Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2021-22 **Semester:** 1

Course: High Performance Computing lab

ESE Exam

22/11/2021 01.00 PM – 04.00 PM

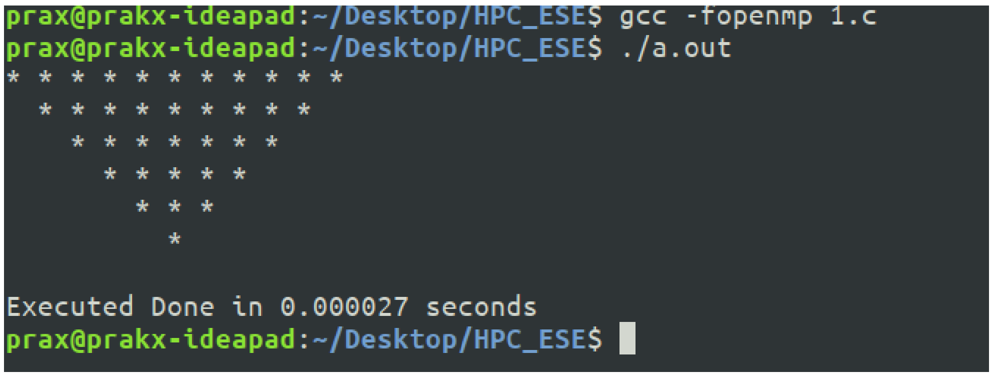
Exam Seat No:2018BTECS00100

Name:Prakash Singh
Exam Seat Number:2018BTECS00100

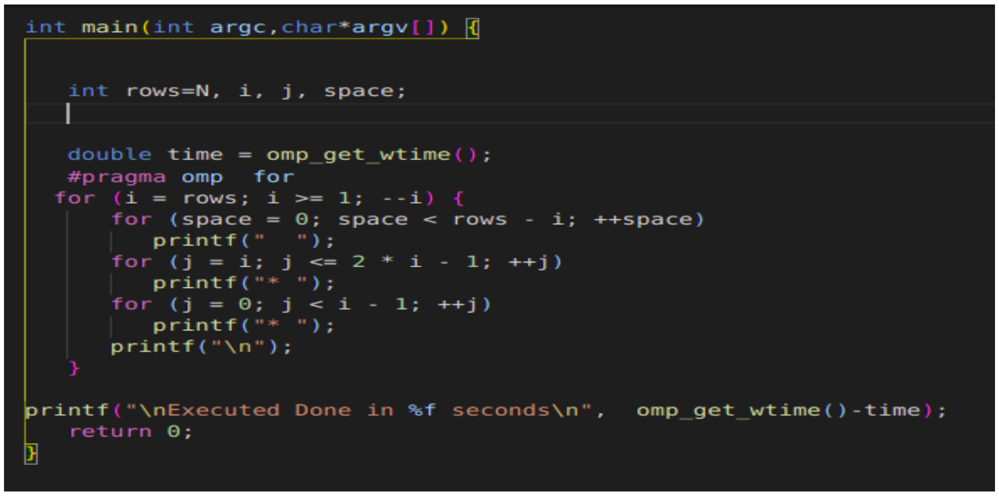
Problem Statement 1

Statement 1:Write an OpenMP program to print inverted pyramid using *.

Screenshot #:



```
prax@prakx-ideapad:~/Desktop/HPC_ESE$ gcc -fopenmp 1.c
prax@prakx-ideapad:~/Desktop/HPC_ESE$ ./a.out
* * * * * * * * * * *
  * * * * * * * * *
    * * * * * * *
      * * * * *
        * * *
          *

Executed Done in 0.000027 seconds
prax@prakx-ideapad:~/Desktop/HPC_ESE$
```



