HPC LAB 4 : MATRIX MULTIPLICATION

Name: Mridul Harish
Roll No: CED18I034
Programming Environment: OpenMP
Problem: Matrix Multiplication
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;
   start_time = omp_get_wtime();

   double a[500][500]; double b[500][500]; double c[500][500];
         int row = 500, column = 500;

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

```
                        for(int k = 0; k < column; k = k+1)
                        {
                                c[i][j] += a[i][k]*b[k][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;
    start_time = omp_get_wtime();

   double a[500][500]; double b[500][500]; double c[500][500];
          int row = 500, column = 500;

    //multiplication
    #pragma omp parallel for
    for(int i = 0; i < row; i = i+1)
          {
                #pragma omp parallel for
                for(int j = 0; j < column; j = j+1)
                {
                        c[i][j] = 0;
                       a[i][j] = i+j+12.549;
                        b[i][j] = i+j+97.949;
                        for(int k = 0; k < column; k = k+1)
                        {
                                c[i][j] += a[i][k]*b[k][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_matrixmultiplication.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	0.632812		
2	0.326172	1.940117484	0.9691345929
4	0.273438	2.314279654	0.7572001374
6	0.205078	3.085713728	0.8111110409
8	0.199219	3.176464092	0.7830680473
10	0.207031	3.056605049	0.747599603
12	0.201172	3.145626628	0.7441072546
16	0.191406	3.306124155	0.7440330883
20	0.197266	3.207912159	0.7244955432
24	0.203125	3.115382154	0.7085343569
32	0.205078	3.085713728	0.6977299276
64	0.214766	2.946518536	0.6711023982
128	0.223125	2.836132213	0.6525048227

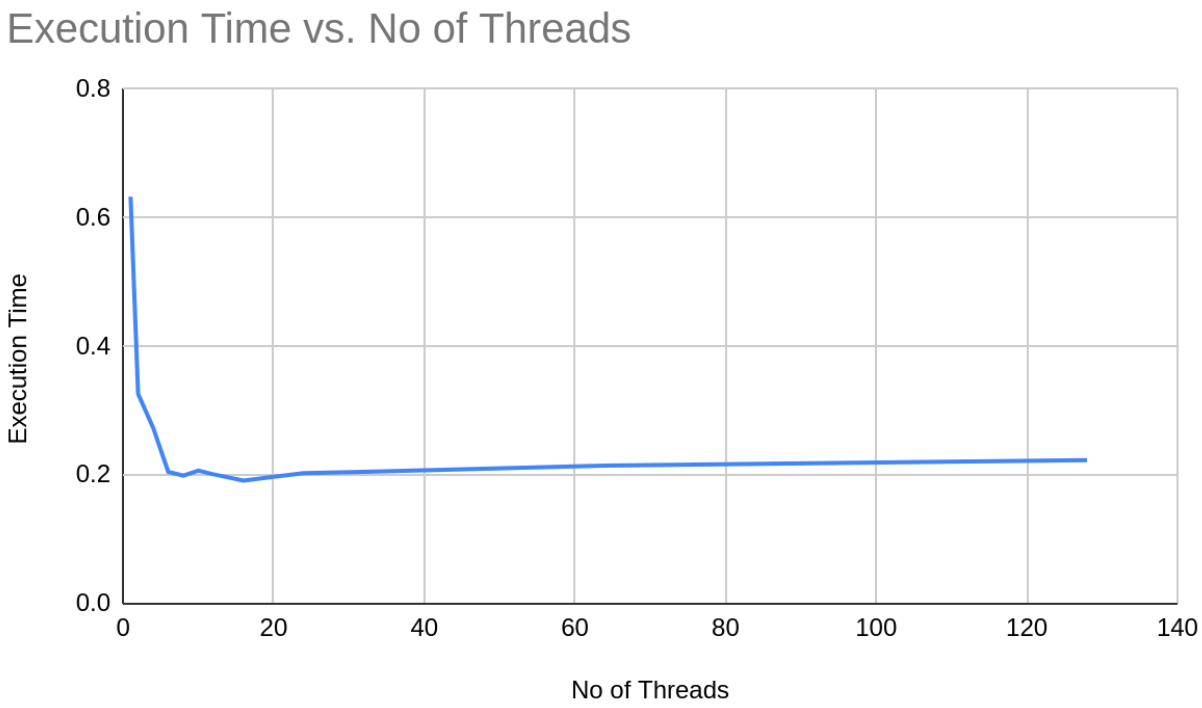
Speed up can be found using the following formula;

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)

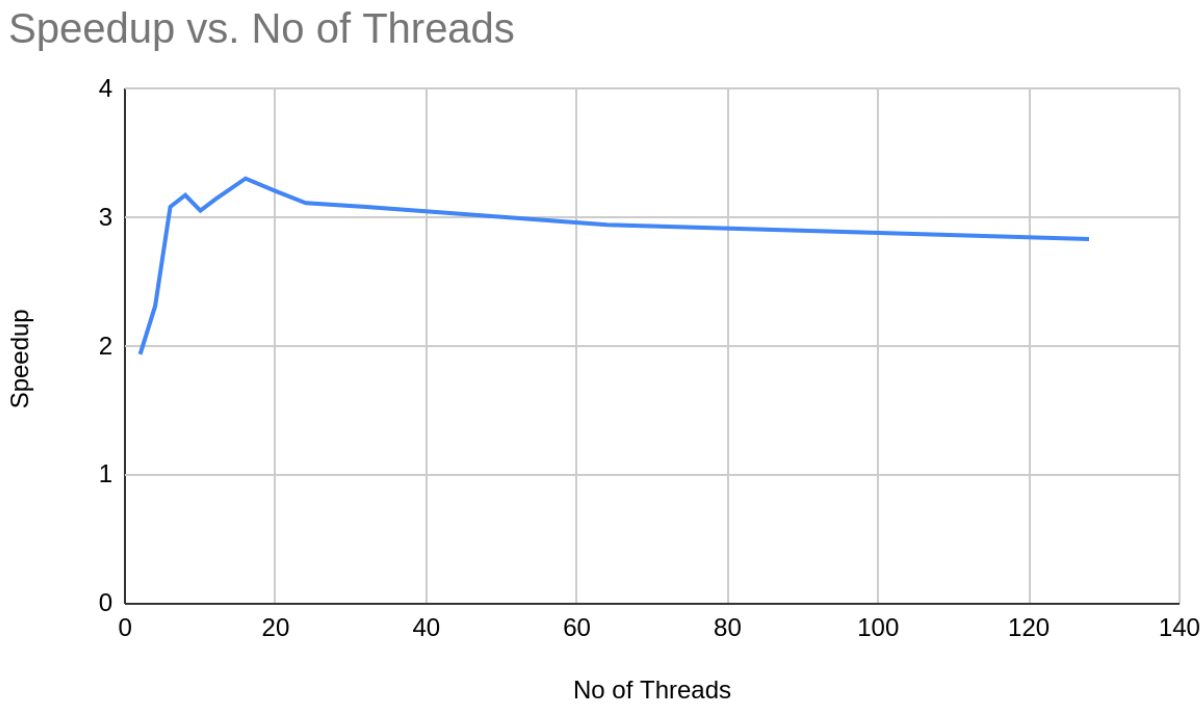
Parallelization Fraction can be found using the following formula;

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

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 16 maximum speedup is observed.
- If thread count is more than 16 the execution time continues increasing and the speedup decreases and tapers out in the end.