CSE4001 - Parallel and Distributed Computing

Lab 21+22

Lab FAT

Submitted by: Alokam Nikhitha

Reg No:19BCE2555

QUESTION 1:

1. Write a c program that uses * and numerical value (1 to 5) to print the following pattern (shown below). To print the left and right parts of the pattern, use nested loops. **[20 marks]**

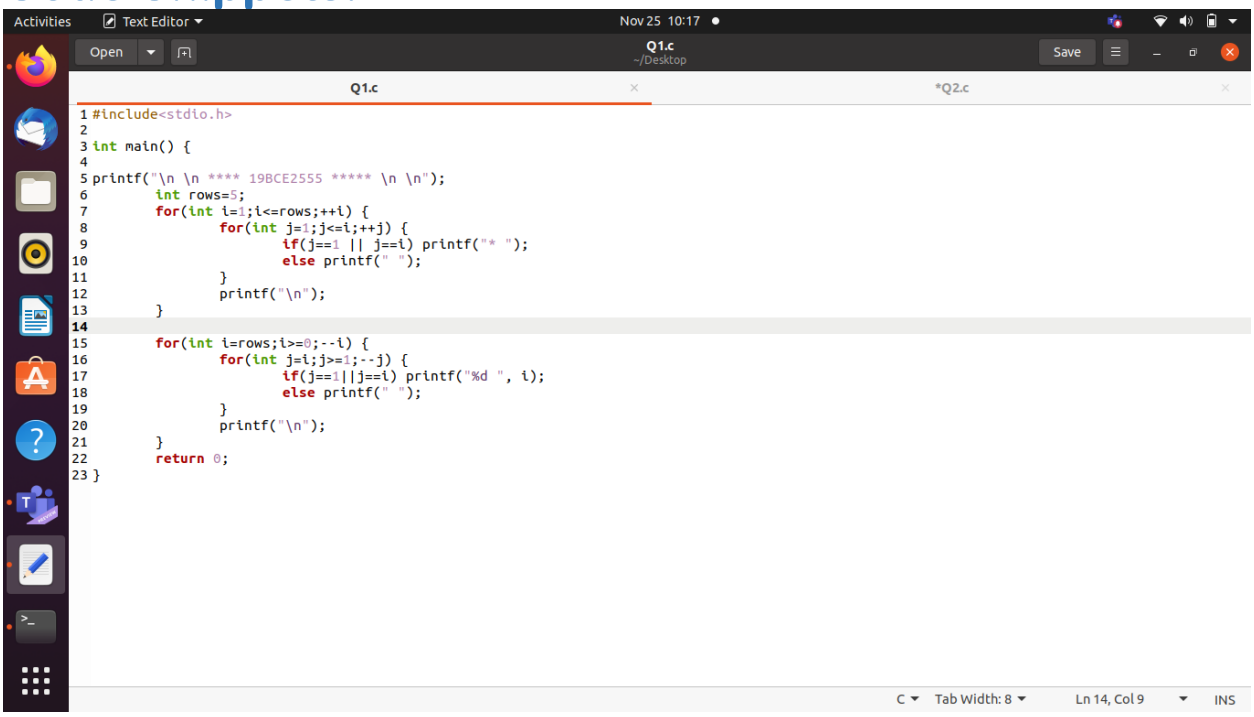
(a)

```
*
*   *
*     *
*       *
*         *
5           5
4         4
3       3
2     2
1
```

