartesian rank
- Coordinates
- List of immediate adjacent processes (left, right, top, bottom, front and rear)

b) Repeat parts (a) to (c) from **Task 2**, but in the context of a 3D Cartesian layout.

Note: The user has an option to specify the grid dimension (e.g., 3x2x2 or 3x3x3) as a command line argument. When executing ***mpirun***, use the ***oversubscribe*** option so that you could run more MPI processes in a single computer.

For instance, if the user specifies a grid dimension as command line argument:

mpirun -np 12 -oversubscribe mpiOut 3 2 2

Alternatively, if the user intends to give total freedom to MPI to specify the grid dimensions:

mpirun -np 12 -oversubscribe mpiOut

Optional - We encourage you to compile and execute your program on CAAS. Should you run your program in CAAS, please configure the job script.

Sample solution:

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
#include <unistd.h>
#include <string.h>

#define NUM_RANGE 100
#define SHIFT_ROW 0
#define SHIFT_COL 1
#define SHIFT_DEP 2 // changing to 2 shifts #define DISP 1

#define MAX_ITERATION 50
#define randomUB 50

// ----------------------------------------------------------------------
/// Is Prime Function Definition
/// Input: integer number
/// Returns: 1 if input argument is a prime number, 0 otherwise
// ----------------------------------------------------------------------
int IsPrime(int input);
int CheckValue(int* recvValues, int val) ;

int main(int argc, char **argv)
{
    int ndims=3, size, my_rank, reorder, my_cart_rank, ierr;
    int nrows, ncols, ndepts;
    int nbr_i_lo, nbr_i_hi;
```

```
    int nbr_j_lo, nbr_j_hi;
    int nbr_k_lo, nbr_k_hi;
    MPI_Comm comm3D;
    int dims[ndims],coord[ndims];
    int wrap_around[ndims];
    char buf[256] = {0};
    FILE *pFile;

    unsigned int randomSeed; // random number generator seed int
    iteration_count = 1;
    int randVal;

    MPI_Request send_request[6];
    MPI_Request receive_request[6];
    MPI_Status send_status[6];
    MPI_Status receive_status[6];

    MPI_Init(&argc, &argv); MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    randomSeed = my_rank * time(NULL); // Initialize random seed for each
process

    /* process command line arguments*/
    if (argc == 4) {
       nrows = atoi (argv[1]);
       ncols = atoi (argv[2]);
       ndepts = atoi (argv[3]);
       dims[0] = nrows; /* number of rows */
       dims[1] = ncols; /* number of columns */
       dims[2] = ndepts;
       if( (nrows*ncols*ndepts) != size) {
           if( my_rank ==0) printf("ERROR: nrows*ncols*ndepts)=%d * %d * %d
= %d != %d\n", nrows, ncols, ndepts, nrows*ncols*ndepts,size);
           MPI_Finalize();
           return 0;
       }
    } else {
       nrows=ncols=(int)sqrt(size);
       dims[0]=dims[1]=dims[2]=0;
    }

    MPI_Dims_create(size, ndims, dims);
    if(my_rank==0)
       printf("Root Rank: %d. Comm Size: %d: Grid Dimension = [%d x
%d x %d] \n",my_rank,size,dims[0],dims[1], dims[2]);

    /* create cartesian mapping */
    wrap_around[0] = 0;
    wrap_around[1] = 0;
    wrap_around[2] = 0; /* periodic shift is .false. */
    reorder = 1;
    ierr = 0;
    ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, wrap_around,
reorder, &comm3D);
```

```
    if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

    MPI_Cart_coords(comm3D, my_rank, ndims, coord); // coordinated is
returned into the coord array
    MPI_Cart_rank(comm3D, coord, &my_cart_rank);

    MPI_Cart_shift(comm3D, SHIFT_ROW, DISP, &nbr_i_lo, &nbr_i_hi);
    MPI_Cart_shift(comm3D, SHIFT_COL, DISP, &nbr_j_lo, &nbr_j_hi);
    MPI_Cart_shift(comm3D, SHIFT_DEP, DISP, &nbr_k_lo, &nbr_k_hi);

    sprintf(buf, "log_%d.txt", my_rank);
    pFile = fopen(buf, "w");

    fprintf(pFile, "Global rank: %d. Cart rank: %d. Coord: (%d, %d, %d).
Rear: %d. Front: %d. Top: %d. Bottom: %d. Left: %d. Right: %d. \n\n",
my_rank, my_cart_rank, coord[0], coord[1], coord[2], nbr_j_lo, nbr_j_hi,
nbr_i_lo, nbr_i_hi, nbr_k_lo, nbr_k_hi);

    while(iteration_count < MAX_ITERATION){
          while(true){
                randVal = rand_r(&randomSeed) % randomUB;
                if(IsPrime(randVal) == 0){
                    break;
                }
          }

          int recvValues[6] = {-1, -1, -1, -1, -1, -1};
          int neighbors[6] = {nbr_j_lo, nbr_j_hi, nbr_i_lo, nbr_i_hi,
nbr_k_lo, nbr_k_hi};

          for (int i = 0; i < 6; i++) {
                MPI_Isend(&randVal, 1, MPI_INT, neighbors[i], 0, comm3D,
&send_request[i]);
                MPI_Irecv(&recvValues[i], 1, MPI_INT, neighbors[i], 0,
comm3D, &receive_request[i]);
          }

          MPI_Waitall(6, send_request, send_status);
          MPI_Waitall(6, receive_request, receive_status);


          if(CheckValue(recvValues, randVal) >= 1){
             fprintf(pFile, "[ITERATION %d] Random value: %d; Recv Rear: %d;
Recv Front: %d; Recv Top: %d; Recv Bottom: %d; Recv Left: %d; Recv Right: %d;
Number of matches %d\n", iteration_count, randVal, recvValues[4],
recvValues[5], recvValues[2], recvValues[3], recvValues[0], recvValues[1],
CheckValue(recvValues, randVal));
          }


          iteration_count++;
    }

    fclose(pFile);
```

```
    MPI_Comm_free( &comm3D );
    MPI_Finalize();
    return 0;
}

int CheckValue(int* recvValues, int val) {
    int retVal = 0;
    for (int i = 0; i < 6; i++) {
       if(recvValues[i] == val){
          retVal++;
       }
    }
    return retVal;
}

int IsPrime(int input)
{
    int j;
    int limit = (int)sqrt((double)input);

    for(j = 2; j <= limit; j++)
    {
        if (input % j == 0)
        {
            return 0;
        }
    }
    return 1;
}
```