# Report for HPC LAB

**Name:** Bhaskar R
**Roll No:** CED18I009
**Programming Environment:** OpenMP
**Problem:** Sum of N Numbers using reduction and critical section
**Date:** 26th August 2021

**Hardware Configuration:**

CPU NAME : Intel core i5 – 8250U @ 1.60 GHz
Number of Sockets : 1
Cores per Socket : 4
Threads per core : 2
L1 Cache size : 32KB (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 <omp.h>
#include <stdlib.h>
#define n 10000
int main()
{
	double a[n], rtime;
	float startTime, endTime, execTime;
	int i, k, omp_rank;
	double sum;
	int thread[] = {1, 2, 4, 6, 8, 12, 16, 20, 32, 64};
	int thread_arr_size = 10;
	sum = 0.0;
	omp_set_num_threads(thread[k]);
	startTime = omp_get_wtime();
	for (i = 0; i < n; i++)
	{
		omp_rank = omp_get_thread_num();
		a[i] = (float)i * 1.76;
		for (int j = 0; j < 100010; j++)
			a[i] = a[i] + 2;
		sum = sum + a[i];
	}
	endTime = omp_get_wtime();
	execTime = endTime - startTime;
	rtime = execTime;
	printf("rtime = %f\n", rtime);
	return 0;
}
```

**Parallel Code:**

```
#include <stdio.h>
#include<time.h>
#include <omp.h>
#include<stdlib.h>
#define n 10000
int main()
{
	double a[n],rtime[n],sumarr[n];
	float startTime, endTime,execTime;
	int i,k,omp_rank;
	double sum;
	int thread[]={1,2,4,6,8,12,16,20,32,64};
	int thread_arr_size=10;
	for(k=0;k<thread_arr_size;k++)
	{
		sum=0.0;
		omp_set_num_threads(thread[k]);
		startTime = omp_get_wtime();
		#pragma omp parallel private(i)
		{
			#pragma omp for reduction (+:sum)
			for(i=0;i<n;i++)
			{
			omp_rank = omp_get_thread_num();
			a[i] = (float)i*1.67 ;
			for(int j=0;j<100010;j++)
				a[i]=a[i]+2;
			sum=sum+a[i];
		}
	}
	endTime = omp_get_wtime();
	execTime= endTime - startTime;
	rtime[k]=execTime;
	sumarr[k]=sum;
	}
	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 sum.cpp**

For execution use

./a.out

**Observations:**

| Number of Threads | Execution Time | Speed-up | Parallelization Fraction |
|---|---|---|---|
| 1 | 3.335938 | 1 | |
| 2 | 1.544922 | 2.15 | 106.9 |
| 4 | 1.070312 | 3.11 | 90.4 |
| 6 | 0.712891 | 4.67 | 94.3 |
| 8 | 0.589844 | 5.65 | 94.0 |
| 12 | 0.615234 | 5.42 | 88.9 |
| 16 | 0.599609 | 5.56 | 87.4 |
| 20 | 0.583984 | 5.71 | 86.8 |
| 32 | 0.539062 | 6.18 | 86.5 |
| 64 | 0.609375 | 5.47 | 83.0 |