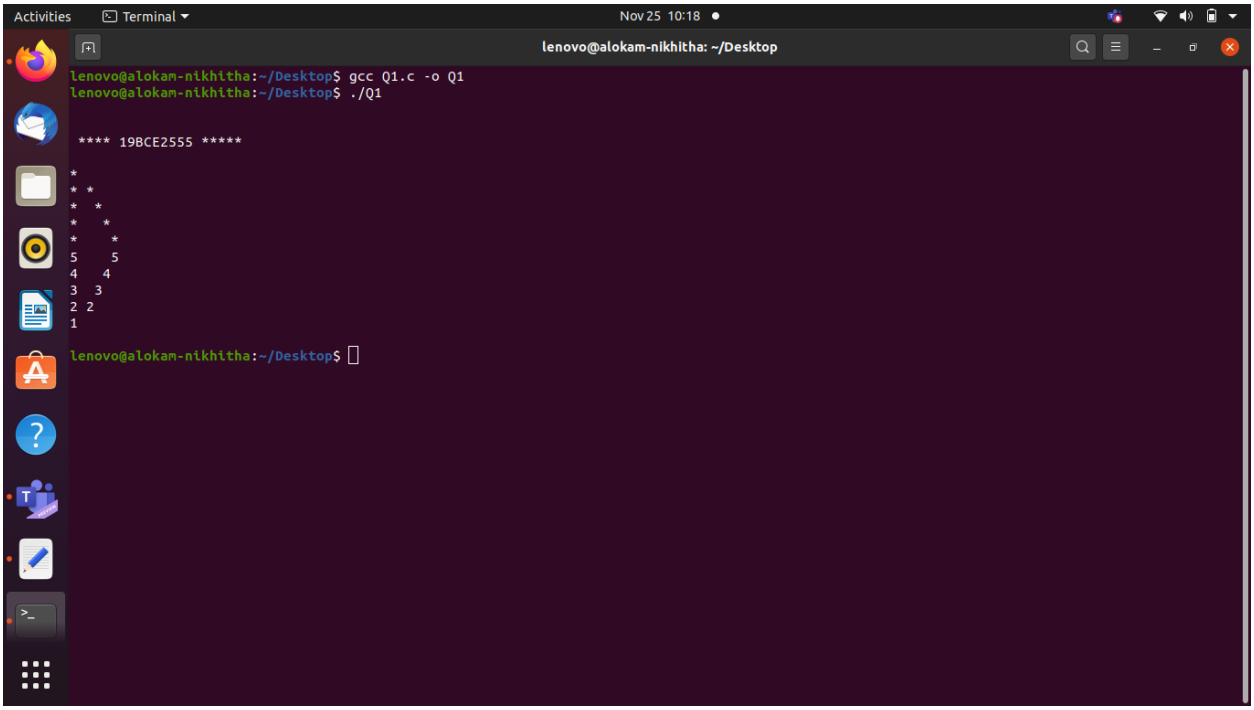
CODE:

```
#include<stdio.h>
int main() {
printf("\n \n ** 19BCE2555 *** \n \n");
        int rows=5;
        for(int i=1;i<=rows;++i) {
                for(int j=1;j<=i;++j) {
                        if(j==1 || j==i) printf("* ");
                        else printf(" ");
                }
                printf("\n");
        }

        for(int i=rows;i>=0;--i) {
                for(int j=i;j>=1;--j) {
                        if(j==1||j==i) printf("%d ", i);
                        else printf(" ");
                }
                printf("\n");
        }
        return 0;
}
```

