HPC - Exercise 4

Eduardo Gonnelli

07 July 2019

1 Introduction

The aim of this report is to write a code that a set of processes are arranged in a ring-like fashion. Each process will send the message to its left neighbor, i.e., the information must move in an anticlockwise direction. The code is divided into two parts. The first part each process will send an integer number to its neighbor and each process will print the received message and the process that sent it. The second part of the code each process stores its rank in MPI_COMM_WORLD into a array of size 2048. Then, each process sends the array to its left neighbor. In addition, each process calculates the sum of all numbers, keep passing it around the ring until the value is back where it started, i.e, each process calculates the sum of all ranks. To avoid deadlocks the non-blocking MPI_Isend was employed.

In the Figures 1 and 2 show the processors arranged in a ring and the way that information rotates around the ring with the rank's value summation, respectively.

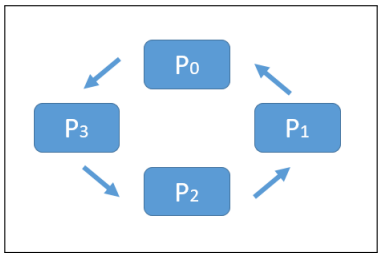

Figure 1: Four processors arranged in a ring. The message sent in an anticlockwise direction

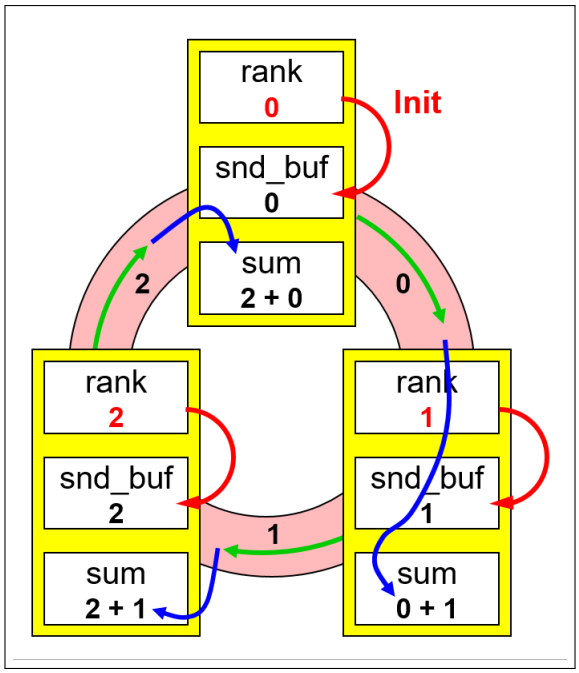

Figure 2: Rotating information around a ring. This image is just an illustration how the rank number was sent and summed to the next rank number. It represents only the first step and not the completed lap. The anticlockwise direction was not considered in the image.

2 Results

The Figure 3 shows the output of the *ring.o* for different numbers of processes. It can be seen that the integer number -999 was sent to the left side of the ring.

```
[egonnell@cn07-31 ring]$ mpirun -n 2 ./ring.o
-999: 0 ----> 1
-999: 1 ----> 0
[egonnell@cn07-31 ring]$ mpirun -n 4 ./ring.o
-999: 0 ----> 3
-999: 1 ----> 0
-999: 2 ----> 1
-999: 3 ----> 2
[egonnell@cn07-31 ring]$ mpirun -n 6 ./ring.o
-999: 3 ----> 2
-999: 4 ----> 3
-999: 5 ----> 4
-999: 0 ----> 5
-999: 1 ----> 0
-999: 2 ----> 1
[egonnell@cn07-31 ring]$ mpirun -n 8 ./ring.o
-999: 0 ----> 7
-999: 1 ----> 0
-999: 2 ----> 1
-999: 3 ----> 2
-999: 4 ----> 3
-999: 5 ----> 4
-999: 6 ----> 5
-999: 7 ----> 6
```

Figure 3: The number of processes (-n) 2, 4, 6, and 8 were took into account.