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

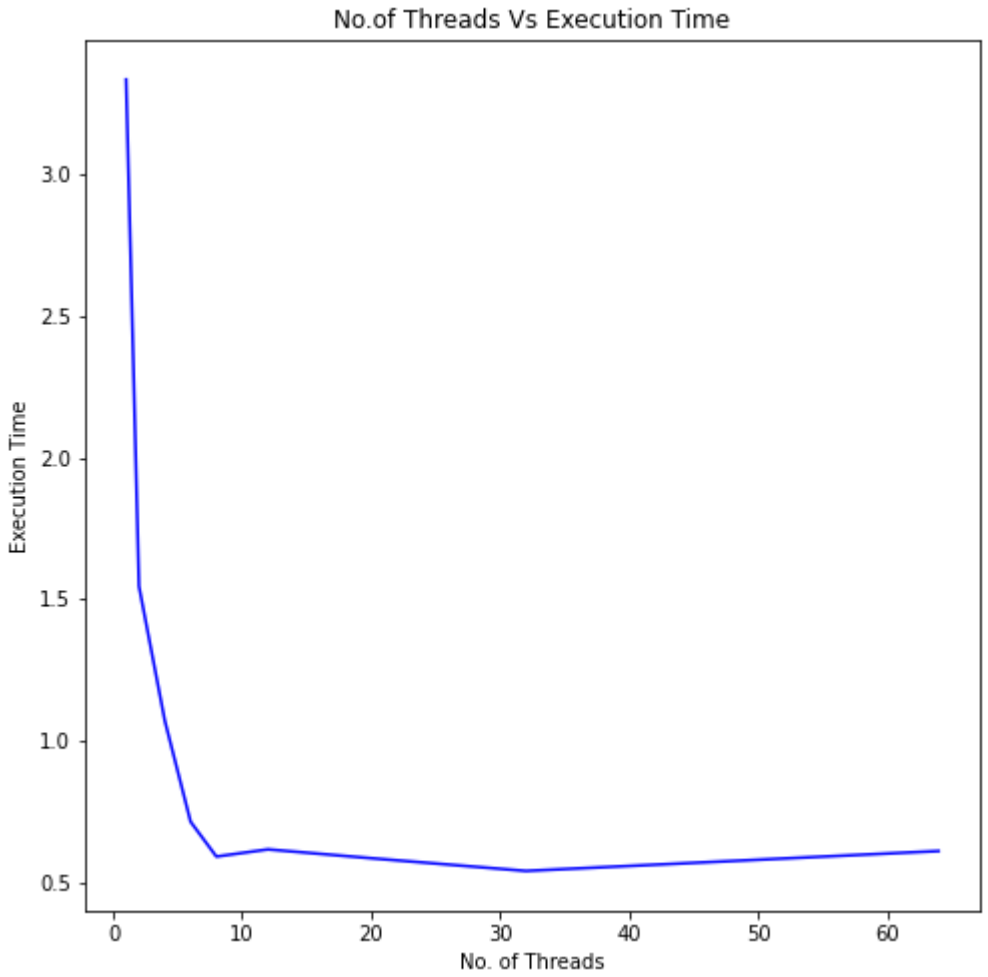
p = Parallelization fraction

**Assumption:**

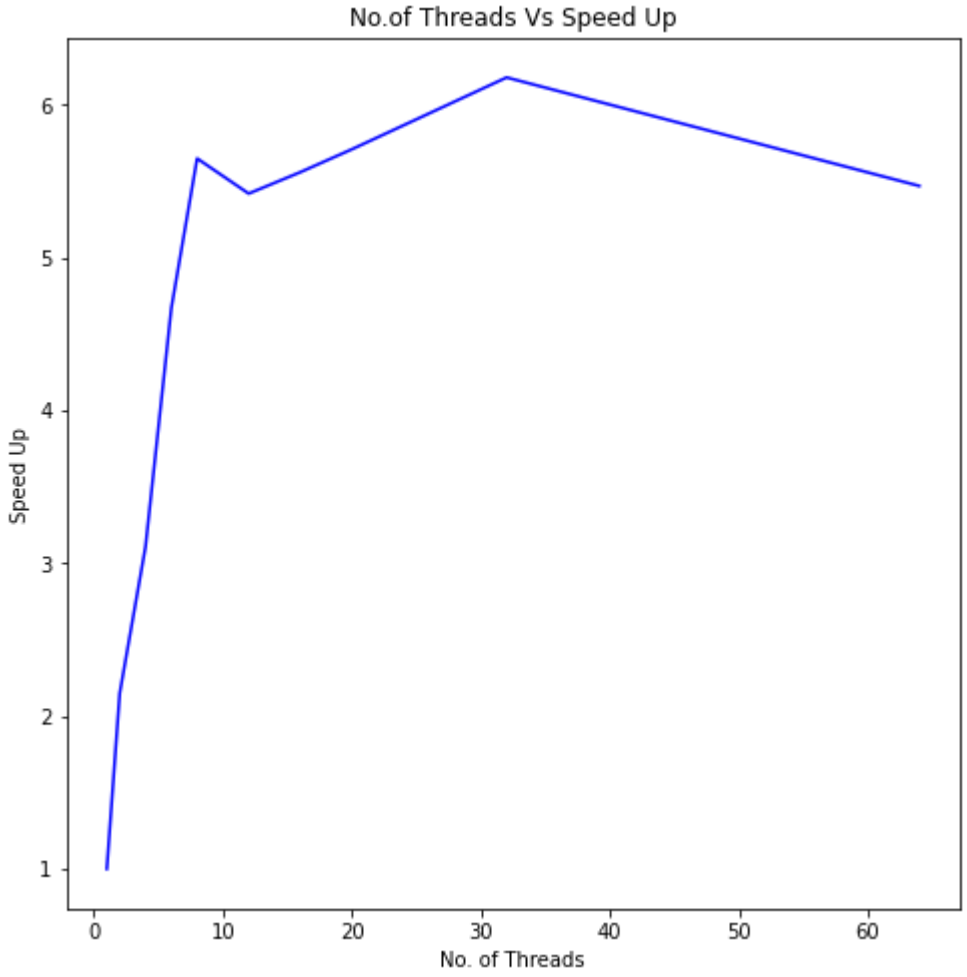
Following extra for loop is added to increase the number of operations in the parallel region to visualize the effect of multi-threading in sum.

```
for (int j = 0; j < 100010; j++)
        a[i] = a[i] + 2;
sum = sum + a[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 32 maximum speedup is observed as the maximum number of parallel threads supported by the hardware is 8.
• If the thread count is more than 8 then the execution time increases slightly and tapers out after 12 threads.

**ii) Parallel Code:(Critical Section)**

