**Parallel and Distributed Computing**

**Assignment #1**

Bahaa Hani Ahmed Shamtoot 202009254

Q1: Write an MPI program that can measure the time for sending and receiving messages between processes. Use the following Pseudocode:

|  |  |
| --- | --- |
| **Program 1** | **Program 2** |
| **P0:**  T1 = Time();  Send(x,P1);  T2 = Time();  TP0 = T2- T1;  Recv(TP1, P1);  Print(TP0);  Print(TP1);  **P1:**  T1 = Time();  recv(x,P0);  T2 = Time();  TP1 = T2- T1;  send(TP1, P0); | **P0:**  T1 = Time();  for (i = 0; i < 1000; i++)  send(x[i],P1);  T2 = Time();  TP0 = T2- T1;  Recv(TP1, P1);  Print(TP0);  Print(TP1);  **P1:**  T1 = Time();  for (i = 0; i < 1000; i++)  recv(x[i],P0);  T2 = Time();  TP1 = T2- T1;  send(TP1, P0); |

The first program will send the array x using 1 message and the second program will send the elements of the array x one by one using a loop. Array size = 1000. Write a report with a graph for TP0 (Time of Sending) and TP1 (Time of Receiving) showing the dependence of the transmission time on the size of the message for both programs. Determine which method is better for sending and receiving (Program 1 or Program 2)

Program1:

A screen shot of a computer

Description automatically generated

Code:

#include <mpi.h>

#include <stdio.h>

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

{

MPI\_Init(&*argc*, &*argv*);

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

if (world\_size != 2)

{

fprintf(stderr, "World size must be two for %s\n", *argv*[0]);

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

*const* int ARRAY\_SIZE = 1000;

int array[ARRAY\_SIZE]; *// Or char, double as needed*

double start\_time, end\_time, time\_taken;

if (world\_rank == 0)

{

*// P0: Send array*

start\_time = MPI\_Wtime();

MPI\_Send(array, ARRAY\_SIZE, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

end\_time = MPI\_Wtime();

time\_taken = end\_time - start\_time;

printf("P0 Sending Time: %f\n", time\_taken);

*// P0: Receive time from P1*

MPI\_Recv(&time\_taken, 1, MPI\_DOUBLE, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("P1 Receiving Time: %f\n", time\_taken);

}

else if (world\_rank == 1)

{

*// P1: Receive array*

start\_time = MPI\_Wtime();

MPI\_Recv(array, ARRAY\_SIZE, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

end\_time = MPI\_Wtime();

time\_taken = end\_time - start\_time;

printf("P1 Receiving Time: %f\n", time\_taken);

*// P1: Send time to P0*

MPI\_Send(&time\_taken, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

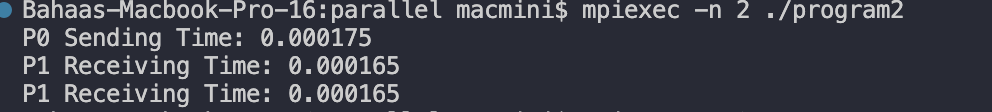
}

MPI\_Finalize();

return 0;

}

Program 2:



Code:

#include <mpi.h>

#include <stdio.h>

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

{

MPI\_Init(&*argc*, &*argv*);

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

if (world\_size != 2)

{

fprintf(stderr, "World size must be two for %s\n", *argv*[0]);

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

*const* int ARRAY\_SIZE = 1000;

int array[ARRAY\_SIZE]; *// Or char, double as needed*

double start\_time, end\_time, time\_taken;

if (world\_rank == 0)

{

*// P0: Send each array element*

start\_time = MPI\_Wtime();

for (int i = 0; i < ARRAY\_SIZE; i++)

{

MPI\_Send(&array[i], 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

}

end\_time = MPI\_Wtime();

time\_taken = end\_time - start\_time;

printf("P0 Sending Time: %f\n", time\_taken);

*// P0: Receive time from P1*

MPI\_Recv(&time\_taken, 1, MPI\_DOUBLE, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("P1 Receiving Time: %f\n", time\_taken);

}

else if (world\_rank == 1)

{

*// P1: Receive each array element*

start\_time = MPI\_Wtime();

for (int i = 0; i < ARRAY\_SIZE; i++)

{

MPI\_Recv(&array[i], 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

end\_time = MPI\_Wtime();

time\_taken = end\_time - start\_time;

printf("P1 Receiving Time: %f\n", time\_taken);

*// P1: Send time to P0*

MPI\_Send(&time\_taken, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

Comparison:

A screen shot of a computer

Description automatically generated

**Report**: Consistent testing results indicate a significant performance disparity between the two programs. Program 1 outshines Program 2 with a notably faster sending and receiving rate. The sending time of Program 1 is 10.94 times quicker, and its receiving time is 18.33 times more rapid than those of Program 2. This translates to, for every 5 milliseconds Program 1 requires for sending, Program 2 would need approximately 54.7 milliseconds, and for receiving, Program 2 would require around 91.65 milliseconds. The iterative sending and receiving of elements in Program 2 leads to this substantial time increase. Hence, for efficiency and speed, Program 1 is the superior choice.