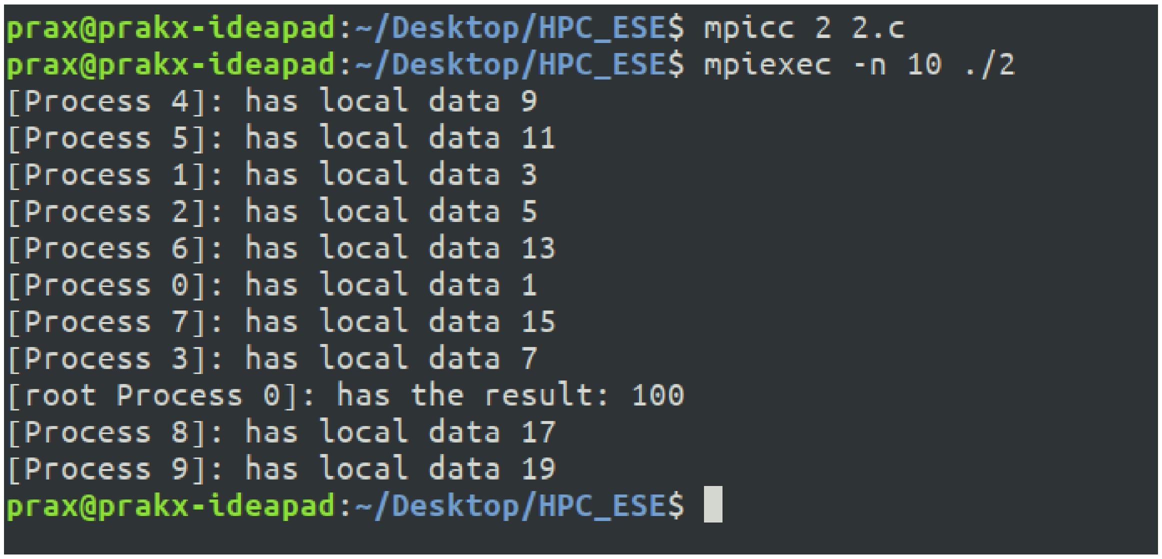
```
int main(int argc,char*argv[]) {

   int rows=N, i, j, space;


   double time = omp_get_wtime();
   #pragma omp  for
  for (i = rows; i >= 1; --i) {
      for (space = 0; space < rows - i; ++space)
         printf("  ");
      for (j = i; j <= 2 * i - 1; ++j)
         printf("* ");
      for (j = 0; j < i - 1; ++j)
         printf("* ");
      printf("\n");
   }

printf("\nExecuted Done in %f seconds\n",  omp_get_wtime()-time);
   return 0;
}
```

Information #:Used openMP to print inverted pyramid using *

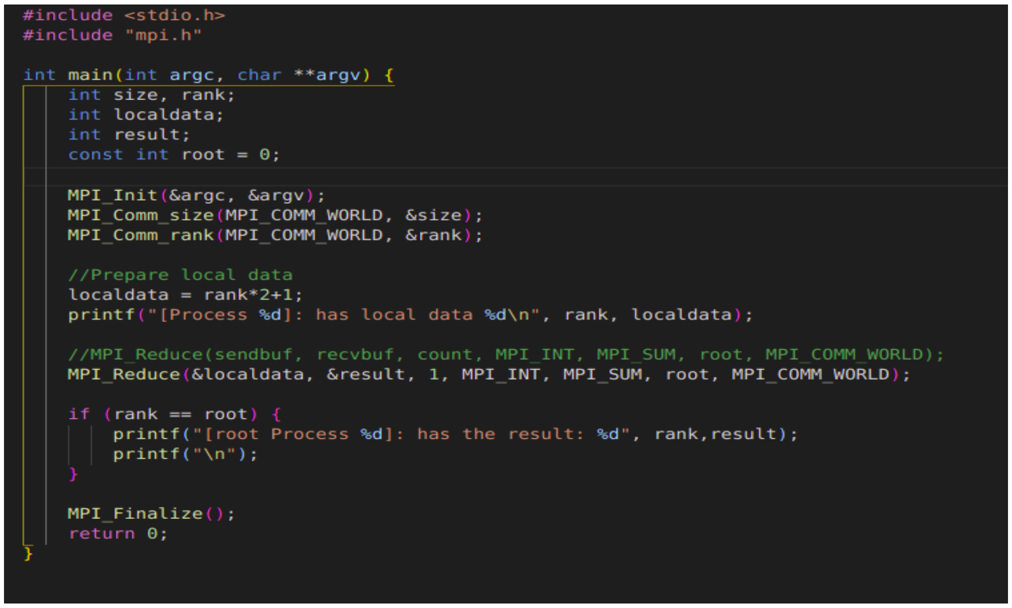
Problem Statement 2

Statement 2: Implement MPI program to reduce the data from n processes to root process.