```
#include <stdio.h>
#include <time.h>
#include <omp.h>
#include <stdlib.h>
#define N 100000
#define DELAY 100000
int threads[] = {1, 2, 4, 6, 8, 10, 12, 16, 20, 32, 64};
double rtime[11];
int main()
{
	double fsum, psum, a[N];
	double start, end;
	int i, k, l;
	for (k = 0; k < 11; k++)
	{
		omp_set_num_threads(threads[k]);
		fsum = 0.0;
		start = omp_get_wtime();
#pragma omp parallel
		{
			double psum = 0.0;
#pragma omp for
			for (i = 0; i < N; i++)
			{
				a[i] = (float)(i % 50) * 1.76;
				for (int kk = 0; kk < DELAY; kk++)
					;
				psum = psum + a[i];
			}
#pragma omp critical
			fsum = fsum + psum;
		}
		end = omp_get_wtime();
		rtime[k] = end - start;
		//printf("FSUM = ",fsum);
	}
	for (l = 0; l < 11; l++)
		printf("\nThread=%d\t rtime=%f\n", threads[l], rtime[l]);
	return 0;
}
```

**Compilation and Execution:**

For enabling OpenMP environment use -fopenmp flag while

compiling using **g++ -fopenmp sum_cs.cpp**

For execution use

./a.out

