WARSAW UNIVERSITY OF TECHNOLOGY

FACULTY OF MATHEMATICS AND INFORMATION SCIENCE

FIRST SEMESTER 2021/2022

HIGH PERFORMANCE COMPUTING (HPC)

PROJECT 1

BY

ANDRA UMORU (324334)

SUBMITTED TO:

OKULICKA-DLUZEWSKA FELICJA

TABLE OF CONTENT

PROJECT ONE

1.1 Project description

Write program using MPI where processes are on the ring. Each process sends 2 messages: on right message „R” and on left message „L”.

1.2 Overview of MPI

Message Passing Interface (MPI) is a specification for an API that allows many computers to communicate with one another. It is used in computer clusters and supercomputers. MPI's design of the message passing model of parallel programming. The first concept is the notion of a communicator. A communicator defines a group of processes that have the ability to communicate with one another. In this group of processes, each is assigned a unique *rank*, and they explicitly communicate with one another by their ranks.

1.3 MPI Ring Program

In MPI ring program, processes use MPI_Send and MPI_Recv to continually send and receive messages off of each other until they decide to stop. In MPI ring program, a value is passed around by all processes in a ring-like pattern. This means that each process sends and receives message one of and to another.

1.4 Prototype for Sending and Receiving functions in C language

MPI presents us with the send and receive function.

```
MPI_Send(
   void* data,
   int count,
   MPI_Datatype datatype,
   int destination,
   int tag,
   MPI_Comm communicator)

MPI_Recv(
   void* data,
   int count,
   MPI_Datatype datatype,
   int source,
   int tag,
   MPI_Comm communicator,
   MPI_Status* status)
```

