

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)

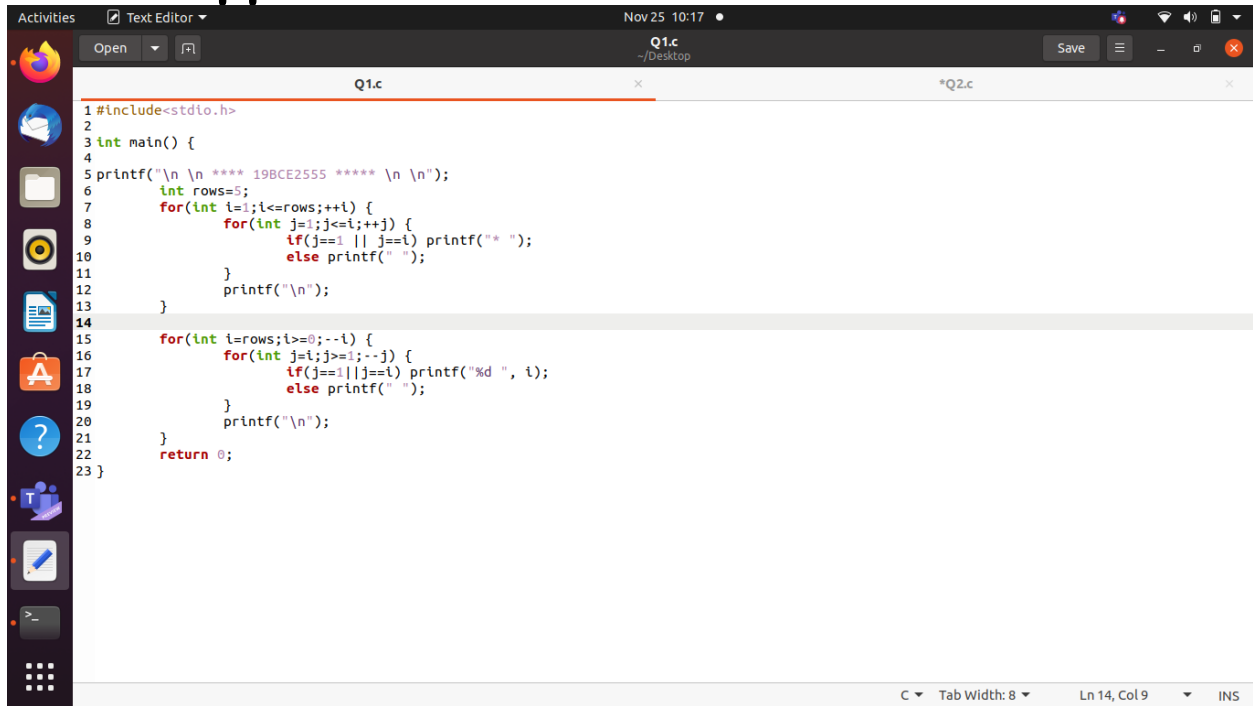
```
*
*  *
*   *
*    *
*     *
5      5
4       4
3        3
2         2
1
```

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:

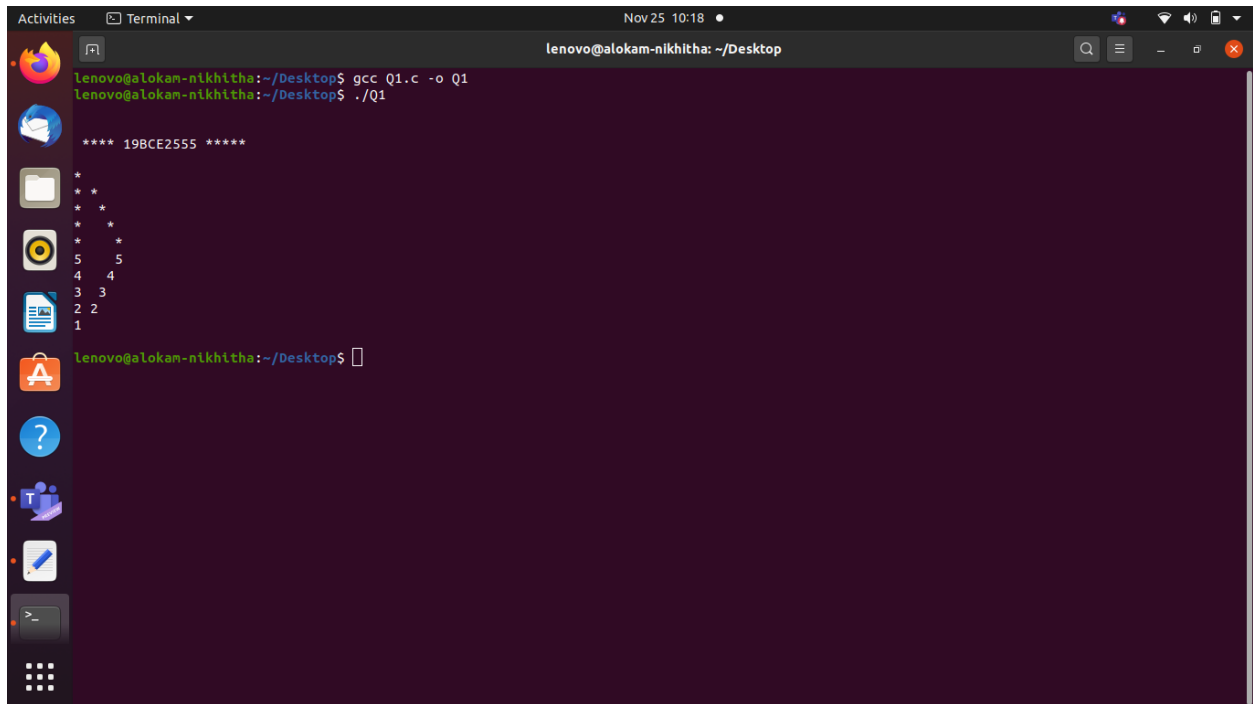


The screenshot shows a text editor window titled 'Q1.c' with the following C code:

```
1 #include<stdio.h>
2
3 int main() {
4
5     printf("\n \n **** 19BCE2555 ***** \n \n");
6     int rows=5;
7     for(int i=1;i<=rows;++i) {
8         for(int j=1;j<=i;++j) {
9             if(j==1 || j==i) printf("* ");
10            else printf(" ");
11        }
12        printf("\n");
13    }
14
15    for(int i=rows;i>=0;--i) {
16        for(int j=i;j>=1;--j) {
17            if(j==1||j==i) printf("%d ", i);
18            else printf(" ");
19        }
20        printf("\n");
21    }
22    return 0;
23 }
```

The code is a C program that prints a diamond pattern of asterisks and a sequence of numbers. The first part of the code prints a diamond shape with 5 rows. The second part prints a sequence of numbers from 5 down to 1, with spaces between them.

OUTPUT:



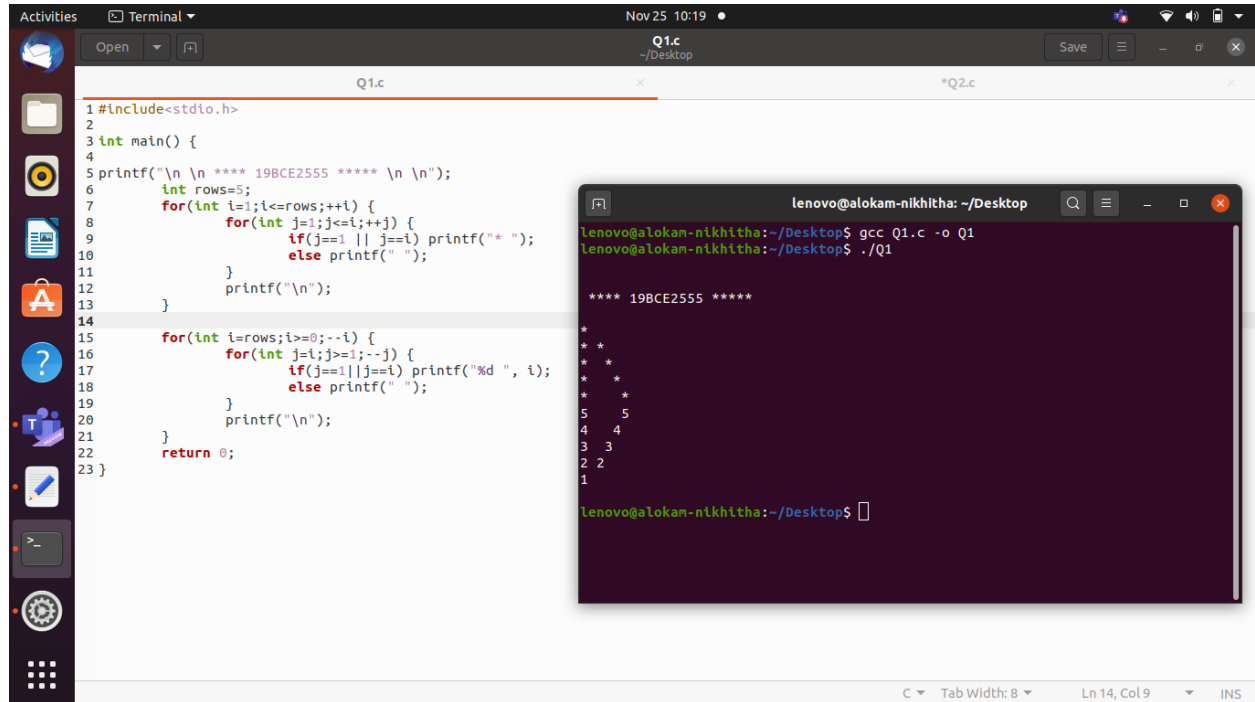
The screenshot shows a terminal window with the following output:

```
lenovo@alokam-nikhitha:~/Desktop$ gcc Q1.c -o Q1
lenovo@alokam-nikhitha:~/Desktop$ ./Q1
**** 19BCE2555 *****
*
* *
* * *
* * * *
* * * * *
5
4 4
3 3
2 2
1

lenovo@alokam-nikhitha:~/Desktop$
```

The output shows the diamond pattern of asterisks and the sequence of numbers from 5 down to 1, with spaces between them.

OUTPUT WITH CODE:



The image shows a code editor window with a C program and a terminal window showing its execution. The code uses nested loops to print a pattern of asterisks and numbers.

```
1#include<stdio.h>
2
3int main() {
4
5printf("\n \n **** 19BCE2555 ***** \n \n");
6    int rows=5;
7    for(int i=1;i<=rows;++i) {
8        for(int j=1;j<=i;++j) {
9            if(j==1 || j==i) printf("* ");
10           else printf(" ");
11        }
12        printf("\n");
13    }
14
15    for(int i=rows;i>=0;--i) {
16        for(int j=i;j>=1;--j) {
17            if(j==1||j==i) printf("%d ", i);
18            else printf(" ");
19        }
20        printf("\n");
21    }
22    return 0;
23 }
```

