Module:	Introduction to Parallel Programming Techniques
Module ID:	EE4107
Student Name:	Umid Narziev
Student ID:	200234832
Assignment Number:	1
Academic Year:	2020 - 2021

Contents

System Specification

Below, shown the system which was used to run and get the result of the simulation:

CPU:	Intel i5-7200U
Architecture:	Kaby Lake
Segment:	Mobile Processors
The number of cores:	2
Number of threads	4
Clock Frequency	2.50-3.10GHz (Turbo Boost)
Cache levels:	3
Cache level 1 size:	128KBytes
Cache level 2 size:	512Kbytes
Cache level 3 size:	3MBytes
RAM	12 GB
SSD:	250 GB
Operating System:	Ubuntu 20.04.2 LTS
Compiler:	Gcc and its libraries
IDE:	Clion (2020.03)

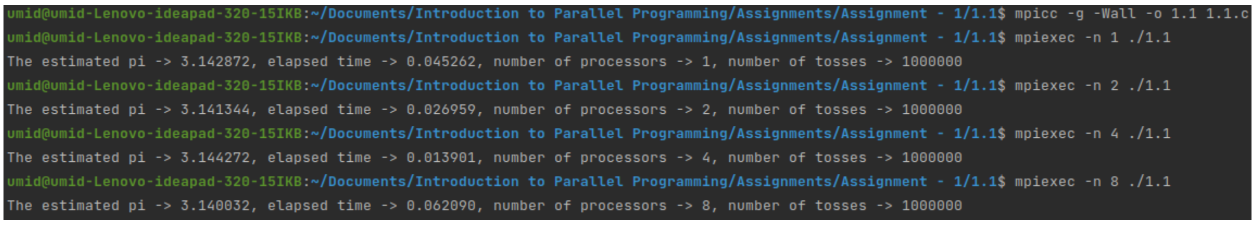
Task 1

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi/mpi.h"

#define MR_MULTIPLIER 279470273
#define MR_INCREMENT 0
#define MR_MODULUS 4294967291U
#define MR_DIVISOR ((double) 4294967291U)

double uniform(double a, double b);

int main() {

   int my_rank, comm_sz;
   double pi_estimate = 0;
   long long int number_of_tosses = 100000000;
   double local_x = 0, local_y = 0;
   long long int local_number_in_circle = 0;
   double local_distance_squared = 0;
   long long int global_number_in_circle = 0;
   double local_start = 0, local_end = 0, local_elapsed = 0, elapsed;
   MPI_Init(NULL, NULL);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

   MPI_Barrier(MPI_COMM_WORLD);
   local_start = MPI_Wtime();
   for (long long int toss = 0; toss < number_of_tosses / comm_sz; toss++) {
      local_x = uniform(-1, 1); // use uniform function
      local_y = uniform(-1, 1);
      local_distance_squared = local_x * local_x + local_y * local_y;
      if (local_distance_squared <= 1) {
         local_number_in_circle++;
      }

   }

   MPI_Reduce(&my_rank, &global_number_in_circle, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
   MPI_Reduce(&local_number_in_circle, &global_number_in_circle, 1, MPI_LONG_LONG_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
   local_end = MPI_Wtime();
   local_elapsed = local_end - local_start;
   MPI_Reduce(&local_elapsed, &elapsed, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
   if (my_rank == 0) {
      pi_estimate = 4 * (double) global_number_in_circle / ((double) number_of_tosses);
      printf("The estimated pi -> %f, elapsed time -> %f, number of processors -> %d, number of tosses ->
%lld\n",
            pi_estimate, elapsed, comm_sz, number_of_tosses);
   }

