HPC LAB 3 : MATRIX ADDITION

Name: Mridul Harish
Roll No: CED18I034
Programming Environment: OpenMP
Problem: Matrix Addition
Date: 19th August 2021

Hardware Configuration:

CPU NAME : Intel Core i5-8250U @ 8x 3.4GHz
Number of Sockets : 1
Cores per Socket : 4
Threads per core : 2
L1 Cache size : 32KB
L2 Cache size : 256KB
L3 Cache size(Shared): 6MB
RAM : 8 GB

Serial Code:

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

int main(int argc, char* argv[])
{
	float start_time; float end_time; float exec_time;
	double a[500][500]; double b[500][500]; double c[500][500];
	int row = 500, column = 500;
	start_time = omp_get_wtime();

	// Addition
	for(int i = 0; i < row; i = i+1)
	{
		for(int j = 0; j < column; j = j+1)
		{
			a[i][j] = i+j+12.549;
			b[i][j] = i+j+97.949;
			c[i][j] = a[i][j] + b[i][j];
		}
```

```
        }

    end_time = omp_get_wtime();
    exec_time = end_time - start_time;
    printf("%f\n", exec_time);
}
```

Parallel Code:

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

int main(int argc, char* argv[])
{
        float start_time; float end_time; float exec_time;
         double a[500][500]; double b[500][500]; double c[500][500];
        int row = 500, column = 500;
        start_time = omp_get_wtime();

        // Addition
        #pragma omp parallel for
        for(int i = 0; i < row; i = i+1)
        {
                for(int k = 0; k < 100000; k = k+1);
                #pragma omp parallel for
                for(int j = 0; j < column; j = j+1)
                {
                        a[i][j] = i+j+12.549;
                        b[i][j] = i+j+97.949;
                        c[i][j] = a[i][j] + b[i][j];
                }
        }

    end_time = omp_get_wtime();
    exec_time = end_time - start_time;
    printf("%f\n", exec_time);
}
```

Compilation and Execution:

For enabling OpenMP environment use -fopenmp flag while compiling using gcc;

```
gcc -fopenmp openMP_matrixaddition.c
```

For execution use;

export OMP_NUM_THREADS = x (Where x is the number of threads we are deploying)
./a.out

OBSERVATION TABLE

No of Threads	Execution Time	Speedup	Parallel Fraction
1	0.232422		
2	0.121094	1.919351908	0.9579816024
4	0.091797	2.531912808	0.8067222552
6	0.048828	4.760014746	0.9478999406
8	0.042969	5.409062347	0.9315715134
10	0.048828	4.760014746	0.8776851302
12	0.054688	4.249963429	0.8342223902
16	0.050781	4.576948071	0.8336147181
20	0.058594	3.966651876	0.7872612838
24	0.074219	3.131570083	0.7102657722
32	0.046875	4.958336	0.8240716761
64	0.068359	3.40002048	0.7170886345
128	0.0695	3.344201439	0.7064944385

Speed up can be found using the following formula;

S(n)=T(1)/T(n) where, S(n) = Speedup for thread count ‘n’
T(1) = Execution Time for Thread count ‘1’ (serial code)
T(n) = Execution Time for Thread count ‘n’ (serial code)

Parallelization Fraction can be found using the following formula;

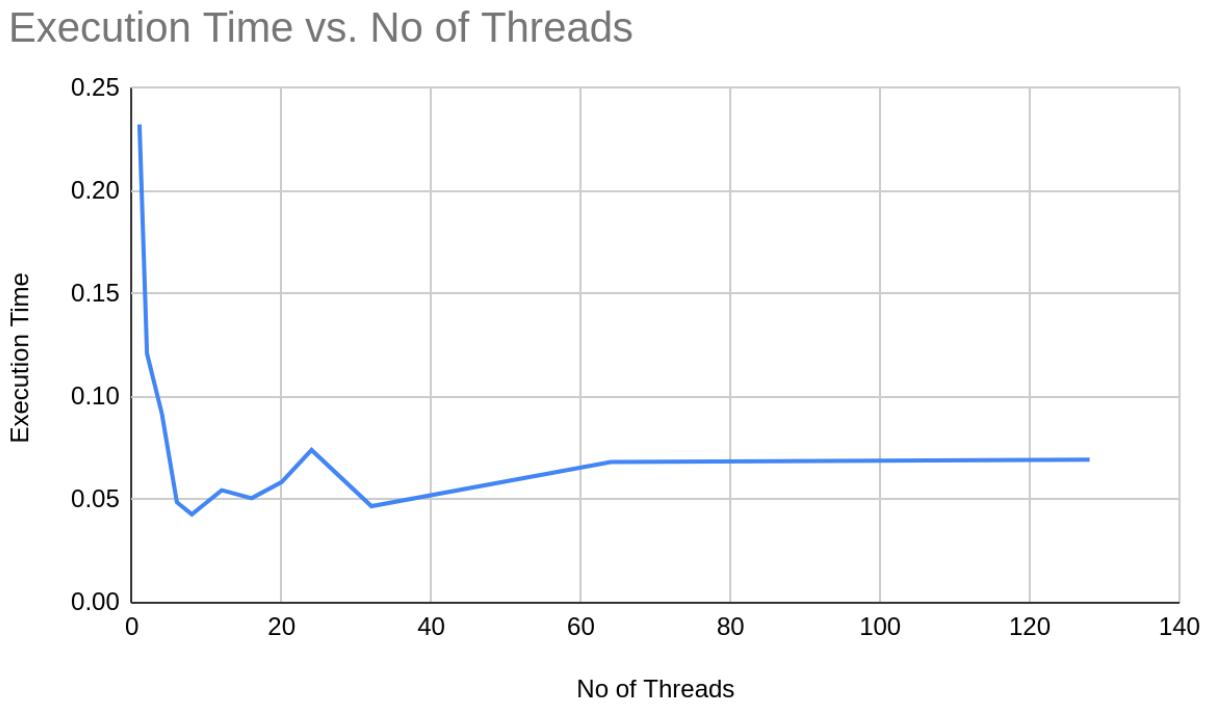
S(n)=1/((1 - p) + p/n where, S(n) = Speedup for thread count ‘n’
n = Number of threads
p = Parallelization fraction