Screenshot #:



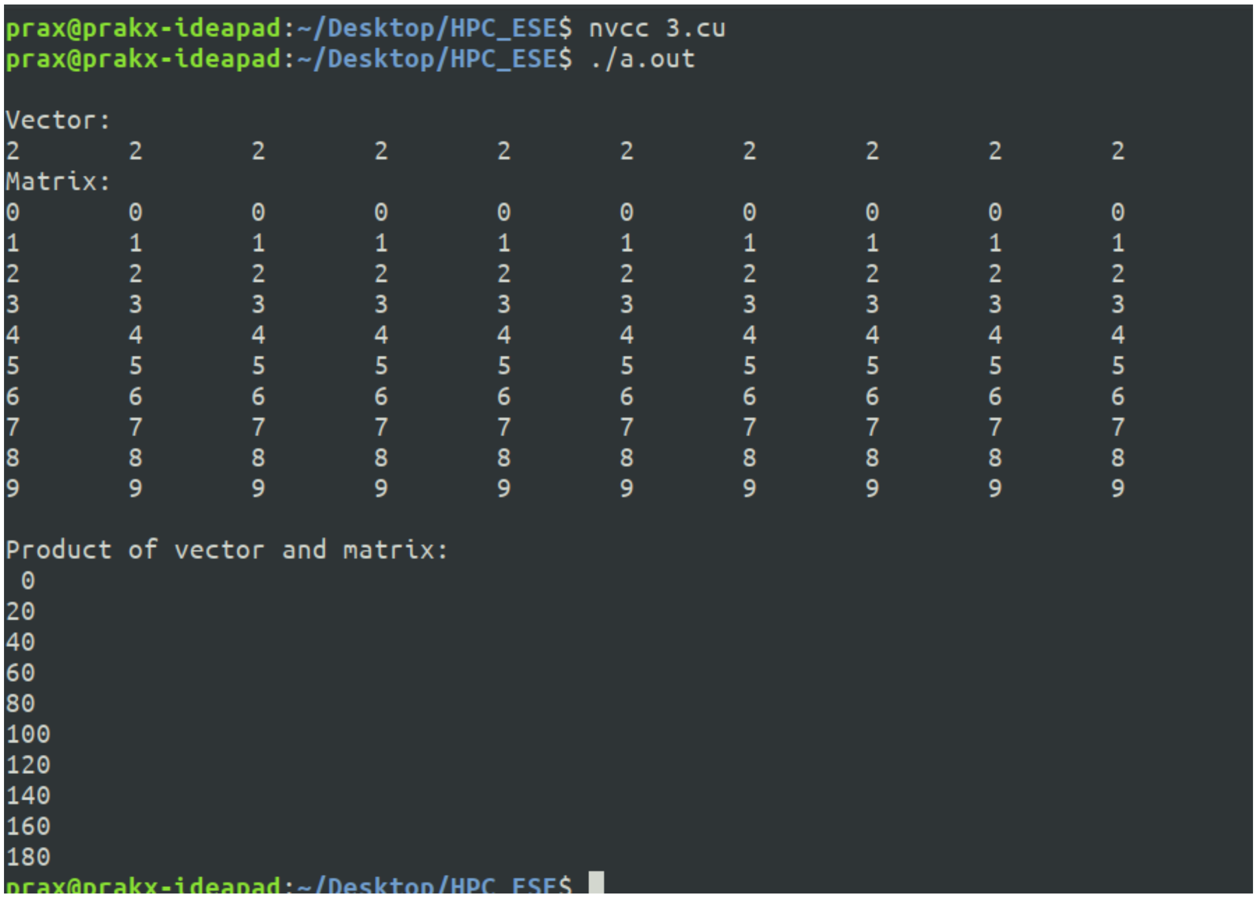
```
prax@prakx-ideapad:~/Desktop/HPC_ESE$ mpicc 2 2.c
prax@prakx-ideapad:~/Desktop/HPC_ESE$ mpiexec -n 10 ./2
[Process 4]: has local data 9
[Process 5]: has local data 11
[Process 1]: has local data 3
[Process 2]: has local data 5
[Process 6]: has local data 13
[Process 0]: has local data 1
[Process 7]: has local data 15
[Process 3]: has local data 7
[root Process 0]: has the result: 100
[Process 8]: has local data 17
[Process 9]: has local data 19
prax@prakx-ideapad:~/Desktop/HPC_ESE$
```