   MPI_Finalize();
   return 0;
}

double uniform(double a, double b) {
   return rand() / (RAND_MAX + 1.0) * (b - a) + a;
}
```

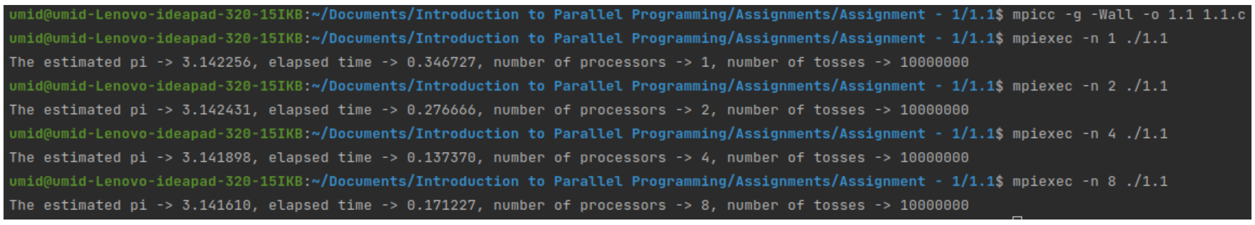
Result:

a) Number of tosses: 1000000



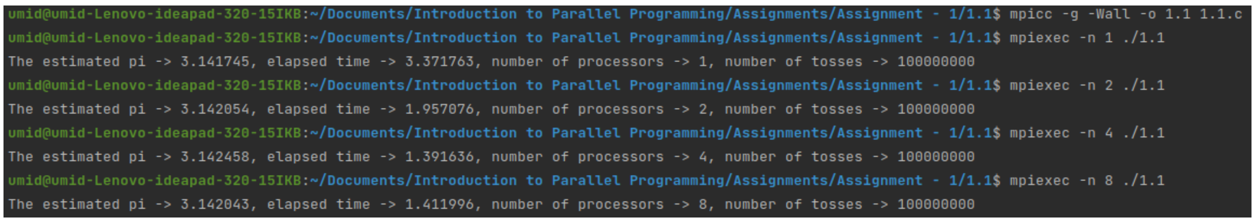
```
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpicc -g -Wall -o 1.1 1.1.c
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 1 ./1.1
The estimated pi -> 3.142872, elapsed time -> 0.045262, number of processors -> 1, number of tosses -> 1000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 2 ./1.1
The estimated pi -> 3.141344, elapsed time -> 0.026959, number of processors -> 2, number of tosses -> 1000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 4 ./1.1
The estimated pi -> 3.144272, elapsed time -> 0.013901, number of processors -> 4, number of tosses -> 1000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 8 ./1.1
The estimated pi -> 3.140032, elapsed time -> 0.062090, number of processors -> 8, number of tosses -> 1000000
```

b) Number of tosses: 10000000



```
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpicc -g -Wall -o 1.1 1.1.c
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 1 ./1.1
The estimated pi -> 3.142256, elapsed time -> 0.346727, number of processors -> 1, number of tosses -> 10000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 2 ./1.1
The estimated pi -> 3.142431, elapsed time -> 0.276666, number of processors -> 2, number of tosses -> 10000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 4 ./1.1
The estimated pi -> 3.141898, elapsed time -> 0.137370, number of processors -> 4, number of tosses -> 10000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 8 ./1.1
The estimated pi -> 3.141610, elapsed time -> 0.171227, number of processors -> 8, number of tosses -> 10000000
```

c) Number of tosses: 10000000



```
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpicc -g -Wall -o 1.1 1.1.c
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 1 ./1.1
The estimated pi -> 3.141745, elapsed time -> 3.371763, number of processors -> 1, number of tosses -> 100000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 2 ./1.1
The estimated pi -> 3.142054, elapsed time -> 1.957076, number of processors -> 2, number of tosses -> 100000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 4 ./1.1
The estimated pi -> 3.142458, elapsed time -> 1.391636, number of processors -> 4, number of tosses -> 100000000
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.1$ mpiexec -n 8 ./1.1
The estimated pi -> 3.142043, elapsed time -> 1.411996, number of processors -> 8, number of tosses -> 100000000
```

SPEEDUP Table

Result	Elapsed time	Number of toss-es	Correctness	Speed-UP	Efficiency	Number of processes
3,142872	4,53E-02	1000000	99,9593%	1,0	100,0%	1
3,141344	2,70E-02	1000000	99,9921%	1,7	83,9%	2
3,144272	1,39E-01	1000000	99,9147%	0,3	8,1%	4
3,140032	6,21E-02	1000000	99,9503%	0,7	9,1%	8
3,142256	3,47E-01	10000000	99,9789%	1,0	100,0%	1
3,142431	2,77E-01	10000000	99,9733%	1,3	62,7%	2
3,141898	1,37E-01	10000000	99,9903%	2,5	63,1%	4
3,141610	1,71E-01	10000000	99,9994%	2,0	25,3%	8
3,141745	3,37E+00	100000000	99,9951%	1,0	100,0%	1
3,142054	1,96E+00	100000000	99,9853%	1,7	86,1%	2
3,142458	1,39E+00	100000000	99,9724%	2,4	60,6%	4
3,142043	1,41E+00	100000000	99,9856%	2,4	29,8%	8

Conclusion:

- From the simulation result, it is seen that for 10^6 number of tosses, speed up decreases with an increase in processes.
- But for 10^7 and 10^8 number of tosses, speed up increases linearly, as I increase processes number.
- It should be noted that I have 2 core – 4 threads and speed up also slows down in progressing up after 4 processes. In conclusion, processes are scheduled in a queue and beyond that 4 processes, it is not efficient to increase the number of processors.

TASK 2

Code:

a) **Tree-structure communication:**

b)
```
/*
 * Tree-Structured 2^n case
 *
 * */
#include "mpi/mpi.h"
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]){
   int count = 0;
   int my_rank, comm_sz;
   int sum = 0;
   int grouping = 2;
//    int n = 8; // Number of processes
   int *a = NULL;
   int dest_var = 0;
   int step = 1;
   MPI_Init(NULL, NULL);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   if(comm_sz%2 != 0){
      printf("Sorry I did not implement this program for any number of processes\n");
      MPI_Finalize();
      return -1;
   }
   if(my_rank == 0){
      a = malloc(comm_sz * sizeof (int)); // n -> comm_sz
      printf("Please enter %d numbers of elements: \n", comm_sz);
      for (int i = 0; i < comm_sz; i++){
         scanf("%d", &a[i]);
      }