The terminal output shows the pattern generated by the code:

```
lenovo@alokam-nikhitha: ~/Desktop
lenovo@alokam-nikhitha:~/Desktop$ gcc Q1.c -o Q1
lenovo@alokam-nikhitha:~/Desktop$ ./Q1

**** 19BCE2555 *****

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

Result and Inferences:

- Nested loops are used in order to print the pattern.
- First loop prints the upper half of the pattern which prints * predominantly.
- Second loop prints the lower half of the pattern where it prints numbers from 5 to 1 in decreasing order.

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.

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

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]**

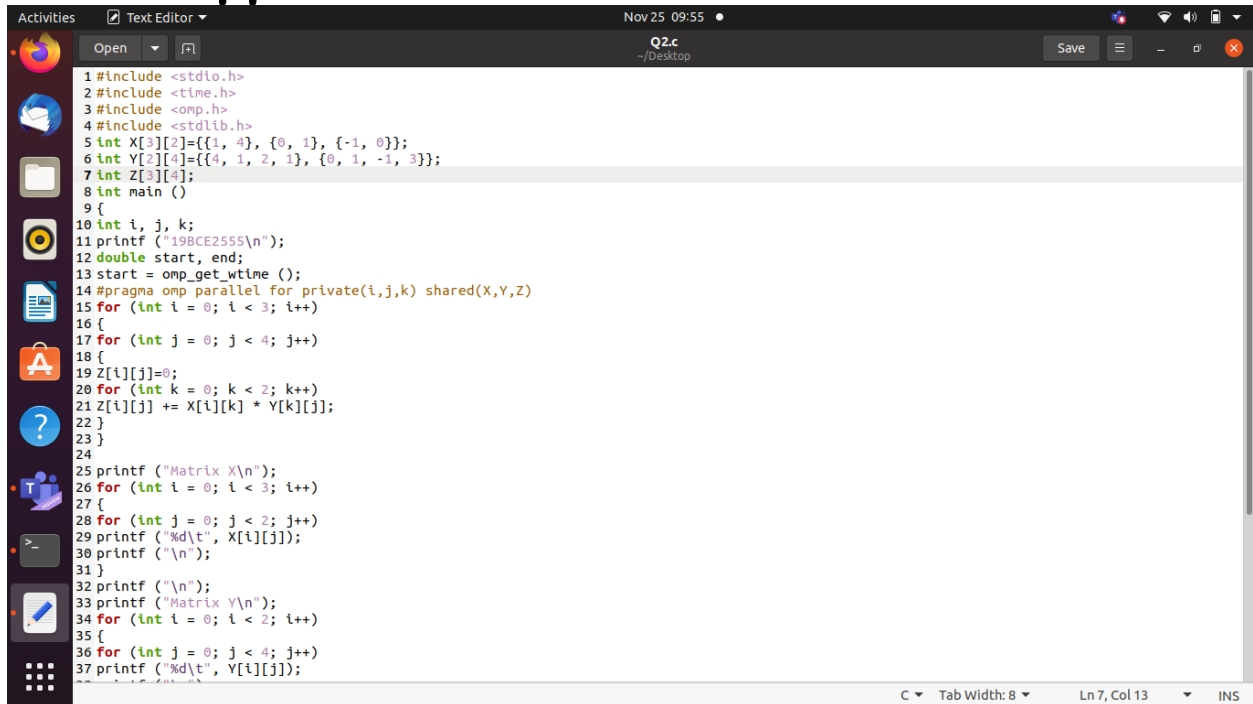
CODE:

```
#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;
}
```

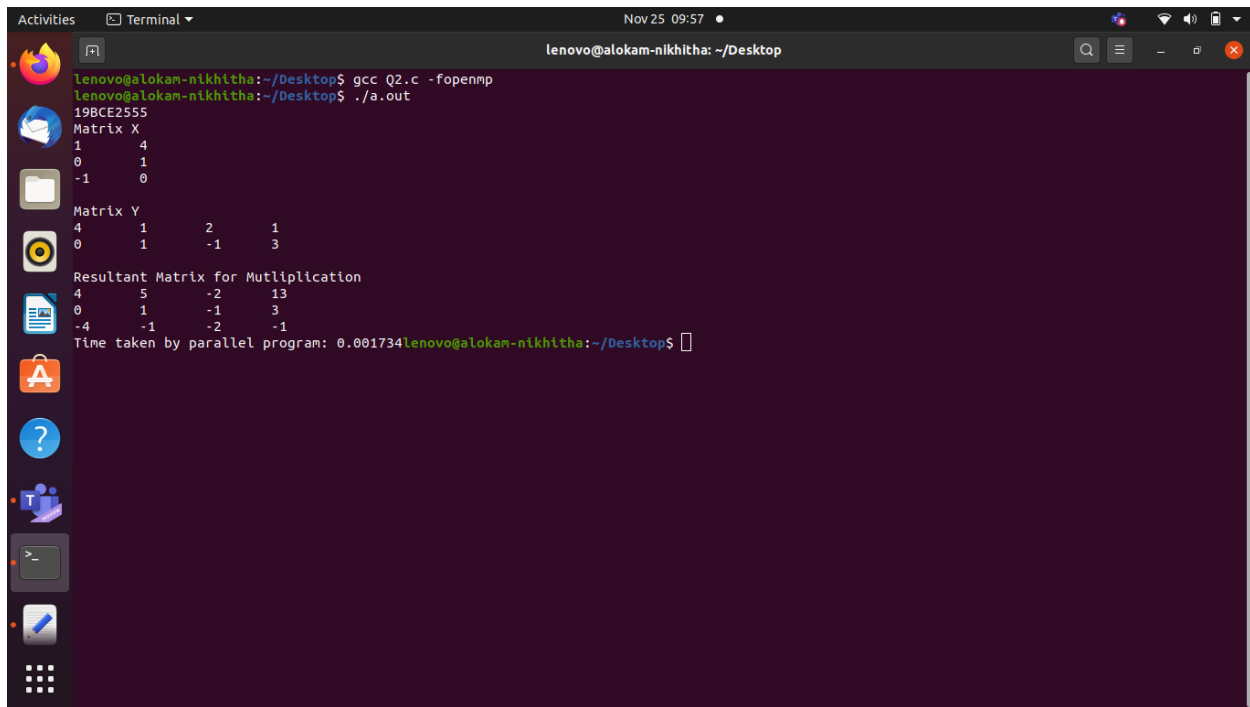
Code Snippets:



```
1 #include <stdio.h>
2 #include <time.h>
3 #include <omp.h>
4 #include <stdlib.h>
5 int X[3][2]={{1, 4}, {0, 1}, {-1, 0}};
6 int Y[2][4]={{4, 1, 2, 1}, {0, 1, -1, 3}};
7 int Z[3][4];
8 int main ()
9 {
10 int i, j, k;
11 printf ("19BCE2555\n");
12 double start, end;
13 start = omp_get_wtime ();
14 #pragma omp parallel for private(i,j,k) shared(X,Y,Z)
15 for (int i = 0; i < 3; i++)
16 {
17 for (int j = 0; j < 4; j++)
18 {
19 Z[i][j]=0;
20 for (int k = 0; k < 2; k++)
21 Z[i][j] += X[i][k] * Y[k][j];
22 }
23 }
24
25 printf ("Matrix X\n");
26 for (int i = 0; i < 3; i++)
27 {
28 for (int j = 0; j < 2; j++)
29 printf ("%d\t", X[i][j]);
30 printf ("\n");
31 }
32 printf ("\n");
33 printf ("Matrix Y\n");
34 for (int i = 0; i < 2; i++)
35 {
36 for (int j = 0; j < 4; j++)
37 printf ("%d\t", Y[i][j]);
38 }
```

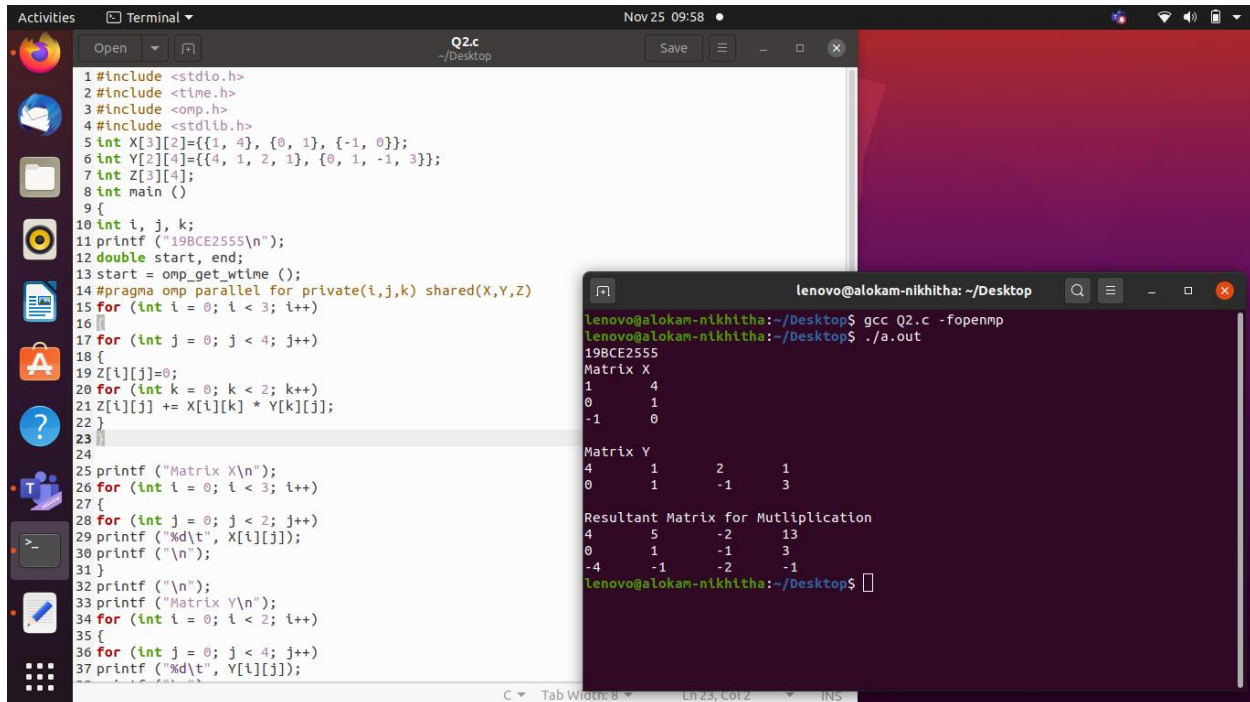
```
39 }
40 printf ("\n");
41 printf ("Resultant Matrix for Mutliplication\n");
42 for (int i = 0; i < 3; i++)
43 {
44 for (int j = 0; j < 4; j++)
45 printf ("%d\t", Z[i][j]);
46 printf ("\n");
47 }
48 end = omp_get_wtime ();
49 printf ("Time taken by parallel program: %f", end - start);
50 return 0;
51 }
```

OUTPUT:



```
lenovo@alokam-nikhitha: ~/Desktop
lenovo@alokam-nikhitha:~/Desktop$ gcc Q2.c -fopenmp
lenovo@alokam-nikhitha:~/Desktop$ ./a.out
19BCE2555
Matrix X
1 4
0 1
-1 0
Matrix Y
4 1 2 1
0 1 -1 3
Resultant Matrix for Mutlpllication
4 5 -2 13
0 1 -1 3
-4 -1 -2 -1
Time taken by parallel program: 0.001734lenovo@alokam-nikhitha:~/Desktop$
```

OUTPUT WITH CODE:



```
1#include <stdio.h>
2#include <time.h>
3#include <omp.h>
4#include <stdlib.h>
5int X[3][2]={{1, 4}, {0, 1}, {-1, 0}};
6int Y[2][4]={{4, 1, 2, 1}, {0, 1, -1, 3}};
7int Z[3][4];
8int main ()
9{
10 int i, j, k;
11 printf ("19BCE2555\n");
12 double start, end;
13 start = omp_get_wtime ();
14 #pragma omp parallel for private(i,j,k) shared(X,Y,Z)
15 for (int i = 0; i < 3; i++)
16 {
17 for (int j = 0; j < 4; j++)
18 {
19 Z[i][j]=0;
20 for (int k = 0; k < 2; k++)
21 Z[i][j] += X[i][k] * Y[k][j];
22 }
23 }
24
25 printf ("Matrix X\n");
26 for (int i = 0; i < 3; i++)