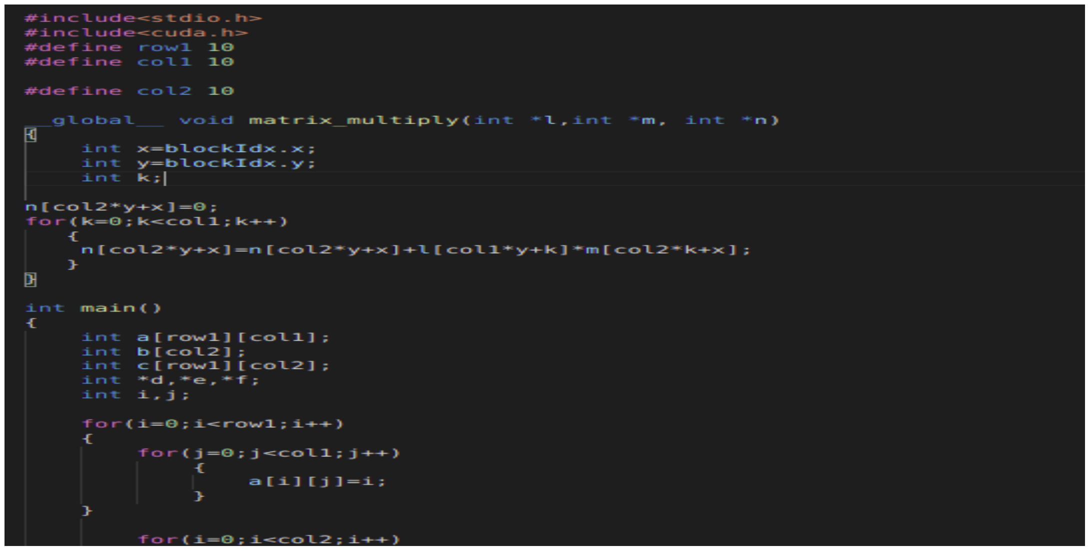
```c
#include <stdio.h>
#include "mpi.h"

int main(int argc, char **argv) {
    int size, rank;
    int localdata;
    int result;
    const int root = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    //Prepare local data
    localdata = rank*2+1;
    printf("[Process %d]: has local data %d\n", rank, localdata);

    //MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD);
    MPI_Reduce(&localdata, &result, 1, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD);

    if (rank == root) {
        printf("[root Process %d]: has the result: %d", rank,result);
        printf("\n");
    }

    MPI_Finalize();
    return 0;
}
```

Information #:Used MPI_reduce function to reduce data from n process to the root process