      MPI_Scatter(a, 1, MPI_INT, &sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
      free(a);
   } else {
         MPI_Scatter(a, 1, MPI_INT, &sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
   }



   while(grouping <= comm_sz){
     // MPI_Barrier(MPI_COMM_WORLD);
      if(my_rank % grouping == step){
         count++;
      //   printf("my_rank - step => %d - %d = %d: \n", my_rank, step, my_rank - step);
         MPI_Send(&sum, 1, MPI_INT, my_rank - step, 0, MPI_COMM_WORLD);
      }
      if(my_rank % grouping == 0){
         count++;
         MPI_Recv(&dest_var, 1, MPI_INT, my_rank + step, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      //   printf("my_rank + step => %d + %d = %d: \n", my_rank, step, my_rank + step);
         sum += dest_var;
      }

      step <<= 1;
      grouping <<= 1;
```

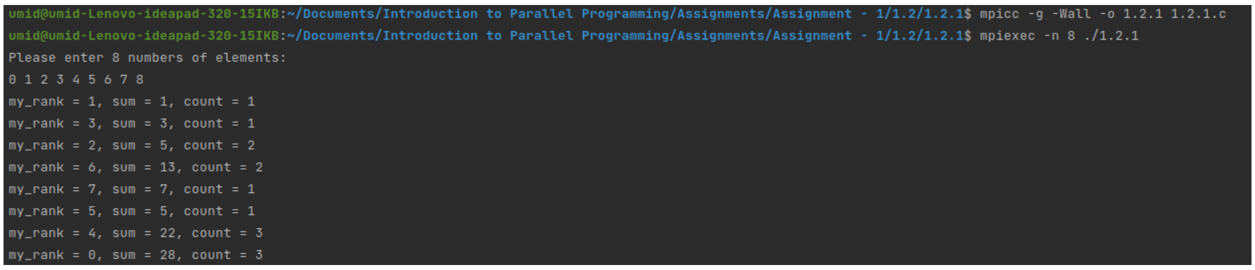
```
  }

  printf("my_rank = %d, sum = %d, count = %d\n", my_rank, sum, count);
  MPI_Barrier(MPI_COMM_WORLD);

  MPI_Finalize();
  return 0;
}
```

b) **Butterfly structure communication:**

c)
```
/*
 * Butterfly structure communication
 */
#include <stdio.h>
#include <mpi/mpi.h>
#include <stdlib.h>

/* Butterfly communication */
int main() {
  int my_rank, comm_sz;
  MPI_Init(NULL, NULL); // Look at this func parameters
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  int group = 2, step = 1, sum = 0;
  int *a = NULL;
  int dest_var = 0;
  if(comm_sz%2 != 0){
    printf("Sorry I did not implement this program for any number of processes\n");
    MPI_Finalize();
    return 1;
  }
  if(my_rank == 0){
    a = malloc(comm_sz * sizeof (int));
    printf("Please enter n = %d of numbers: \n", comm_sz);
    for (int i = 0; i < comm_sz; ++i) {
      scanf("%d", &a[i]);
    }
    MPI_Scatter(a, 1, MPI_INT, &sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
    free(a);
  }else{
    MPI_Scatter(a, 1, MPI_INT, &sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
  }
  printf("Before my_rank -> %d, value -> %d\n", my_rank, sum);
  while (group <= comm_sz){
    if(my_rank % group >= step){
      MPI_Sendrecv(&sum, 1, MPI_INT, my_rank - step, 0, &dest_var, 1, MPI_INT, my_rank - step, 0,
MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
      sum += dest_var;
    }else{
      MPI_Sendrecv(&sum, 1, MPI_INT, my_rank + step, 0, &dest_var, 1, MPI_INT, my_rank + step, 0,
MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
      sum += dest_var;
    }