27 {
28 for (int j = 0; j < 2; j++)
29 printf ("%d\t", X[i][j]);
30 printf ("\n");
31 }
32 printf ("\n");
33 printf ("Matrix Y\n");
34 for (int i = 0; i < 2; i++)
35 {
36 for (int j = 0; j < 4; j++)
37 printf ("%d\t", Y[i][j]);
38 }
```

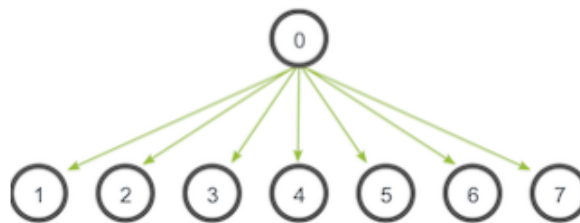
```
lenovo@alokam-nikhitha:~/Desktop$ gcc Q2.c -fopenmp
lenovo@alokam-nikhitha:~/Desktop$ ./a.out
19BCE2555
Matrix X
1 4
0 1
-1 0
Matrix Y
4 1 2 1
0 1 -1 3
Resultant Matrix for Mutlpllication
4 5 -2 13
0 1 -1 3
-4 -1 -2 -1
lenovo@alokam-nikhitha:~/Desktop$
```


Result and Inferences:

- Here we have initialized our X, Y and Z matrices globally for global access.
- We have then multiplied them parallelly using OpenMP.
- Time taken by parallel program to execute the same is 0.001734 milliseconds (ms).

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]**



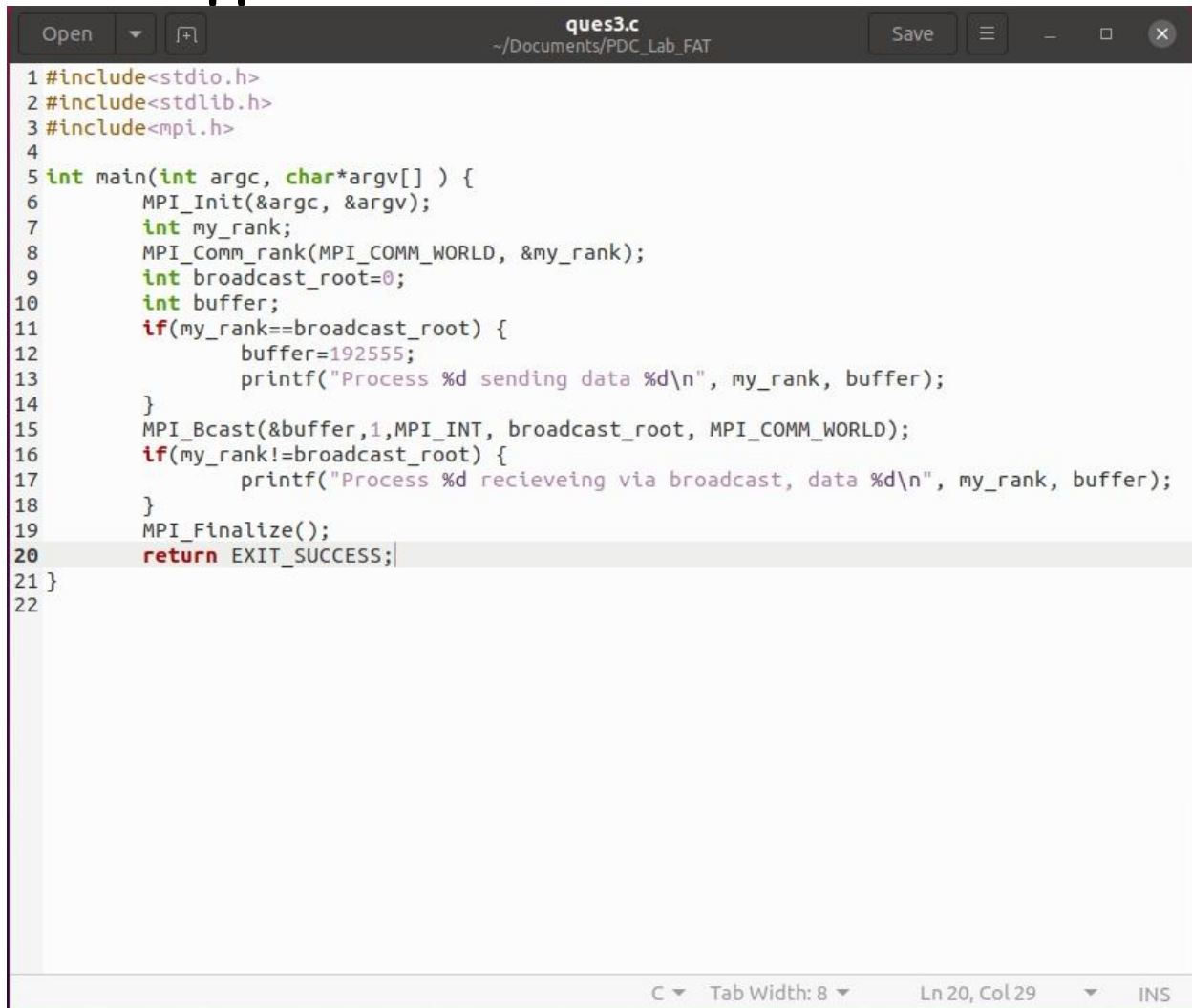
CODE:

```
#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  
Save  
1 #include<stdio.h>  
2 #include<stdlib.h>  
3 #include<mpi.h>  
4  
5 int main(int argc, char*argv[] ) {  
6     MPI_Init(&argc, &argv);  
7     int my_rank;  
8     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);  
9     int broadcast_root=0;  
10    int buffer;  
11    if(my_rank==broadcast_root) {  
12        buffer=192555;  
13        printf("Process %d sending data %d\n", my_rank, buffer);  
14    }  
15    MPI_Bcast(&buffer,1,MPI_INT, broadcast_root, MPI_COMM_WORLD);  
16    if(my_rank!=broadcast_root) {  
17        printf("Process %d recieveing via broadcast, data %d\n", my_rank, buffer);  
18    }  
19    MPI_Finalize();  
20    return EXIT_SUCCESS;  
21 }  
22  
C Tab Width: 8 Ln 20, Col 29 INS
```

OUTPUT:

```
forkbomb@nikhitha: ~/Documents/PDC_Lab_FAT
forkbomb@nikhitha:~/Documents/PDC_Lab_FAT$ mpicc -o ques3 ques3.c
forkbomb@nikhitha:~/Documents/PDC_Lab_FAT$ mpirun -np 8 ./ques3
Process 0 sending data 192555
Process 2 recieveing via broadcast, data 192555
Process 4 recieveing via broadcast, data 192555
Process 7 recieveing via broadcast, data 192555
Process 1 recieveing via broadcast, data 192555
Process 3 recieveing via broadcast, data 192555
Process 5 recieveing via broadcast, data 192555
Process 6 recieveing via broadcast, data 192555
forkbomb@nikhitha:~/Documents/PDC_Lab_FAT$
```

OUTPUT WITH CODE:

```
ques3.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<mpi.h>
4
5 int main(int argc, char*argv[]) {
6     MPI_Init(&argc, &argv);
7     int my_rank;
8     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
9     int broadcast_root=0;
10    int buffer;
11    if(my_rank==broadcast_root) {
12        buffer=192555;
13        printf("Process %d sending data %d\n", my_rank, buffer);
14    }
15    MPI_Bcast(&buffer,1,MPI_INT, broadcast_root, MPI_COMM_WORLD);
16    if(my_rank!=broadcast_root) {
17        printf("Process %d recieveing via broadcast, data %d\n", my_rank, buffer);
18    }
19    MPI_Finalize();
20    return EXIT_SUCCESS;
21 }
22
```

```
forkbomb@nikhitha: ~/Documents/PDC_Lab_FAT
forkbomb@nikhitha:~/Documents/PDC_Lab_FAT$ mpicc -o ques3 ques3.c
forkbomb@nikhitha:~/Documents/PDC_Lab_FAT$ mpirun -np 8 ./ques3
Process 0 sending data 192555
Process 2 recieveing via broadcast, data 192555
Process 4 recieveing via broadcast, data 192555
Process 7 recieveing via broadcast, data 192555
Process 1 recieveing via broadcast, data 192555
Process 3 recieveing via broadcast, data 192555
Process 5 recieveing via broadcast, data 192555
Process 6 recieveing via broadcast, data 192555
forkbomb@nikhitha:~/Documents/PDC_Lab_FAT$
```

Result and Inferences:

- We used Broadcast method of mpi to demonstrate the broadcasting of a message.
- The message broadcasted is 192555, (corresponding to registration number 19BCE2555).
- We can see that the process 0 is broadcasting the above number. All the other processes numbered from 1 to 7 are receiving the broadcasted message 192555.