Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram

High Performance Computing Practice - COM403P

EXPERIMENT 4

Name	Roll Number
N Sree Dhyuti	CED19I027

Matrix Multiplication

OBJECTIVE:

1. Perform matrix multiplication on double precision floating point numbers of stored in n X n matrices

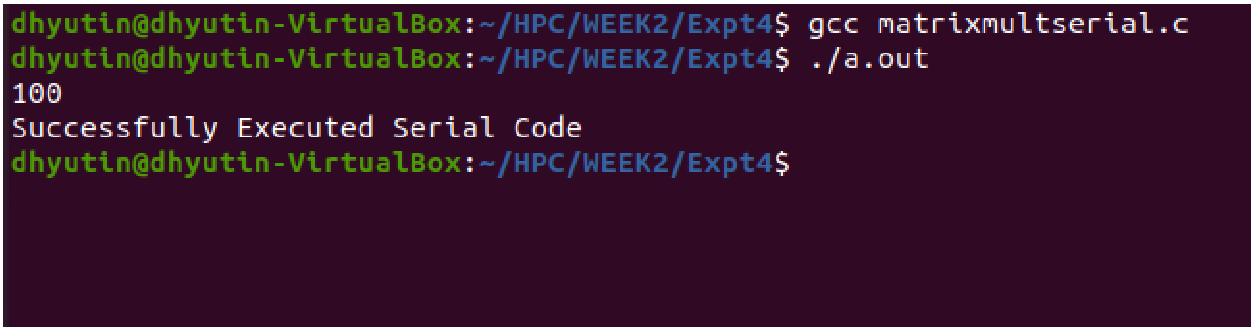
---

Serial Code:

```
#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:

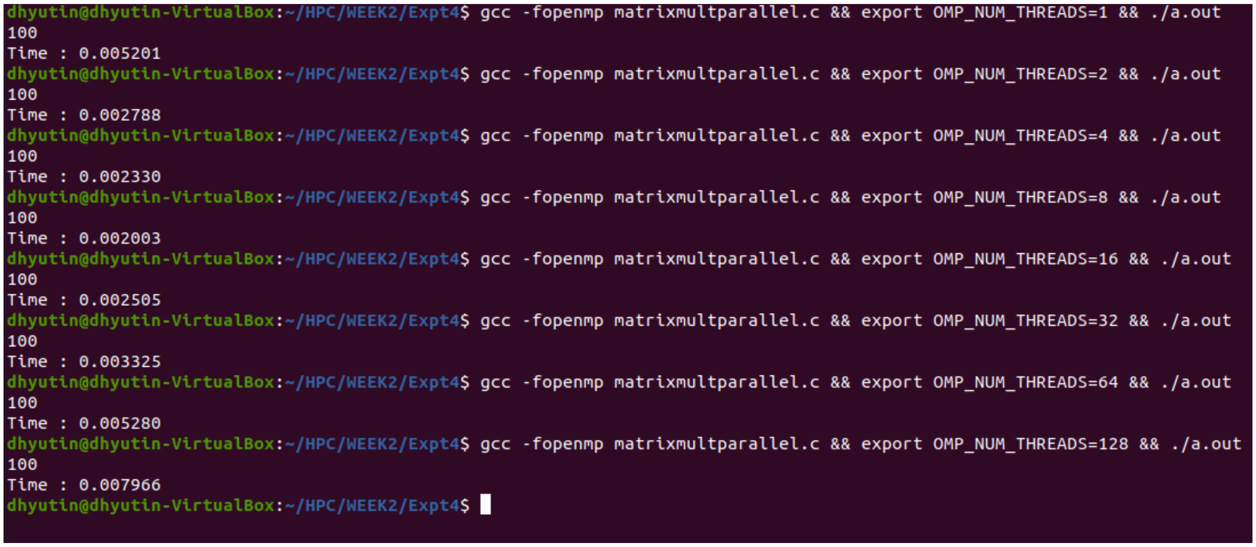