1.5 The Source Code

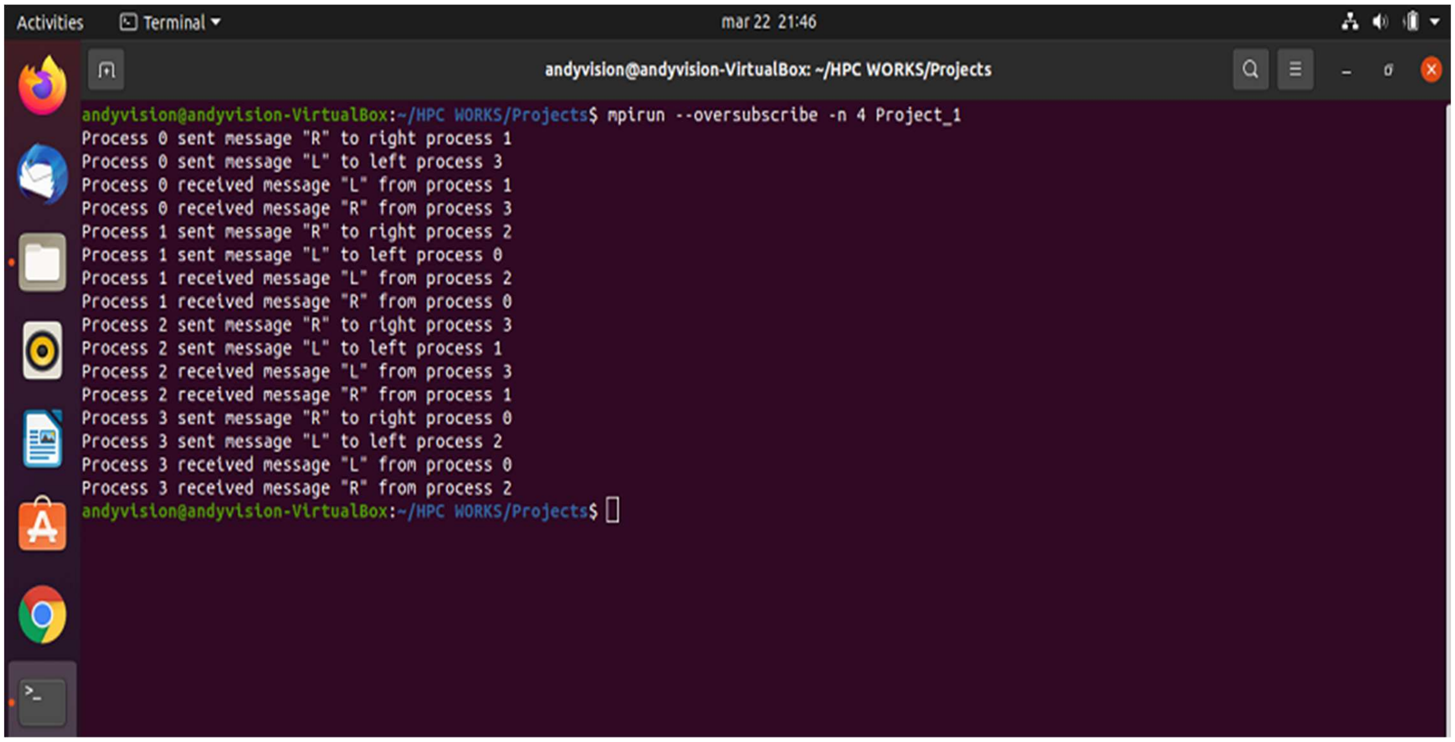
The source code was written in C programming language. Below is the source code for project one:

```
#include <stdio.h>     //including the C standard library
#include <mpi.h>      //Including the MPI library

// C Language main function
int main (int argc, char** argv) {
  MPI_Request request;

  //Declaring the size and the rank variables
  int the_size,the_rank;

  //Initializing the MPI API
  MPI_Init(&argc, &argv);

  // Getting number of process
  MPI_Comm_size(MPI_COMM_WORLD, &the_size);

   // Getting rank of process
  MPI_Comm_rank(MPI_COMM_WORLD, &the_rank);

  //Declaring and setting the message variables to be sent over the processes
  char right_message = 'R';
  char left_message = 'L';

  //Initializing and setting the movement between the processes
  int next_proc= (the_rank+1) % the_size;
  int prev_proc = 0;
            if (the_rank == 0)
                        prev_proc = the_size - 1;
            else
                        prev_proc = the_rank -1;

  //Sending and printing the message to the process on the right
  MPI_Isend(&right_message, 1, MPI_UNSIGNED_CHAR, next_proc, 1, MPI_COMM_WO
RLD, &request);
  printf("Process %d sent message \"%c\" to right process %d\n", the_rank, right_message, nex
t_proc);

  //Sending and printing the message to the process on the left (in a way, forming the ring)
  MPI_Isend(&left_message, 1, MPI_UNSIGNED_CHAR, prev_proc, 2, MPI_COMM_WOR
LD, &request);
  printf("Process %d sent message \"%c\" to left process %d\n", the_rank, left_message, prev_
proc);

  //Receiving and printing the message received from the intializing process
  MPI_Irecv(&left_message, 1, MPI_UNSIGNED_CHAR, next_proc, 1, MPI_COMM_WOR
LD, &request);
```

```
  printf("Process %d received message \"%c\" from process %d\n", the_rank, left_message, ne
xt_proc);

  //Receiving and printing the message received from the first process (in a way, forming the ri
ng)
  MPI_Irecv(&right_message, 1, MPI_UNSIGNED_CHAR, prev_proc, 2, MPI_COMM_WO
RLD, &request);
  printf("Process %d received message \"%c\" from process %d\n", the_rank, right_message, p
rev_proc);

  //Exiting MPI routine
  MPI_Finalize();

  return 0;
}
```

1.6 Result (Output)

After compiling and running the MPI program, the figure below is the output:



```
andyvision@andyvision-VirtualBox:~/HPC WORKS/Projects$ mpirun --oversubscribe -n 4 Project_1
Process 0 sent message "R" to right process 1
Process 0 sent message "L" to left process 3
Process 0 received message "L" from process 1
Process 0 received message "R" from process 3
Process 1 sent message "R" to right process 2
Process 1 sent message "L" to left process 0
Process 1 received message "L" from process 2
Process 1 received message "R" from process 0
Process 2 sent message "R" to right process 3
Process 2 sent message "L" to left process 1
Process 2 received message "L" from process 3
Process 2 received message "R" from process 1
Process 3 sent message "R" to right process 0
Process 3 sent message "L" to left process 2
Process 3 received message "L" from process 0
Process 3 received message "R" from process 2
andyvision@andyvision-VirtualBox:~/HPC WORKS/Projects$ 
```

Figure 1: Project 1 Output

1.7 Reference

- HPC Lecture Notes