Assumption:

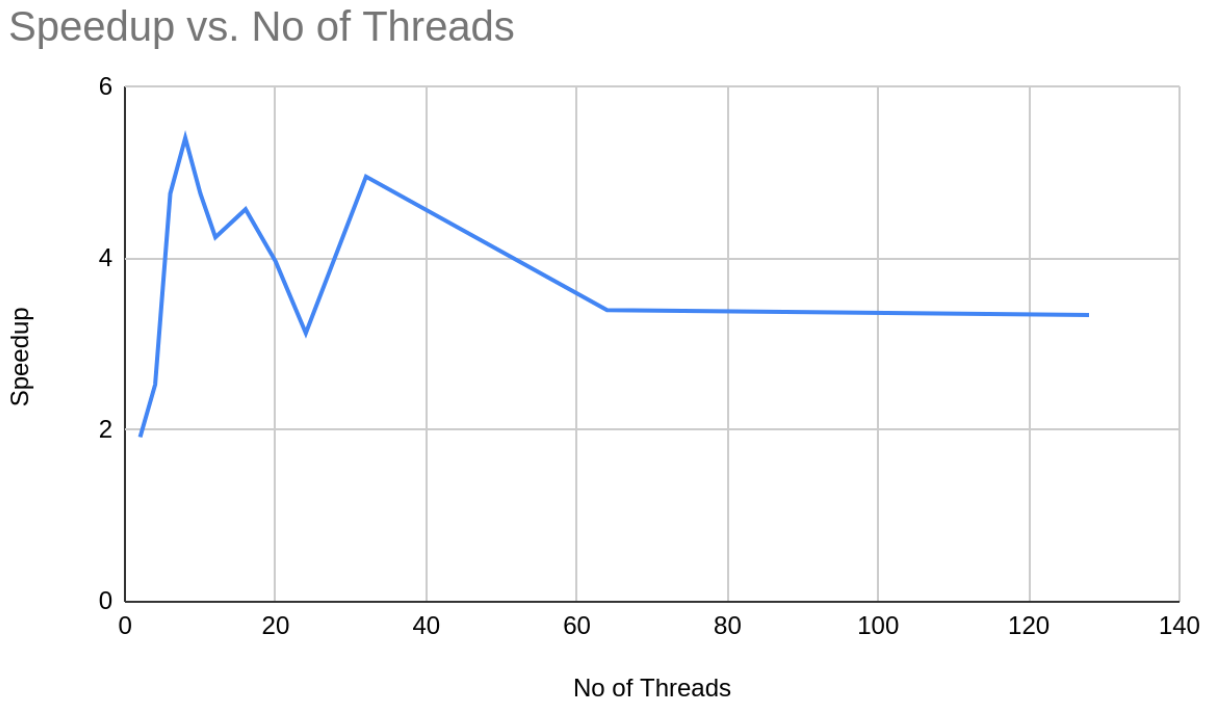
Following extra for loop is added to increase the number of operations in the parallel region to visualize the effect of multithreading in matrix addition.

for(int k = 0; k < m; k = k+1)

Number of Threads(x axis) vs Execution Time(y axis) :



Number of Threads(x axis) vs Speedup(y axis):



Inference:

(Note: Execution time, graph and inference will be based on hardware configuration)

• At thread count 8 maximum speedup is observed.

• If thread count 8, execution time increases gradually and speedup decreases. There is a spike in execution time at thread count 24 and sharp decrease in speedup at that point but after that execution time increases, speedup decreases and tapers out in the end.