HPC LAB 1 : Vector Addition

Name: Mridul Harish
Roll No: CED18I034
Programming Environment: OpenMP
Problem: Vector 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>

#define n 10000

int main(int argc, char argv[])
{
    int i = 0;
    double  a[n]; double b[n]; double c[n];
    float start_time; float end_time; float exec_time;

    start_time = omp_get_wtime();
                    for(i = 0; i < n; i = i+1)
                    {
                              for(int j = 0; j < n; j = j+1);
                              a[i] = i * 58.473;
                              b[i] = i * 218.893;
                              c[i] = a[i] + b[i];
                              printf("a[%d] = %lf, b[%d] = %lf, c[%d] = %lf\n", i, a[i], i, b[i], i, c[i]);
                    }
```

```
    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>

#define n 10000

int main(int argc, char argv[])
{
    int omp_rank;
    int i = 0;
    double  a[n]; double b[n]; double c[n];
    float start_time; float end_time; float exec_time;

    start_time = omp_get_wtime();
    #pragma omp parallel private(i) shared (a, b, c)
    {
            #pragma omp for
                    for(i = 0; i < n; i = i+1)
                    {
                            omp_rank = omp_get_thread_num();
                            for(int j = 0; j < n; j = j+1);
                            a[i] = i * 58.473;
                            b[i] = i * 218.893;
                            c[i] = a[i] + b[i];

                            printf("a[%d] = %lf, b[%d] = %lf, c[%d] = %lf, Thread ID = %d \n", i, a[i], i,
b[i], i, c[i], omp_rank);
                    }
    }
    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_vectoraddition.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	0.179688		
2	0.123535	1.454551342	0.6250055652
4	0.07959	2.257670562	0.7427541071
6	0.069824	2.573441797	0.7336984106
8	0.062012	2.897632716	0.7484465136
10	0.050498	3.558319141	0.7988538157
12	0.049316	3.64360451	0.7915052758
16	0.10273	1.749128784	0.4568392621
20	0.061523	2.920663817	0.6922232454
24	0.050293	3.572823256	0.7514184006
32	0.047852	3.755078158	0.7573615055
64	0.061035	2.944015729	0.670809297
128	0.072246	2.487168078	0.6026250318

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;

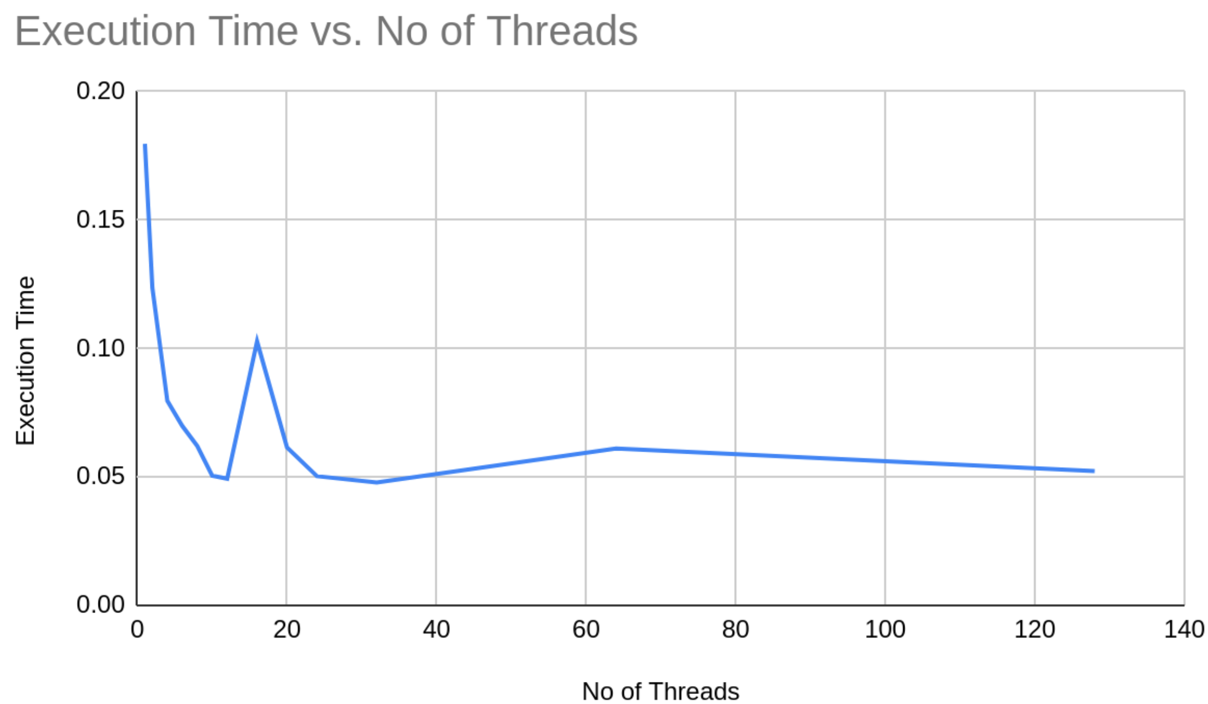
S(n)=1/((1 - p) + p/n where, S(n) = Speedup for thread count ‘n’
n = Number of threads
p = Parallelization fraction