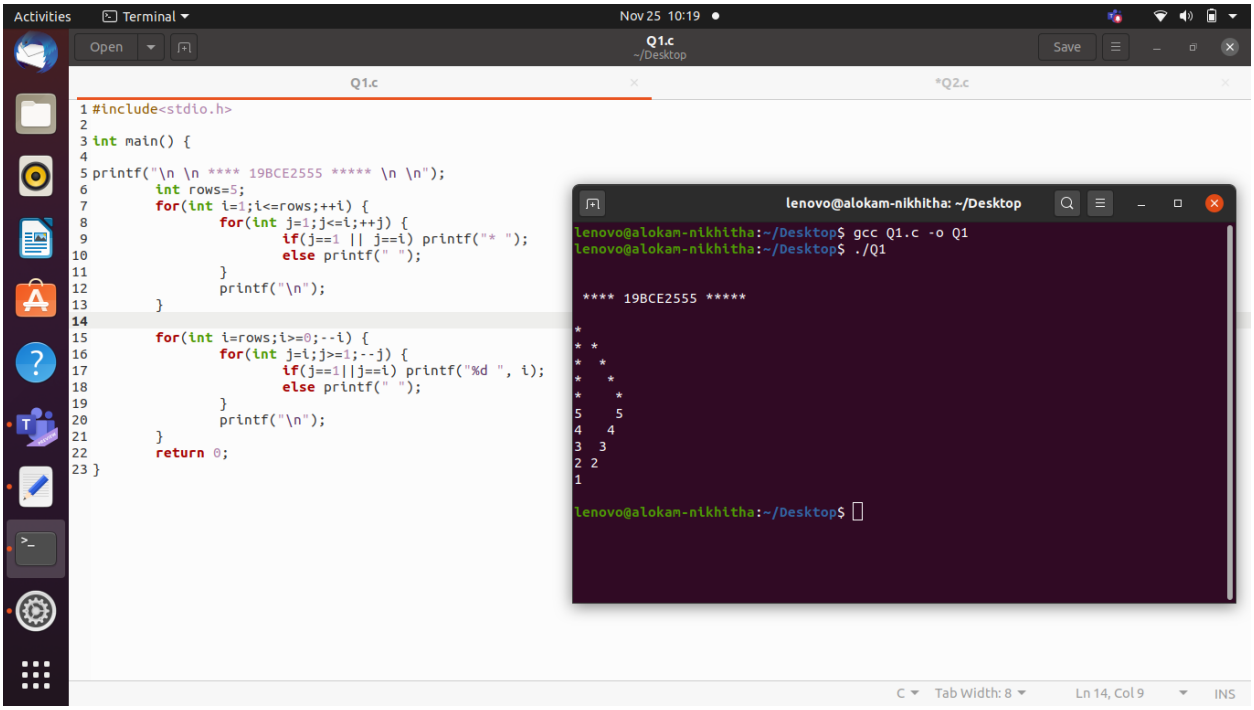
Code Snippets:



OUTPUT:



OUTPUT WITH CODE:



QUESTION 2:

2. Write a C program to perform parallel matrix multiplication using OpenMP. You should first create three matrices, X, Y, and Z, and then initialise X and Y with the values listed below.

$$X = \begin{bmatrix} 1 & 4 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \qquad Y = \begin{bmatrix} 4 & 1 & 2 & 1 \\ 0 & 1 & -1 & 3 \end{bmatrix}$$

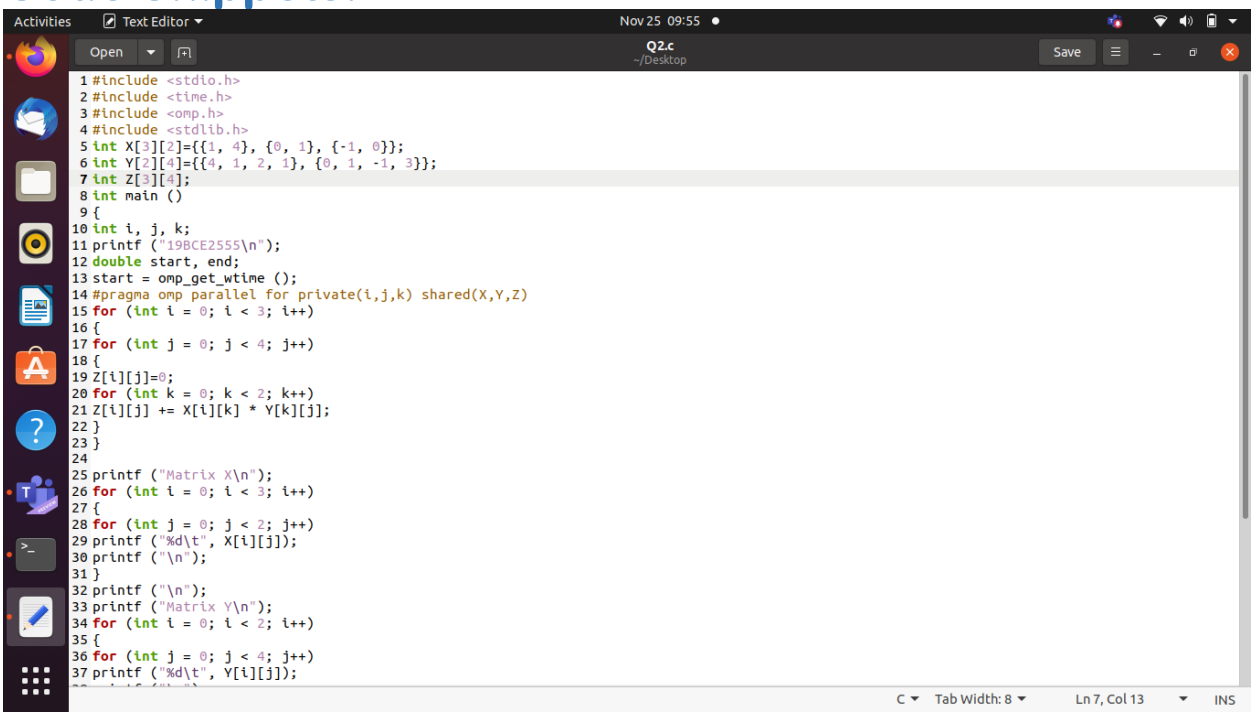
In your code, try to improve the performance by (re)using the same set of threads for initializing X and Y and for calculating Z. **[40 Marks]**

