Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram

High Performance Computing Practice - COM403P

EXPERIMENT 6

Name	Roll Number
N Sree Dhyuti	CED19I027

1. Matrix Addition and Matrix Multiplication

OBJECTIVE:

1. Implement Matrix Addition And Matrix Multiplication using MPI.

Serial Code:

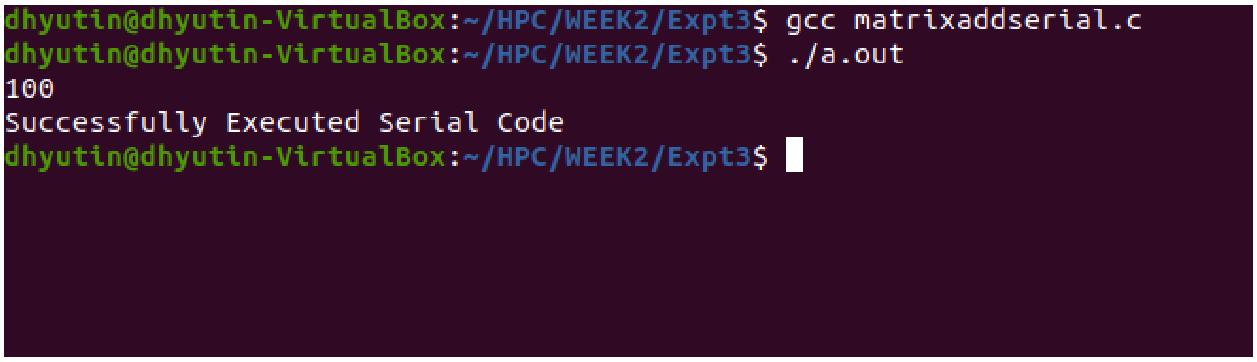
Matrix Addition:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "omp.h"

int main(){

    int n;
    scanf("%d", &n);

    double v1[n][n], v2[n][n], ans[n][n];

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            v1[i][j] = (float)rand()/(float)(RAND_MAX/n);
            v2[i][j] = (float)rand()/(float)(RAND_MAX/n);
            ans[i][j] = 0;
        }
    }

     for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            ans[i][j] = v1[i][j] + v2[i][j];
        }
     }
     printf("Successfully Executed Serial Code\n");

    return 0;
}
```

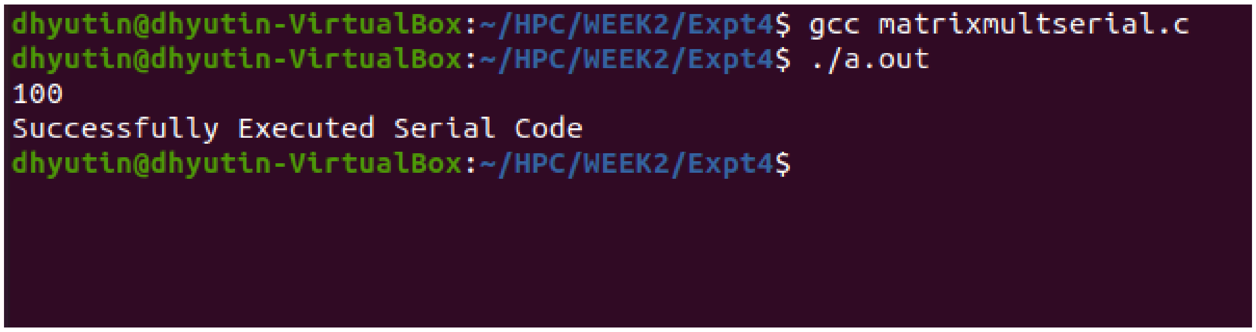
Matrix Multiplication:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "omp.h"

int main(){

    int n;
    scanf("%d", &n);

    double v1[n][n], v2[n][n], ans[n][n];

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            v1[i][j] = (float)rand()/(float)(RAND_MAX/n);
            v2[i][j] = (float)rand()/(float)(RAND_MAX/n);
            ans[i][j] = 0;
        }

    }


     for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            for(int k = 0; k < n; k++){
                ans[i][j] += v1[i][k]*v2[k][j];
            }
        }
     }
    printf("Successfully Executed Serial Code\n");

    return 0;
}
```

Output:



```
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt3$ gcc matrixaddserial.c
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt3$ ./a.out
100
Successfully Executed Serial Code
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt3$
```