The Figure 4 shows the output of the *ring_array.o* for different numbers of processes. One could be noticed that all the ranks printed the same value of the summation of the rank's value. For example, for -n 4, the sum of all rank's values *0+1+2+3* will result in the number 6, as expected.

```
[egonnell@cn06-08 ring]$ mpirun -n 2 ./ring_array.o
Rank [0] ----> The sum is: [1]
Rank [1] ----> The sum is: [1]
[egonnell@cn06-08 ring]$ mpirun -n 4 ./ring_array.o
Rank [0] ----> The sum is: [6]
Rank [1] ----> The sum is: [6]
Rank [2] ----> The sum is: [6]
Rank [3] ----> The sum is: [6]
[egonnell@cn06-08 ring]$ mpirun -n 6 ./ring_array.o
Rank [3] ----> The sum is: [15]
Rank [4] ----> The sum is: [15]
Rank [5] ----> The sum is: [15]
Rank [0] ----> The sum is: [15]
Rank [1] ----> The sum is: [15]
Rank [2] ----> The sum is: [15]
[egonnell@cn06-08 ring]$ mpirun -n 8 ./ring_array.o
Rank [0] ----> The sum is: [28]
Rank [1] ----> The sum is: [28]
Rank [2] ----> The sum is: [28]
Rank [3] ----> The sum is: [28]
Rank [4] ----> The sum is: [28]
Rank [5] ----> The sum is: [28]
Rank [6] ----> The sum is: [28]
Rank [7] ----> The sum is: [28]
```

Figure 4: The number of processes (-n) 2, 4, 6, and 8 were took into account.

References

[1] Irish Centre for High-End Computing (ICHEC): *Introduction to the Message Passing Interface (MPI)* `https://www.ichec.ie/`

Appendices

A Codes

Listing 1: MPI C++ Code - Ring (simple message: -999)

```
/* This Program send an integer number in a ring-like (circle) fashion */

#include<iostream>
#include<mpi.h>

    int main(int argc, char** argv){

    int size, rank;
    int message = -999;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Request request;

    if (size < 2)
    {
        std::cout << "Number␣of␣processes␣must␣be␣at␣least␣2␣" << "\n";
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    /* each rank will have an array filled with
       the own rank value */

    for(int i=0;i<size;i++)
    {
          MPI_Isend(&message, 1, MPI_INT,(rank - 1 + size) % size,
           101, MPI_COMM_WORLD, &request);

          MPI_Recv(&message, 1, MPI_INT, (rank + 1 + size) % size,
           101, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

          MPI_Barrier(MPI_COMM_WORLD);

    std::cout << message << ":␣" << rank << "␣---->␣" <<
     (rank -  1 + size) % size  << "\n";

          MPI_Finalize();

        return 0;
      }
```

Listing 2: MPI C++ Code - Array (Rank's value summation)

```
/* The Program sends an array in a ring-like (circle) fashion */

#include<iostream>
#include<mpi.h>

    int main(int argc, char** argv)
    {

    const int sizeArray = 2048;
    int rank, size;

          MPI_Init(&argc, &argv);
          MPI_Comm_rank(MPI_COMM_WORLD, &rank);
          MPI_Comm_size(MPI_COMM_WORLD, &size);
          MPI_Request request;

    if (size < 2) {
        std::cout << "Number␣of␣processes␣must␣be␣at␣least␣2␣" << "\n" ;
            MPI_Abort(MPI_COMM_WORLD, 1);
                }


    // size of array
    // dynamic allocation employing the 'new'

    int* array = new int [sizeArray];
    int* from_Right = new int [sizeArray];
    int* sum = new int [sizeArray];

    /* each rank will have an array filled with
      the own rank value */

    for(int i = 0; i < sizeArray; i++)
    {
```

```
    array[i] = rank;
    sum[i] = 0;
    from_Right[i] = 0;
  }

        MPI_Barrier(MPI_COMM_WORLD);

  for (int j = 0; j < size; j++)
  {

  // MPI_ISEND(buf, count, type, dest, tag, comm, request, ierr)

  MPI_Isend(array, sizeArray, MPI_INT, (rank - 1 + size) % size, 101,
  MPI_COMM_WORLD, &request);

  //MPI_RECV(buf, count, type, dest, tag, comm, status, ierr)

  MPI_Recv(from_Right, sizeArray, MPI_INT, (rank + 1 + size) % size, 101,
  MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    for (int j = 0; j < sizeArray; j++)
    {
      sum[j] += from_Right[j];
    }

        MPI_Wait(&request, MPI_STATUS_IGNORE);

    array = from_Right ;
    }

        MPI_Barrier(MPI_COMM_WORLD);

  //print the first element of the array to confirm the summation

  std::cout << "Rank␣[" <<rank << "]" << "␣---->␣" << "␣The␣sum␣is:␣[" << sum[0]<< "]"
  << "\n";

        MPI_Finalize();

  delete[] array;
  delete[] from_Right;
  delete[] sum;

        return 0;
}
```