Parallel Programming L3

REPORT PW5

Nafila Amirli

Ex2

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mpi.h>
int main(int argc, char **argv)
{
    int rank, size;
    MPI_Init(&argc, &argv);                /* starts MPI */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* get current process id */
    if (rank % 2 == 0)
    {
        printf("I am process %d, my rank is even\n", rank);
    }
    else if (rank % 2 == 1)
    {
        printf("I am process %d, my rank is odd\n", rank);
    }
    MPI_Comm_size(MPI_COMM_WORLD, &size); /* get number of processes */
    MPI_Finalize();
    return 0;
}
```

Terminal:

```
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpicc ex2.c -o ex2
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpiexec -n 4 ./ex2
I am process 0, my rank is even
I am process 1, my rank is odd
I am process 2, my rank is even
I am process 3, my rank is odd
```

Ex3

Write a MPI program with two processes in such a way that the process 0 sends a message (array of float) containing 1000 random reals to process 1.

Code:

```c
if (rank == 0)
{
    // int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
    // Sending, from the address buf, a message of count elements of type
    // datatype, tagged tag, to the process of rank dest in the communicator
    // comm.
    code = MPI_Send(my_array, 100, MPI_FLOAT, 1, tag, MPI_COMM_WORLD);
}
else if (rank == 1)
{
    // int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm,
    // MPI_Status *status)
    // Receiving, at the address buf, a message of count elements of type
    // datatype, tagged tag, from the process of rank source in the communicator comm.
    code = MPI_Recv(recieve, 100, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
    printf("Process 1, has received the array from process 0.\n");

    for (int i = 0; i < 100; i++)
    {
        printf("array[%d]=%f.\n", i, recieve[i]);
    }
    // printf("I am process %d, my rank is odd\n", rank);
}
```

Terminal :

```
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpicc ex3.c
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpiexec -n 2 ./a.out
Process 1, has received the array from process 0.
array[0]=1069849856.000000.
array[1]=2021045248.000000.
array[2]=1849161984.000000.
array[3]=1192452480.000000.
array[4]=417547520.000000.
array[5]=499209728.000000.
array[6]=751635712.000000.
array[7]=1581998336.000000.
array[8]=2039204352.000000.
array[9]=710744064.000000.
```

Complete the script in such a way that the process 1 sends back the message to the process 0, and measure the communication duration with the MPI_Wtime() function.

Code and terminal:

```
21      MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* get current process id */
22      if (rank == 0)
23      {
24          code = MPI_Send(my_array, N, MPI_FLOAT, 1, tag, MPI_COMM_WORLD);
25          code = MPI_Recv(my_array, N, MPI_FLOAT, 1, tag, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
26          printf("array recieved back by process 0\n");
27      }
28      else if (rank == 1)
29      {
30          code = MPI_Recv(recieve, N, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
31          printf("Process 1, has received the array from process 0.\n");
32
33          for (int i = 0; i < N; i++)
34          {
35              printf("array[%d]=%f.\n", i, recieve[i]);
36          }
37          code = MPI_Send(recieve, N, MPI_FLOAT, 0, tag, MPI_COMM_WORLD);
38      }
39      MPI_Comm_size(MPI_COMM_WORLD, &size); /* get number of processes */
40      double t2 = MPI_Wtime();
41      printf("MPI_Wtime measured : %f\n", t2 - t1);
```

```
TERMINAL  PROBLEMS 6  OUTPUT  DEBUG CONSOLE                    1: bash

array[994]=225118144.000000.
array[995]=1966690176.000000.
array[996]=1196757760.000000.
array[997]=1790477568.000000.
array[998]=1036508224.000000.
array[999]=1969647232.000000.
MPI_Wtime measured : 0.022221
array recieved back by process 0
MPI_Wtime measured : 0.022295
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$
```

Investigate how back-and-forth message passing time varies with the size of the message (change the number and the size of the messages to get meaningful measures) by reporting performance numbers in a spreadsheet file (e.g., LibreOffice Calc) to generate a graph to be discussed. The execution of the program on your local machine is used for measuring the throughput and the latency of your memory.

When N is 1000:

```
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpicc ex3_3.c
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpiexec -n 2 ./a.out
Time=0.003437
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpiexec -n 2 ./a.out
Time=0.001301
```

N=100:

```
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpicc ex3_3.c
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpiexec -n 2 ./a.out
Time=0.000742
```

N=10000:

```
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpicc ex3_3.c
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpiexec -n 2 ./a.out
Time=0.016841
```

N=100000

```
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpicc ex3_3.c
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$ mpiexec -n 2 ./a.out
Time=0.076747
nafila@nafila-Lenovo-V110-15ISK:~/paralel/PW5$
```

So, when the size increases the execution time also increases.