CODE:

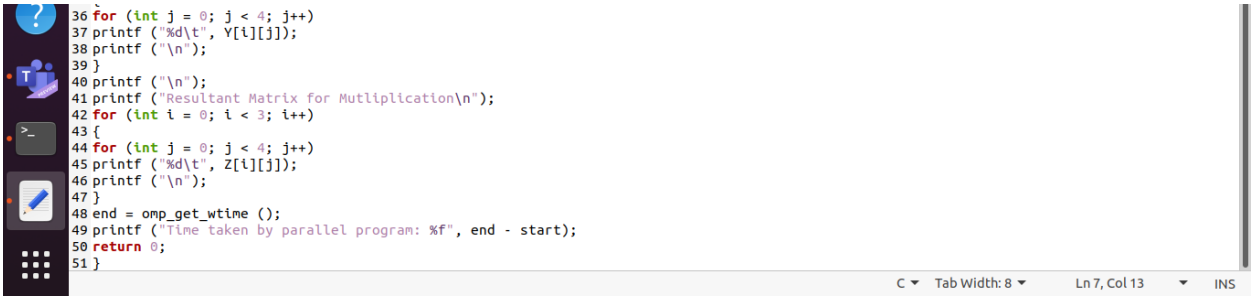
```
#include <stdio.h>
#include <time.h>
#include <omp.h>
#include <stdlib.h>
int X[3][2]={{1, 4}, {0, 1}, {-1, 0}};
int Y[2][4]={{4, 1, 2, 1}, {0, 1, -1, 3}};
int Z[3][4];
int main ()
{
int i, j, k;
printf ("19BCE2555\n");
double start, end;
start = omp_get_wtime ();
#pragma omp parallel for private(i,j,k) shared(X,Y,Z)
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 4; j++)
{
Z[i][j]=0;
for (int k = 0; k < 2; k++)
Z[i][j] += X[i][k] * Y[k][j];
}
```

```
}

printf ("Matrix X\n");
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 2; j++)
printf ("%d\t", X[i][j]);
printf ("\n");
}
printf ("\n");
printf ("Matrix Y\n");
for (int i = 0; i < 2; i++)
{
for (int j = 0; j < 4; j++)
printf ("%d\t", Y[i][j]);
printf ("\n");
}
printf ("\n");
printf ("Resultant Matrix for Mutliplication\n");
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 4; j++)
printf ("%d\t", Z[i][j]);
printf ("\n");
}
end = omp_get_wtime ();
printf ("Time taken by parallel program: %f", end - start);
return 0;
}
```

Code Snippets:

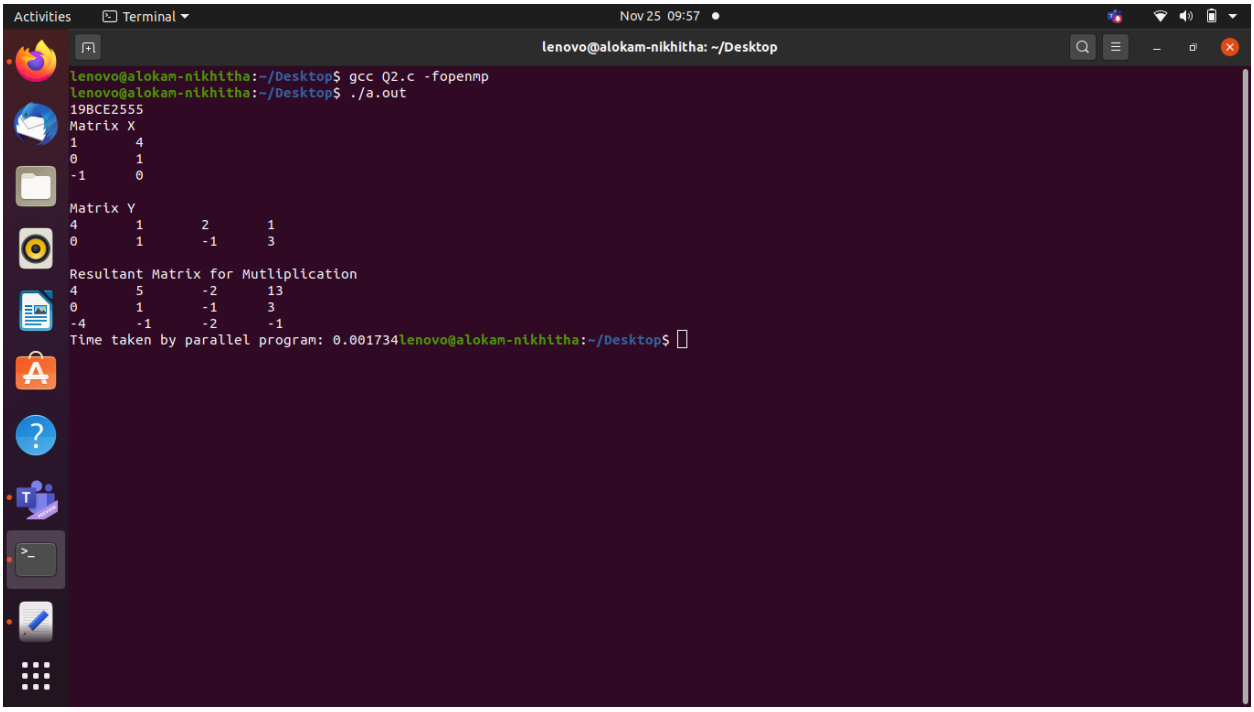



```c
#include <stdio.h>
#include <time.h>
#include <omp.h>
#include <stdlib.h>
int X[3][2]={{1, 4}, {0, 1}, {-1, 0}};
int Y[2][4]={{4, 1, 2, 1}, {0, 1, -1, 3}};
int Z[3][4];
int main ()
{
int i, j, k;
printf ("19BCE2555\n");
double start, end;
start = omp_get_wtime ();
#pragma omp parallel for private(i,j,k) shared(X,Y,Z)
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 4; j++)
{
Z[i][j]=0;
for (int k = 0; k < 2; k++)
Z[i][j] += X[i][k] * Y[k][j];
}
}

printf ("Matrix X\n");
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 2; j++)
printf ("%d\t", X[i][j]);
printf ("\n");
}
printf ("\n");
printf ("Matrix Y\n");
for (int i = 0; i < 2; i++)
{
for (int j = 0; j < 4; j++)
printf ("%d\t", Y[i][j]);
```