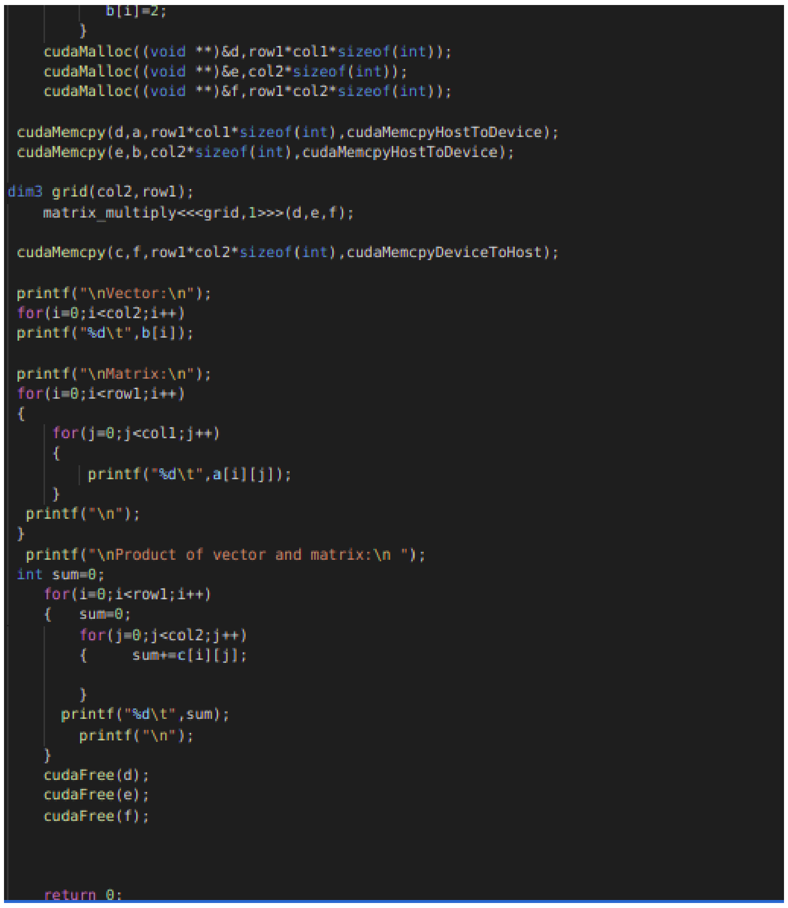
Statement 3: Implement Matrix-Vector multiplication using CUDA.
Screenshot :



```
prax@prakx-ideapad:~/Desktop/HPC_ESE$ nvcc 3.cu
prax@prakx-ideapad:~/Desktop/HPC_ESE$ ./a.out

Vector:
2       2       2       2       2       2       2       2       2       2
Matrix:
0       0       0       0       0       0       0       0       0       0
1       1       1       1       1       1       1       1       1       1
2       2       2       2       2       2       2       2       2       2
3       3       3       3       3       3       3       3       3       3
4       4       4       4       4       4       4       4       4       4
5       5       5       5       5       5       5       5       5       5
6       6       6       6       6       6       6       6       6       6
7       7       7       7       7       7       7       7       7       7
8       8       8       8       8       8       8       8       8       8
9       9       9       9       9       9       9       9       9       9

Product of vector and matrix:
 0
20
40
60
80
100
120
140
160
180
prax@prakx-ideapad:~/Desktop/HPC_ESE$
```



```
#include<stdio.h>
#include<cuda.h>
#define row1 10
#define col1 10

#define col2 10

__global__ void matrix_multiply(int *l,int *m, int *n)
{
    int x=blockIdx.x;
    int y=blockIdx.y;
    int k;

n[col2*y+x]=0;
for(k=0;k<col1;k++)
   {
    n[col2*y+x]=n[col2*y+x]+l[col1*y+k]*m[col2*k+x];
   }
}

int main()
{
    int a[row1][col1];
    int b[col2];
    int c[row1][col2];
    int *d,*e,*f;
    int i,j;

    for(i=0;i<row1;i++)
    {
        for(j=0;j<col1;j++)
            {
                a[i][j]=i;
            }
    }

        for(i=0;i<col2;i++)
```



```
            b[i]=2;
        }
    cudaMalloc((void **)&d,row1*col1*sizeof(int));
    cudaMalloc((void **)&e,col2*sizeof(int));
    cudaMalloc((void **)&f,row1*col2*sizeof(int));

 cudaMemcpy(d,a,row1*col1*sizeof(int),cudaMemcpyHostToDevice);
 cudaMemcpy(e,b,col2*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1);
    matrix_multiply<<<grid,1>>>(d,e,f);

 cudaMemcpy(c,f,row1*col2*sizeof(int),cudaMemcpyDeviceToHost);

 printf("\nVector:\n");
 for(i=0;i<col2;i++)
 printf("%d\t",b[i]);

 printf("\nMatrix:\n");
 for(i=0;i<row1;i++)
 {
     for(j=0;j<col1;j++)
     {
         printf("%d\t",a[i][j]);
     }
  printf("\n");
 }
  printf("\nProduct of vector and matrix:\n ");
 int sum=0;
    for(i=0;i<row1;i++)
    {   sum=0;
        for(j=0;j<col2;j++)
        {     sum+=c[i][j];

        }
      printf("%d\t",sum);
        printf("\n");
    }
    cudaFree(d);
    cudaFree(e);
    cudaFree(f);



    return 0;
```

Technologies Used: OpenMP, MPI, CUDA

GitHub Link:https://github.com/prakx1/HPC_ESE