Report for HPC LAB

Name: Bhaskar R
Roll No: CED18I009
Programming Environment: OpenMP
Problem: Vector Multiplication
Date: 19th August 2021

Hardware Configuration:

CPU NAME : Intel core i5 – 8250U @ 1.60 Ghz
Number of Sockets : 1
Cores per Socket : 4
Threads per core : 8
L1 Cache size : 64KB (Per Core)
L2 Cache size : 256KB (Per Core)
L3 Cache size : 6MB (Shared)
RAM : 8 GB

Serial Code:

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

#define n 100000
#define m 100000

int main()
{
	double a[n],b[n], c[n];
	float startTime, endTime,execTime;
	int i,k;
	int omp_rank;
	float rtime;



	startTime = omp_get_wtime();
	for(i=0;i<n;i++)
	{

		a[i] = i * 10.236 ; // Use Random function and assign a[i]
		b[i] = i * 152.123; // Use Random function and assign b[i]
		for(int j=0;j<m;j++)
			c[i] = a[i] * b[i];
		//printf("The value of a[%d] = %lf and b[%d] = %lf and result c[%d] = %lf done by
worker Thread ID = %d\n", i, a[i], i, b[i], i, c[i], omp_rank);
	}
	endTime = omp_get_wtime();

	execTime = endTime - startTime;
	rtime=execTime;
```

```
        printf("\n rtime=%f\n",rtime);
        return(0);
}
```

Parallel Code:

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

#define n 100000
#define m 100000

int main()
{
        double a[n],b[n], c[n];
        float startTime, endTime,execTime;
        int i,k;
        int omp_rank;
        float rtime[20];
        int thread[]={1,2,4,6,8,10,12,16,20,32,64,128,150};
        int thread_arr_size=13;
        for(k=0;k<thread_arr_size;k++)
        {
                omp_set_num_threads(thread[k]);

                startTime = omp_get_wtime();

                #pragma omp parallel private (i) shared (a,b,c)
                {
                        #pragma omp for
                        for(i=0;i<n;i++)
                        {

                                omp_rank = omp_get_thread_num();
                                        a[i] = i * 10.236 ; // Use Random function and assign a[i]
                                        b[i] = i * 152.123; // Use Random function and assign b[i]
                                for(int j=0;j<m;j++)
                                        c[i] = a[i] * b[i];
        // printf("The value of a[%d] = %lf and b[%d] = %lf and result c[%d] =  %lf done by worker
Thread ID = %d\n", i, a[i], i, b[i], i, c[i], omp_rank);
                        }

                }
                endTime = omp_get_wtime();
                execTime = endTime - startTime;
                rtime[k]=execTime;
        }
        for (k=0;k<thread_arr_size;k++)
                printf("\nThread=%d\t rtime=%f\n",thread[k],rtime[k]);
        return(0);
}
```

Compilation and Execution:
For enabling OpenMP environment use -fopenmp flag while compiling using g++.

```
g++ -fopenmp vectormul.cpp
```

For execution use

```
./a.out
```

Observations:

Number of Threads	Execution Time	Speed-up	Parallelization Fraction
1	28.968750	1	
2	14.753125	1.96	97.9
4	9.855469	2.94	87.9
6	9.886719	2.93	79.0
8	9.695312	2.98	75.9
10	8.820312	3.28	77.2
12	9.738281	2.97	72.3
16	9.496094	3.05	71.6
20	9.460938	3.06	70.8
32	10.593750	2.73	65.4
64	10.691406	2.70	63.9
128	8.894531	3.25	69.7
150	9.632812	3.00	67.1