**Observations:**

| Number of Threads | Execution Time | Speed-up | Parallelization Fraction |
|---|---|---|---|
| 1 | 20.70 | 1 | |
| 2 | 10.37 | 1.99 | 99.49 |
| 4 | 5.06 | 4.09 | 100.7 |
| 6 | 4.24 | 4.88 | 95.4 |
| 8 | 4.13 | 5.01 | 91.4 |
| 12 | 4.37 | 4.73 | 86.0 |
| 16 | 4.55 | 4.54 | 83.1 |
| 20 | 3.96 | 5.22 | 85.0 |
| 32 | 4.06 | 5.09 | 82.9 |
| 64 | 3.95 | 5.24 | 82.2 |

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)

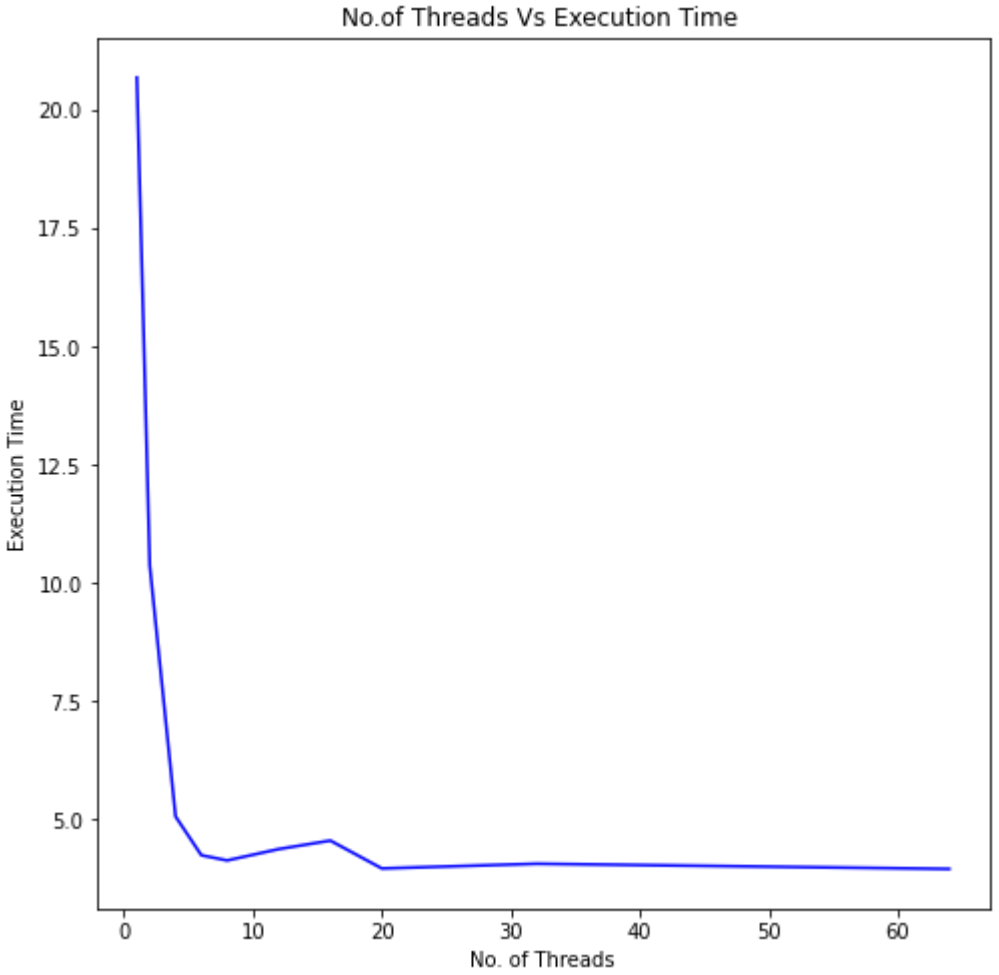
Parallelization Fraction can be found using the following formula, **S(n)=1/((1 - p) + p/n)**

where, S(n) = Speedup for thread count ‘n’

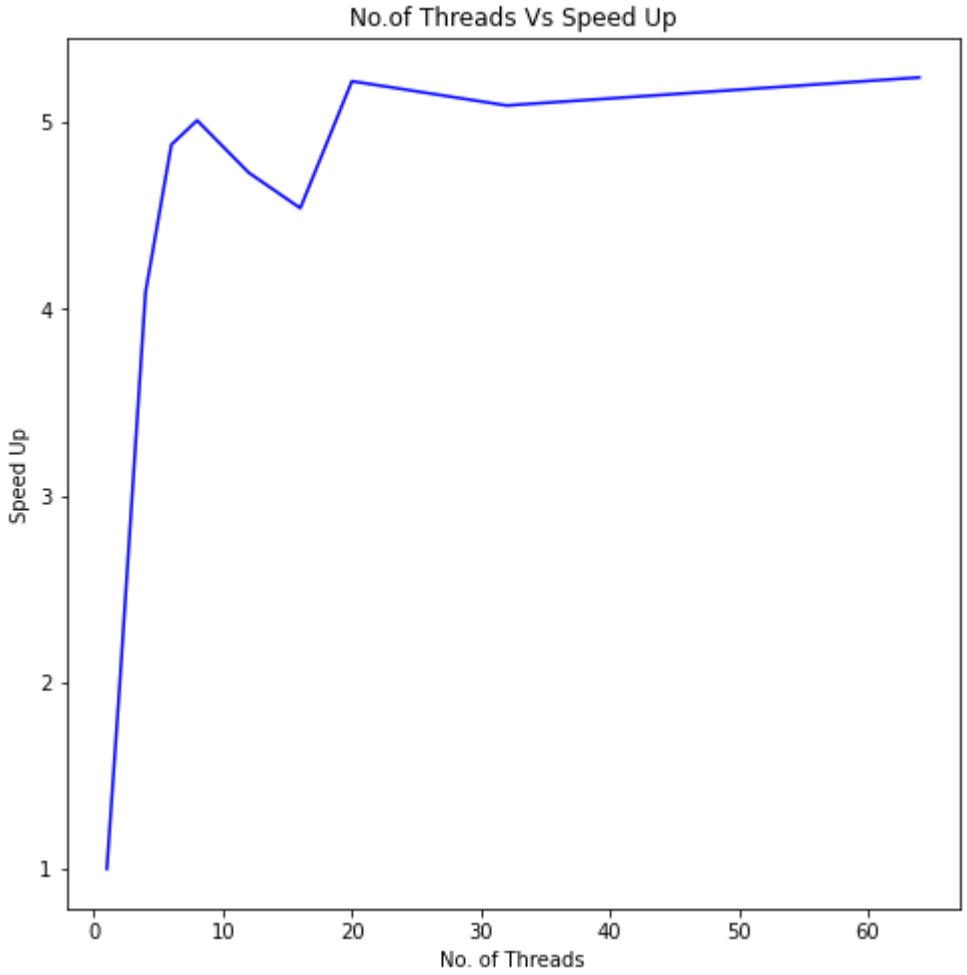
n = Number of threads

p = Parallelization fraction

**Number of Threads 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 20 maximum speedup is observed as the maximum number of parallel threads supported by the hardware is 8.
• If the thread count is more than 10 then the execution time increases slightly and tapers out after 16 threads.