    group <<= 1;
    step <<= 1;
 }
  printf("After my_rank -> %d, value -> %d\n", my_rank, sum);
  MPI_Finalize();
  return 0;
}
```

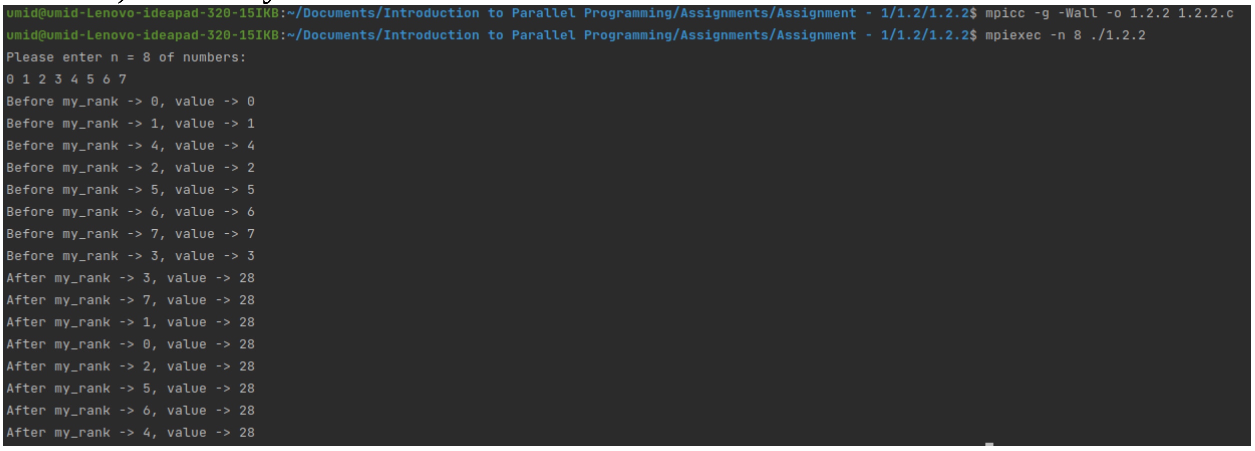
Result:

a) Tree – structure communication



```
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.2/1.2.1$ mpicc -g -Wall -o 1.2.1 1.2.1.c
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.2/1.2.1$ mpiexec -n 8 ./1.2.1
Please enter 8 numbers of elements:
0 1 2 3 4 5 6 7 8
my_rank = 1, sum = 1, count = 1
my_rank = 3, sum = 3, count = 1
my_rank = 2, sum = 5, count = 2
my_rank = 6, sum = 13, count = 2
my_rank = 7, sum = 7, count = 1
my_rank = 5, sum = 5, count = 1
my_rank = 4, sum = 22, count = 3
my_rank = 0, sum = 28, count = 3
```

b) Butterfly structure communication



```
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.2/1.2.2$ mpicc -g -Wall -o 1.2.2 1.2.2.c
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.2/1.2.2$ mpiexec -n 8 ./1.2.2
Please enter n = 8 of numbers:
0 1 2 3 4 5 6 7
Before my_rank -> 0, value -> 0
Before my_rank -> 1, value -> 1
Before my_rank -> 4, value -> 4
Before my_rank -> 2, value -> 2
Before my_rank -> 5, value -> 5
Before my_rank -> 6, value -> 6
Before my_rank -> 7, value -> 7
Before my_rank -> 3, value -> 3
After my_rank -> 3, value -> 28
After my_rank -> 7, value -> 28
After my_rank -> 1, value -> 28
After my_rank -> 0, value -> 28
After my_rank -> 2, value -> 28
After my_rank -> 5, value -> 28
After my_rank -> 6, value -> 28
After my_rank -> 4, value -> 28
```

Conclusion:

- Tree – structure communication is implemented for n^2 processes, and the result collected in process = 0, in the screenshot it is seen that from 0 to 7, the value collected in process 0 is 28. That's 0+1+2+3+4+5+6+7 => 28;
- Butterfly structure communication is also implemented for n^2 processes. Value is distributed over the process from one, and with a given algorithm, butterfly communication is done. From the screenshot, it is seen that after computation processes have the summation of all values that's 28 for 0,1,2,3,4,5,6,7.

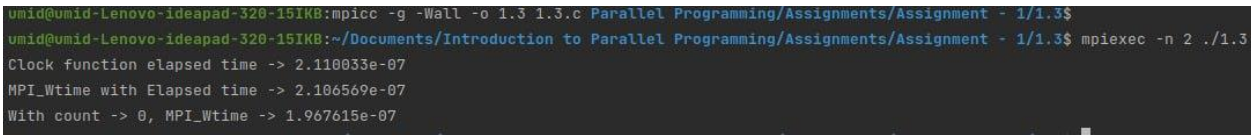
TASK 3

Code:

```
#include <stdio.h>
#include <mpi/mpi.h>
#include <time.h>

#define MAX_CYCLE 16386


int main() {

    int my_rank, comm_sz;
    int AA[MAX_CYCLE];
    double start, elapsed, elapsed_t;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    if(comm_sz != 2){
        if (my_rank == 0){
            printf("Sorry only 2 process are needed to run this program\n");
        }
        MPI_Finalize();
        return -1;
    }
    for (int i = 0; i < MAX_CYCLE; ++i) {
        AA[i] = 10;
    }



    MPI_Barrier(MPI_COMM_WORLD);
    if (my_rank == 0) {
        start = clock() / (double) CLOCKS_PER_SEC;
        for (int i = 0; i < MAX_CYCLE; ++i) {
            MPI_Send(&AA[i], 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&AA[i], 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
        }

        elapsed_t = clock() / (double) CLOCKS_PER_SEC - start;
    } else if (my_rank == 1) {
        for (int i = 0; i < MAX_CYCLE; ++i) {
            MPI_Recv(&AA[i], 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
            MPI_Send(&AA[i], 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        }
    }

    MPI_Barrier(MPI_COMM_WORLD);
    if (my_rank == 0) {
        start = MPI_Wtime();
        for (int i = 0; i < MAX_CYCLE; ++i) {
            MPI_Send(&AA[i], 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&AA[i], 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
        }
        elapsed = MPI_Wtime() - start;
    } else if (my_rank == 1) {
        for (int i = 0; i < MAX_CYCLE; ++i) {
            MPI_Recv(&AA[i], 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
            MPI_Send(&AA[i], 1, MPI_INT, 0, 0, MPI_COMM_WORLD);

        }
    }
```

```c
  if (my_rank == 0) {
    printf("Clock function elapsed time -> %e\n", elapsed_t/(2*MAX_CYCLE));
    printf("MPI_Wtime with Elapsed time -> %e\n", elapsed/(2*MAX_CYCLE));
  }

  MPI_Barrier(MPI_COMM_WORLD);
  if (my_rank == 0) {
    start = MPI_Wtime();
    for (int i = 0; i < MAX_CYCLE; ++i) {
      MPI_Send(&AA[i], 0, MPI_INT, 1, 0, MPI_COMM_WORLD);
      MPI_Recv(&AA[i], 0, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
    }
    elapsed = MPI_Wtime() - start;
  } else if (my_rank == 1) {
    for (int i = 0; i < MAX_CYCLE; ++i) {
      MPI_Recv(&AA[i], 0, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
      MPI_Send(&AA[i], 0, MPI_INT, 0, 0, MPI_COMM_WORLD);

    }
  }
  if (my_rank == 0) {
    printf("With count -> 0, MPI_Wtime -> %e\n", elapsed/(2*MAX_CYCLE));
  }

  MPI_Finalize();
  return 0;
}
```

Result:



```
umid@umid-Lenovo-ideapad-320-15IKB:mpicc -g -Wall -o 1.3 1.3.c Parallel Programming/Assignments/Assignment - 1/1.3$
umid@umid-Lenovo-ideapad-320-15IKB:~/Documents/Introduction to Parallel Programming/Assignments/Assignment - 1/1.3$ mpiexec -n 2 ./1.3
Clock function elapsed time -> 2.110033e-07
MPI_Wtime with Elapsed time -> 2.106569e-07
With count -> 0, MPI_Wtime -> 1.967615e-07
```

Conclusion:

a) The clock gave a non-zero value in ~2.11e-07 seconds, however, it is linked with clock resolution. That’s, the resolution is minimum non-zero run-time. MPI_Wtime and C clock function gave almost the same time with a small difference. Actually, the clock() function should be lesser than MPI_Wtime, as it counts CPU elapsed time The 16386 iterations were done, the result is obtained with round-trip in mind.

b) When the count is zero, I got a non-zero value, and it is associated with overhead in communication, i.e. time to start and end the communication.