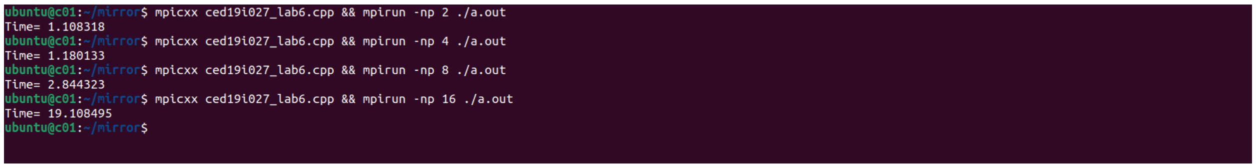
```
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc matrixmultserial.c
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ ./a.out
100
Successfully Executed Serial Code
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$
```

Parallelized Code:

Matrix Addition and Multiplication done as one code:

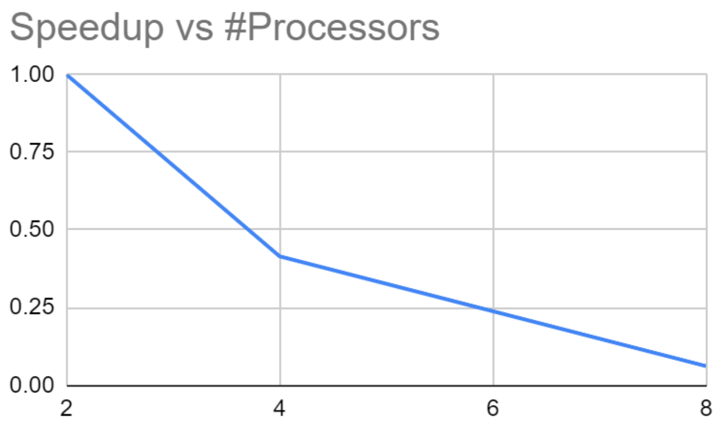
```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "omp.h"

int main(){

    int n;
    scanf("%d", &n);

    double v1[n][n], v2[n][n], ans[n][n];

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            v1[i][j] = (float)rand()/(float)(RAND_MAX/n);
            v2[i][j] = (float)rand()/(float)(RAND_MAX/n);
            ans[i][j] = 0;
        }

    }


     for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            for(int k = 0; k < n; k++){
                ans[i][j] += v1[i][k]*v2[k][j];
            }
            }
        }
    printf("Successfully Executed Serial Code\n");

    return 0;
}
```

Output:

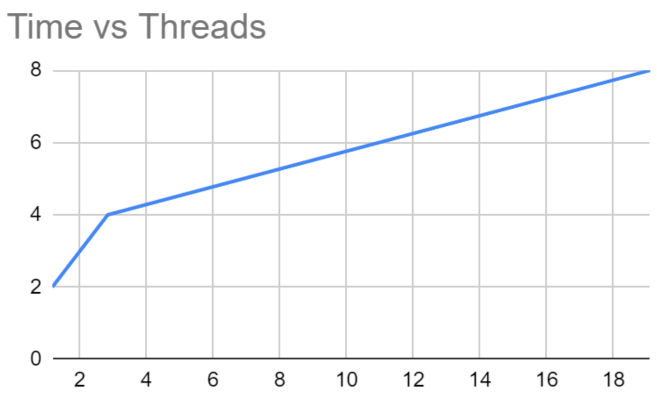


```
ubuntu@c01:~/mirror$ mpicxx ced19i027_lab6.cpp && mpirun -np 2 ./a.out
Time= 1.108318
ubuntu@c01:~/mirror$ mpicxx ced19i027_lab6.cpp && mpirun -np 4 ./a.out
Time= 1.180133
ubuntu@c01:~/mirror$ mpicxx ced19i027_lab6.cpp && mpirun -np 8 ./a.out
Time= 2.844323
ubuntu@c01:~/mirror$ mpicxx ced19i027_lab6.cpp && mpirun -np 16 ./a.out
Time= 19.108495
ubuntu@c01:~/mirror$
```

Speedup V/S Number of Processors:



Execution Time V/S Number of Threads:



Parallelization Factor (f):

#Threads	Execution Time	Speed UP	Efficiency (in %)	f
1	n/a	n/a	n/a	n/a
2	1.180133	1	100	n/a
4	2.844323	0.4149082224	10.37270556	-1.880228754
8	19.108495	0.06175959959	0.7719949949	-17.36207408

Inference:

In both matrix addition and multiplication, it can be observed that the parallelizing factor f is reducing as the number of threads increases. This indicates that parallelizing is only increasing our cost in this case. Therefore, using only one thread for matrix addition and multiplication would be optimal.

THE END
