A6 - Message Passing Interface (MPI)

Team Members
Meghana Gudivada : 16CO112
Likitha Tejaswi Bandaru : 16CO105

Q1

Given below is the output

```
Hello World! from process of Rank 3 out of 10 no of processes, running on dell
Hello World! from process of Rank 7 out of 10 no of processes, running on dell
Hello World! from process of Rank 5 out of 10 no of processes, running on dell
Hello World! from process of Rank 4 out of 10 no of processes, running on dell
Hello World! from process of Rank 2 out of 10 no of processes, running on dell
Hello World! from process of Rank 1 out of 10 no of processes, running on dell
Hello World! from process of Rank 6 out of 10 no of processes, running on dell
Hello World! from process of Rank 0 out of 10 no of processes, running on dell
Hello World! from process of Rank 9 out of 10 no of processes, running on dell
Hello World! from process of Rank 8 out of 10 no of processes, running on dell
```

Number of threads is 10. Hence “Hello World !” is printed 10 times.

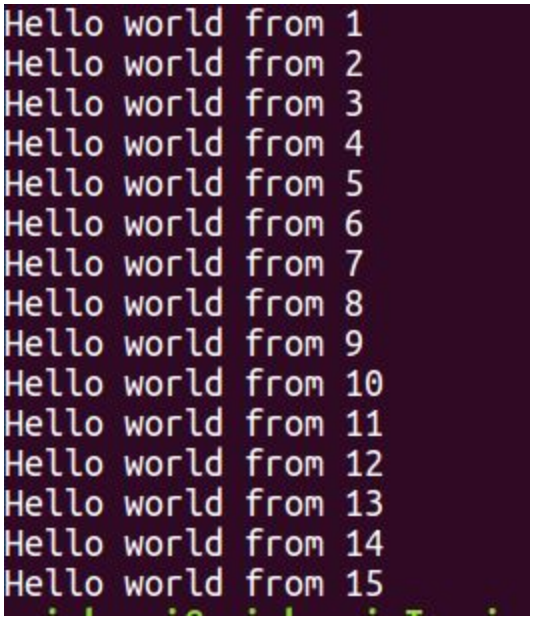
Q2

Given below is the output

```
Time taken by 2 processes is 0.111043 seconds
Speedup for 2 processes is 2.399136
Time taken by 4 processes is 0.124960 seconds
Speedup for 4 processes is 2.689064
Time taken by 8 processes is 0.650347 seconds
Speedup for 8 processes is 7.461200
Time taken by 16 processes is 12.127924 seconds
Speedup for 16 processes is 62.551280
```

Speedup is shown in the output.

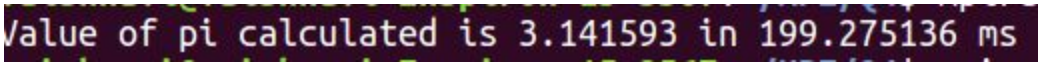
Q3



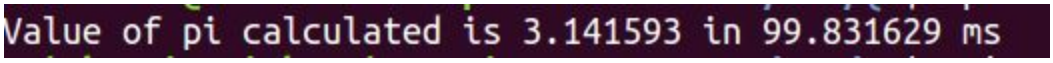
“Hello world” message is sent to master process from the other 15 processes.

Q4

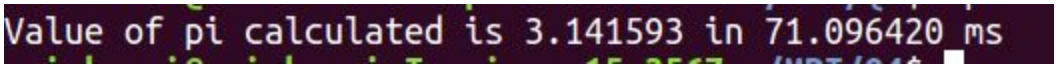
No of processes=1



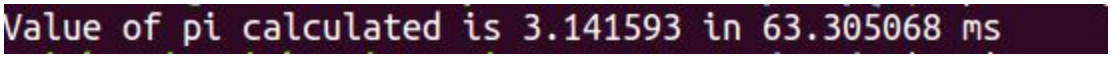
No of processes=2



No of processes=3



No of processes=4



No of processes=5

No of processes=6

No of processes=7

Value of pi is calculated. The time taken by different number of processes is shown in the output.

Q5

No of Processes used is 5

```
Sum of first 1000000 natural numbers is 500000500000
```

Sum of N = 1000000, is shown in the output.

Q6

```
The old array:
1.000000 4.000000 9.000000 16.000000 25.000000 36.000000 49.000000 64.000000 81.000000 100.000000
The new array:
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000 10.000000
```

The square root of the old array is calculated by scattering its elements into different processes, calculating square root there and gathering it back at the master process.

Q7

```
Collective Communication Rank : 0 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Collective Communication Rank : 1 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Collective Communication Rank : 8 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Point-to-point communication Rank : 8 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Collective Communication Rank : 2 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Collective Communication Rank : 9 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Point-to-point communication Rank : 9 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Collective Communication Rank : 4 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Point-to-point communication Rank : 4 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Collective Communication Rank : 6 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Point-to-point communication Rank : 6 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Collective Communication Rank : 7 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Point-to-point communication Rank : 7 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Point-to-point communication Rank : 2 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Point-to-point communication Rank : 1 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Collective Communication Rank : 5 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Point-to-point communication Rank : 5 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
Collective Communication Rank : 3 Structure : 1 - 49 50 - 49.000000 50.000000 51.000000 52.000000
Point-to-point communication Rank : 3 Structure : a - 97 98 - 97.000000 98.000000 99.000000 100.000000
```

The following code snippet is used to fill the structure

```
struct dd get_filled_struct(char key)
{
        struct dd temp;
        temp.c = key;
        for(int i=0;i<4;++i)
        {
```

```
            if(i<2)
            {
                temp.iA[i] = key + i;
            }
            temp.fA[i] = key + i;
        }
        return temp;
}
```

The derived datatype was created and broadcasted by the master process, whose output is shown prefixed with *Collective Communication.*
Also, the master process invidually send the datatype to the other process, whose output is prefixed by *Point-to-point communication.*

Q8

```
Packed in Rank : 0 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 1 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 2 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 8 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 3 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 4 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 5 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 9 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 6 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
Unpacked - Rank : 7 Values : A - 65 66 - 65.000000 66.000000 67.000000 68.000000
```

The same problem statement in Q7, is done using the Pack and Unpack functions in MPI.

Q9

```
Sent Matrix
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
Received Matrix
0 1 2 3
0 5 6 7
0 0 10 11
0 0 0 15
```

An indexed derived datatype which taken only the upper triangle of a matrix is declared. The above shown matrix is send my the master process using this derived datatype to process 1.
The matrix received by process 1 is an upper triangular matrix as shown in the output.

Q10

```
n 9 a.out
Operand matrix A
1 2 3
4 5 6
7 8 9
Operand matrix B
1 2 3
4 5 6
7 8 9
Resultant matrix C
30 36 42
66 81 96
102 126 150
```

```
n 16 a.out
Operand matrix A
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Operand matrix B
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Resultant matrix C
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```

```
n 25 a.out
Operand matrix A
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
Operand matrix B
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
Resultant matrix C
215 230 245 260 275
490 530 570 610 650
765 830 895 960 1025
1040 1130 1220 1310 1400
1315 1430 1545 1660 1775
```

The cannon's algorithm was implemented. The output is shown for 3x3, 4x4 and 5x5 matrices respectively.
The matrices are filled using the following code snippet

```
for(int i = 0; i < N; ++i)
    for(int j = 0; j < N; ++j)
        arr[i][j] = i*N + j;
```

To change the dimension of the matrix do the following
1. #define N 4 : Change this macro in line 3 of the code
2. while running run : mpirun -n 25 a.out, where 25 is for a 5x5 matrix.