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

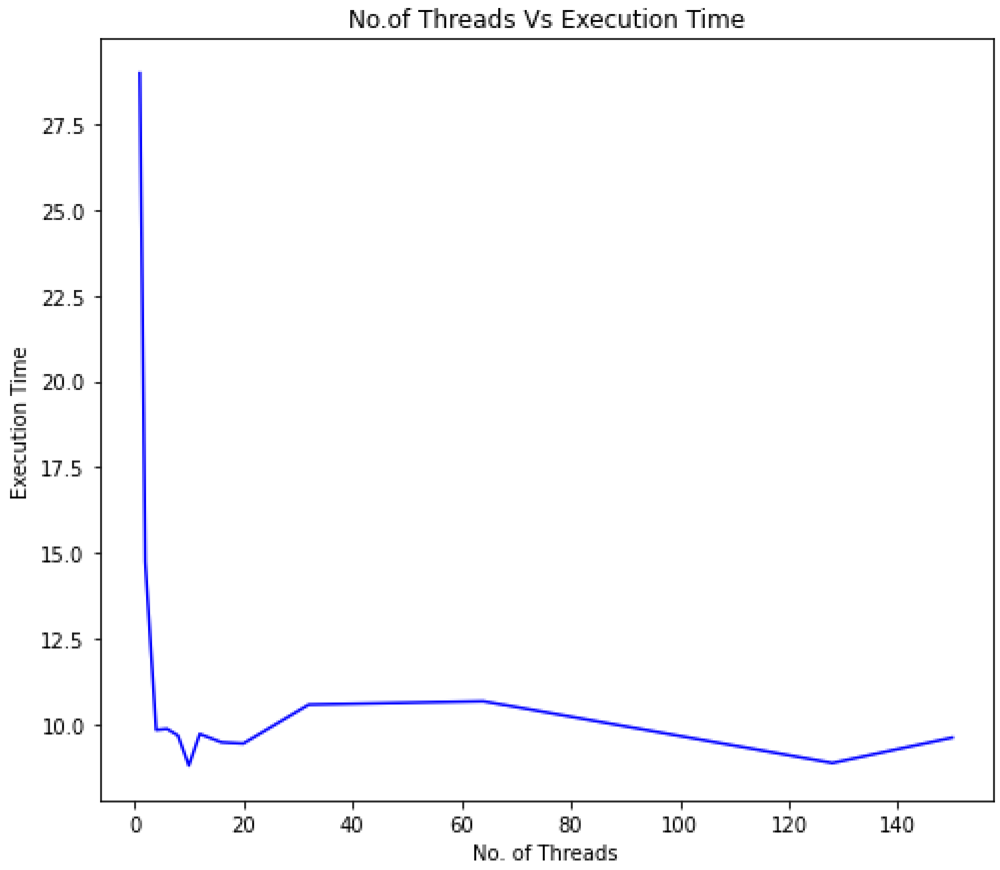
p = Parallelization fraction

Assumption:

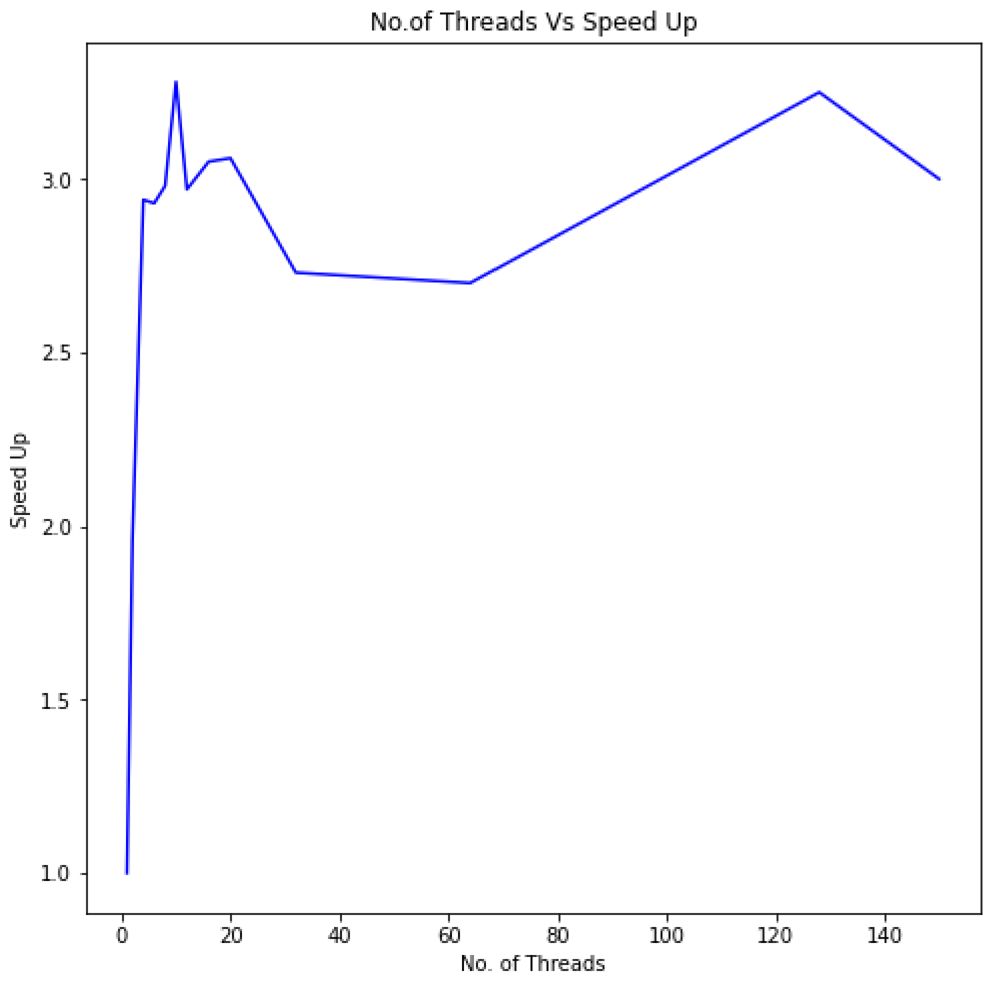
Following extra for loop is added to increase the number of operations in the parallel region to visualize the effect of multi-threading in vector addition.

```
for(int j=0;j<m;j++)
        c[i] = a[i] * b[i];
```

Number of Threads vs Execution Time:



Number of Threads vs Speed Up:



Inference:
(Note: Execution time, graph and inference will be based on hardware configuration**)**
• At thread count 10 maximum speedup is observed as the maximum number of parallel threads supported by the hardware is 8.
• If the thread count is more than 2 then the execution time increases slightly and tapers out after 20 threads.