

FIT3143 Lab Week 9

Lecturers: ABM Russel (MU Australia) and Vishnu Monn (MU Malaysia)

MPI VIRTUAL TOPOLOGY AND MASTER & SLAVE

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 VM [Refer to Lab Week 1]
- Setup eFolio (including Git) and share with tutor and partner [Refer to Lab Week 1]

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.

WHAT TO SUBMIT:

1. Algorithm or code description, analysis of results, screenshot of the running programs and git repository URL in the eFolio.
2. Code in the Git.

EVALUATION CRITERIA

- This Lab-work is part of grading
- Code compile without errors (2), well commented (2), lab-work questions fully answered (4), analysis/report is well formatted (2) = 10 marks

LAB ACTIVITIES (10 MARKS)

Task1 - 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:

- Current rank
- Cartesian rank
- Coordinates
- List of immediate adjacent processes (left, right, top and bottom)

The use has an option to specify the grid size as a command line argument.

Sample solution:

```

/* Gets the neighbors in a cartesian communicator
 * Originally 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;
}

```

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:

- 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.
- Upon exchanging the random numbers, each process compares the received prime numbers with its own prime number. If all the prime numbers are the same, 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).
- Repeat part (b) above using a loop based on a fixed number of iterations.

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;
}
```

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`

OPTIONAL BONUS ACTIVITIES (2 MARKS)

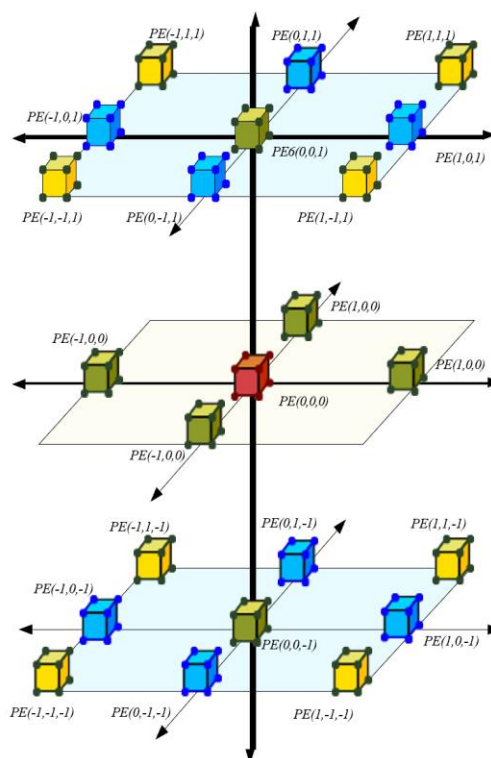


Figure 2: 3D Cartesian Mapping.

Image source: <https://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-Cart-Comms-and-Topos.pdf>

Using MPI virtual topology, implement a 3D Cartesian as illustrated in Figure 2.

- a) Each MPI process prints out its immediate neighborhood nodes.
- b) Repeat parts (a) to (c) from Task 2, but in the context of a 3D Cartesian layout.