Assumption:

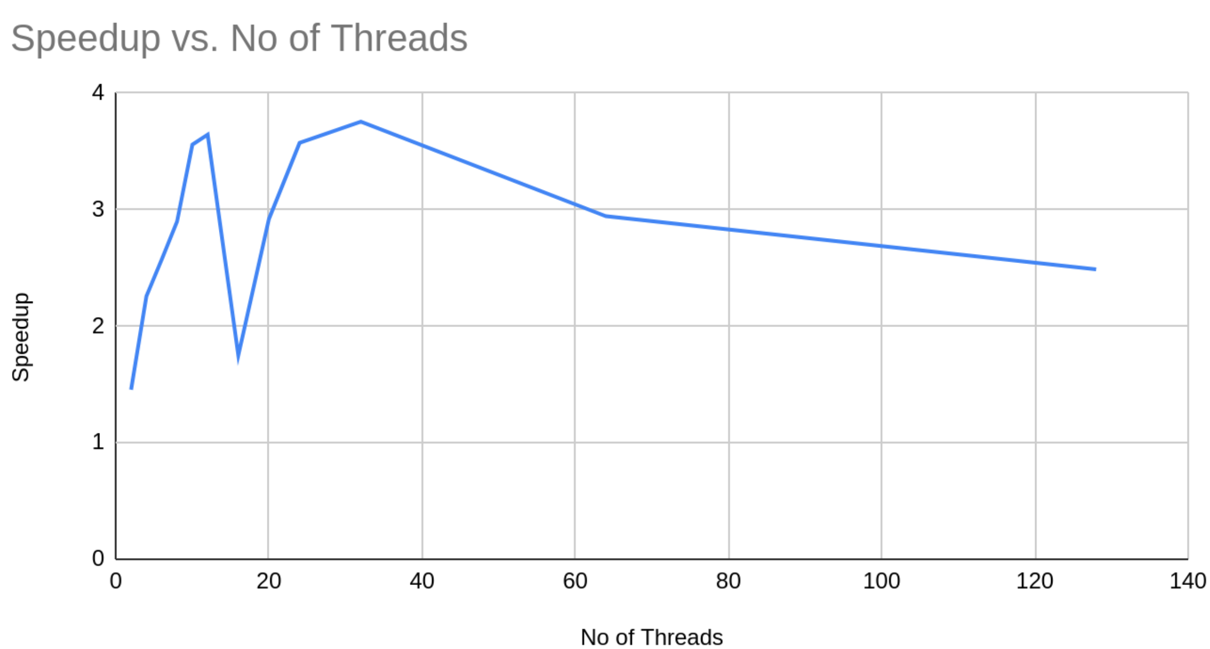
Following extra for loop is added to increase the number of operations in the parallel region to visualize the effect of multi-threading in vector addition.

```
for(int j=0;j<m;j++)
```

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 32 maximum speedup is observed.
- If thread count is more than 32 then the execution time continues increasing and the speedup decreases and tapers out in the end.