```
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc matrixmultserial.c
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ ./a.out
100
Successfully Executed Serial Code
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$
```

Parallelized Code:

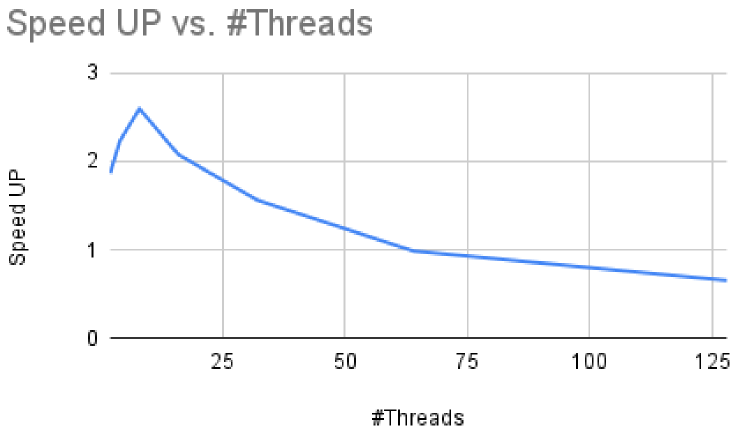
```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "omp.h"

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;
    }

    }
    double wallclock_initial = omp_get_wtime();
    #pragma omp parallel
    {
    int id = omp_get_thread_num();
    #pragma omp for collapse(3)
    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];
        }
    }
    }
    }
    double wallclock_final = omp_get_wtime();
    printf("Time : %lf\n", wallclock_final - wallclock_initial);
    return 0;
}
```

Output:



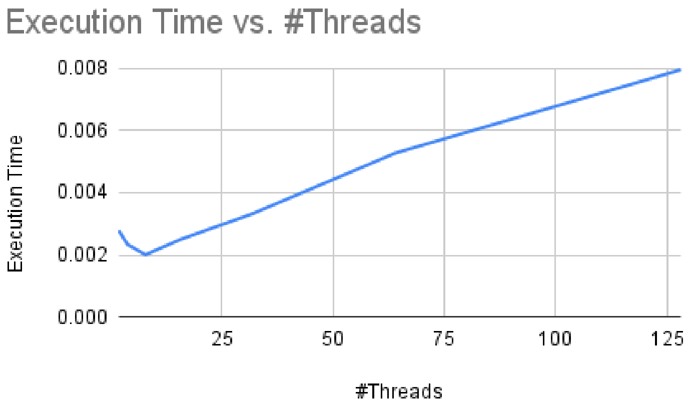
```
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=1 && ./a.out
100
Time : 0.005201
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=2 && ./a.out
100
Time : 0.002788
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=4 && ./a.out
100
Time : 0.002330
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=8 && ./a.out
100
Time : 0.002003
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=16 && ./a.out
100
Time : 0.002505
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=32 && ./a.out
100
Time : 0.003325
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=64 && ./a.out
100
Time : 0.005280
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$ gcc -fopenmp matrixmultparallel.c && export OMP_NUM_THREADS=128 && ./a.out
100
Time : 0.007966
dhyutin@dhyutin-VirtualBox:~/HPC/WEEK2/Expt4$
```

Speedup V/S Number of Processors:



Inference:
It can be inferred from the above graph that Speedup value increases till 8 threads, and then it continues to decrease.

Execution Time V/S Number of Threads:



Inference:
Similar to the previous graph, it can be seen that we attain minimum execution time when we choose 8 threads to execute the program.
This means choosing 8 threads for this program will give us maximum time efficiency.

Parallelization Factor (f):

#Threads	Execution Time	Speed UP	Efficiency (in %)	f
1	0.005201	1	100	n/a
2	0.002788	1.865494978	93.27474892	0.9278984811
4	0.00233	2.232188841	55.80472103	0.7360123053
8	0.002003	2.596605092	32.45756365	0.702722004
16	0.002505	2.076247505	12.97654691	0.5529193104
32	0.003325	1.564210526	4.888157895	0.3723353449
64	0.00528	0.9850378788	1.539121686	-0.01543048803
128	0.007966	0.6528998243	0.5100779877	-0.5358145844

Inference:

It can be noticed from the pattern of ‘f’ values that taking 2 threads will give utmost parallelization. Choosing 2 threads to execute this program will give the most efficient parallelization.

THE END