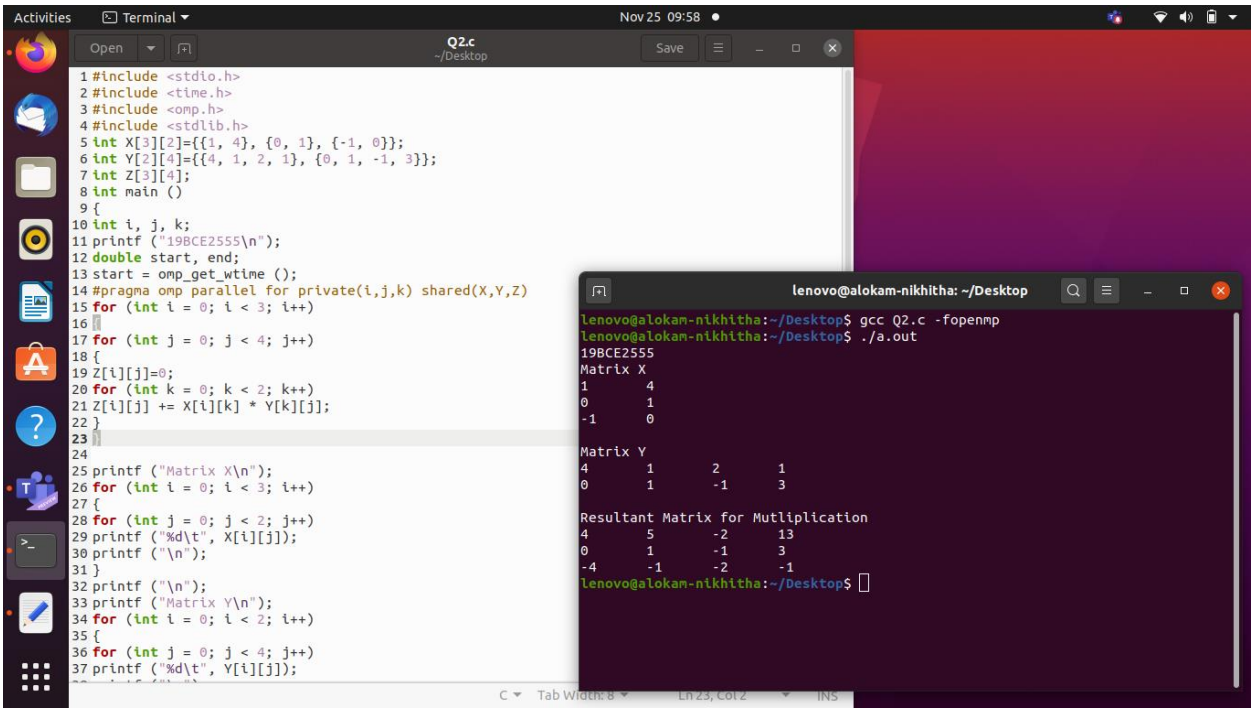


```c
for (int j = 0; j < 4; j++)
printf ("%d\t", Y[i][j]);
printf ("\n");
}
printf ("\n");
printf ("Resultant Matrix for Mutliplication\n");
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 4; j++)
printf ("%d\t", Z[i][j]);
printf ("\n");
}
end = omp_get_wtime ();
printf ("Time taken by parallel program: %f", end - start);
return 0;
}
```

OUTPUT:

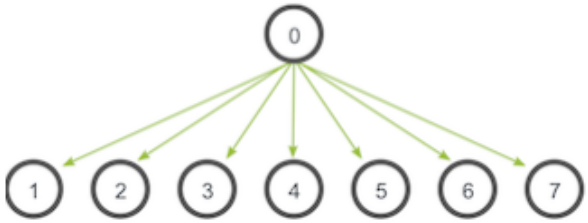


OUTPUT WITH CODE:



QUESTION 3:

3. Create a C programme to initialise the broadcast communication pattern on the MPI application interface. The code logic can typically have a process zero [as root], which has the initial copy of the data to broadcast to other processes [as shown in the below figure]. Show the MPI_Bcast time with different number of processors. **[40 Marks]**



CODE:

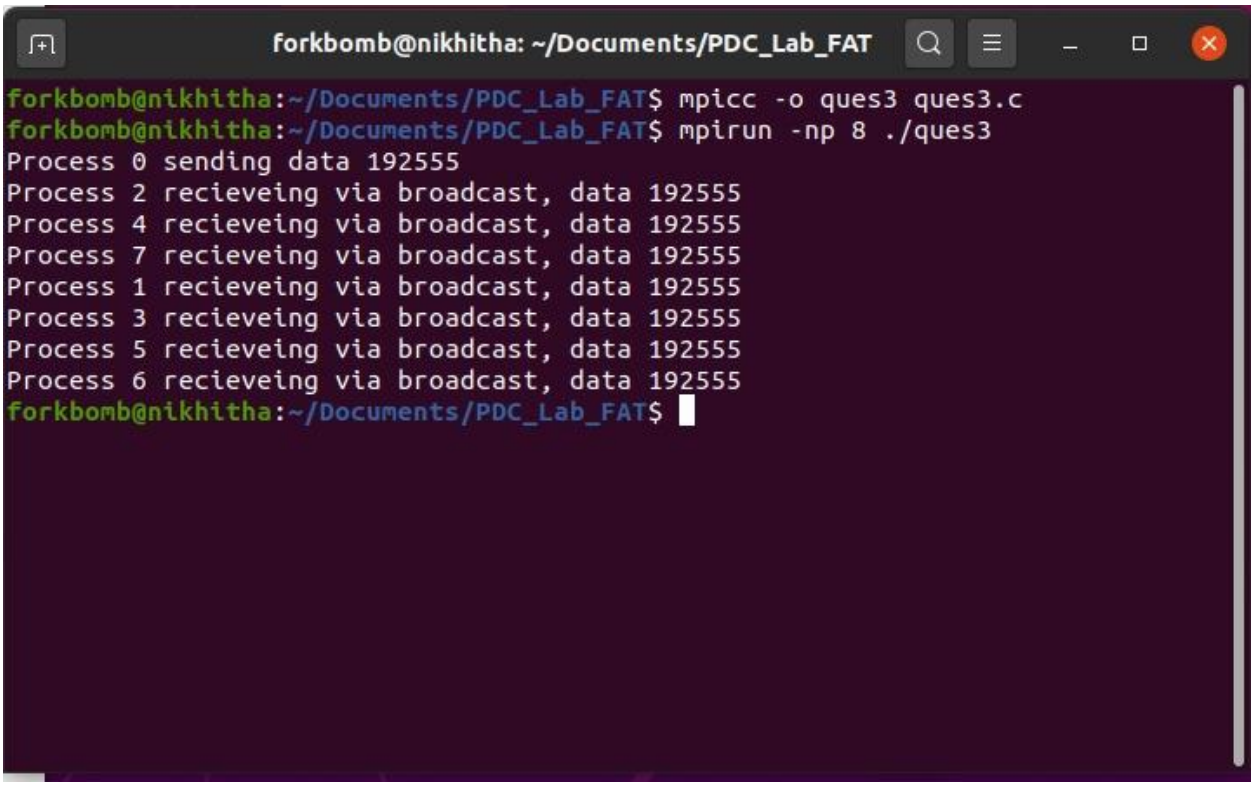
```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>

int main(int argc, char*argv[] ) {
        MPI_Init(&argc, &argv);
        int my_rank;
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
        int broadcast_root=0;
        int buffer;
        if(my_rank==broadcast_root) {
                buffer=192555;
                printf("Process %d sending data %d\n", my_rank, buffer);
        }
        MPI_Bcast(&buffer,1,MPI_INT, broadcast_root, MPI_COMM_WORLD);
        if(my_rank!=broadcast_root) {
                printf("Process %d recieveing via broadcast, data %d\n", my_rank,
buffer);
        }
        MPI_Finalize();
        return EXIT_SUCCESS;
}
```

Code Snippets:

ques3.c
~/Documents/PDC_Lab_FAT

```c
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>

int main(int argc, char*argv[] ) {
        MPI_Init(&argc, &argv);
        int my_rank;
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
        int broadcast_root=0;
        int buffer;
        if(my_rank==broadcast_root) {
                buffer=192555;
                printf("Process %d sending data %d\n", my_rank, buffer);
        }
        MPI_Bcast(&buffer,1,MPI_INT, broadcast_root, MPI_COMM_WORLD);
        if(my_rank!=broadcast_root) {
                printf("Process %d recieveing via broadcast, data %d\n", my_rank, buffer);
        }
        MPI_Finalize();
        return EXIT_SUCCESS;
}
```

OUTPUT:



OUTPUT WITH CODE: