PARALLEL AND DISTRIBUTED COMPUTING LAB

REPORT

NAME: S Shyam Sundaram

REG NO: 19BCE1560

PROGRAMMING ENVIRONMENT: OpenMP

PROBLEM: Vector and Matrix Addition

DATE: 25th August, 2021

HARDWARE CONFIGURATION:

CPU NAME	:	Intel core i5 – 1035G1 @ 1.00 Ghz
Number of Sockets:	:	1
Cores per Socket	:	4
Threads per core	:	1
L1 Cache size	:	320KB
L2 Cache size	:	2MB
L3 Cache size (Shared):		6MB
RAM	:	8 GB

VECTOR ADDITION

CODE

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

#define N 600000

int main()
{
   float a[N],b[N],c[N];
   int i;
   float start,end,exec;
   printf("Name: Shyam Sundaram\nReg num: 19BCE1560\nPDC Lab:\n\n");

   for(i=0;i<N;++i)
   {

      a[i]=(i+1)*1.0;
      b[i]=(i+1)*2.0;
   }

   int thread[]={1,2,4,8,16,32,64,128,256,512};
```

```
    float serial;
    for(int t=0;t<10;++t)
    {
        omp_set_num_threads(thread[t]);
        start=omp_get_wtime();
        #pragma omp parallel default(none), private(i,m), shared(a,b,c)
        {
            #pragma omp for
            for(i=0;i<N;++i)
            {
                for(int j=0;j<1000;++j) //m is 1000 here to increase workload
                c[i]=a[i]+b[i];
            }
        }
        end=omp_get_wtime();
        exec=end-start;
        if(t==0) serial=exec;
        printf("Thread count: %d Time taken is: %f  ",thread[t],exec);
        float pf=(1-(exec/serial))/(1-(1/thread[t]));
        printf(" PF = %f ",pf);
        float s=1-pf;
        float speedup=1/(s+(pf/thread[t]));
        printf(" Speedup = %f\n",speedup);
    }
    return 0;
}
```

COMPILATION AND EXECUTION

```
gcc -fopenmp three.c
./a.out
```

OBSERVATIONS

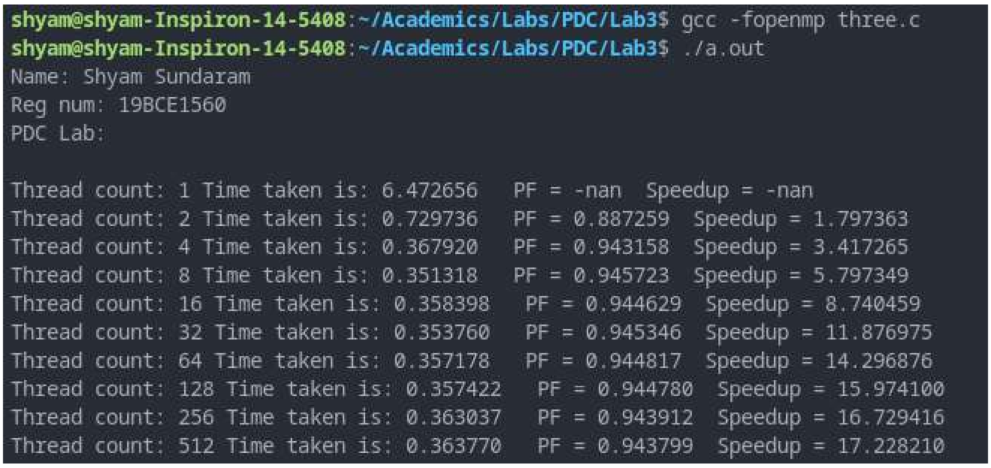
NUMBER OF THREADS	EXECUTION TIME	SPEED-UP
1	6.472656	1
2	0.729736	1.797363
4	0.367920	3.417265
8	0.351318	5.797349
16	0.358398	8.740459
32	0.353760	11.876975
64	0.357178	14.296876
128	0.357422	15.974100
256	0.363037	16.729416
512	0.363770	17.228210

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];
```

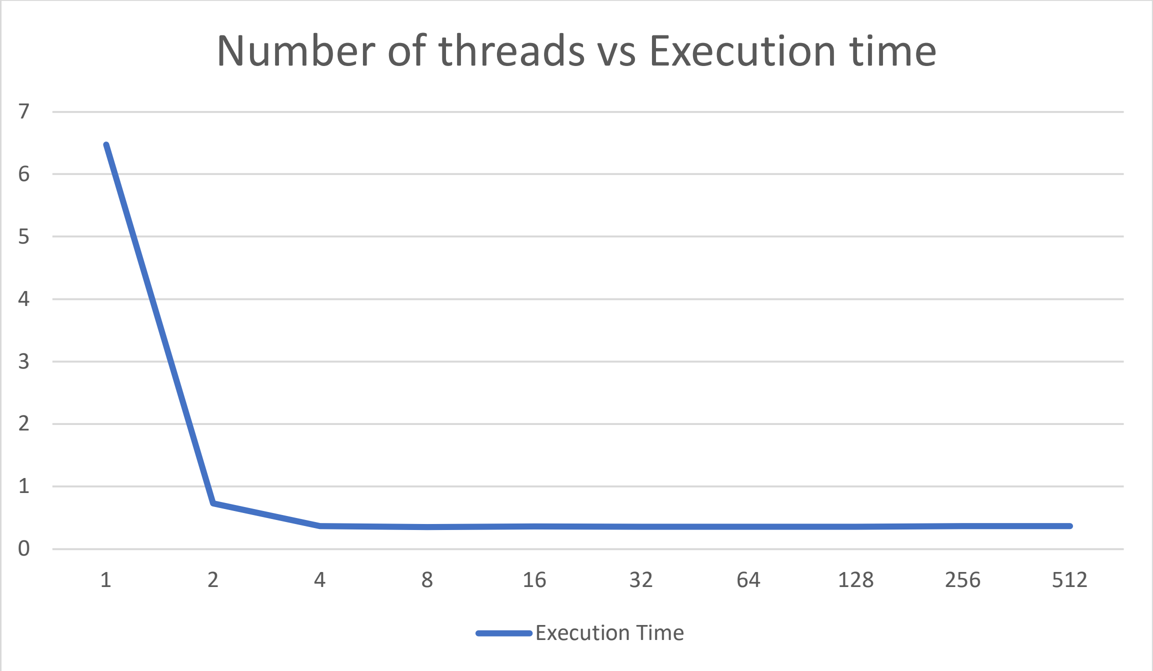
SCREENSHOT

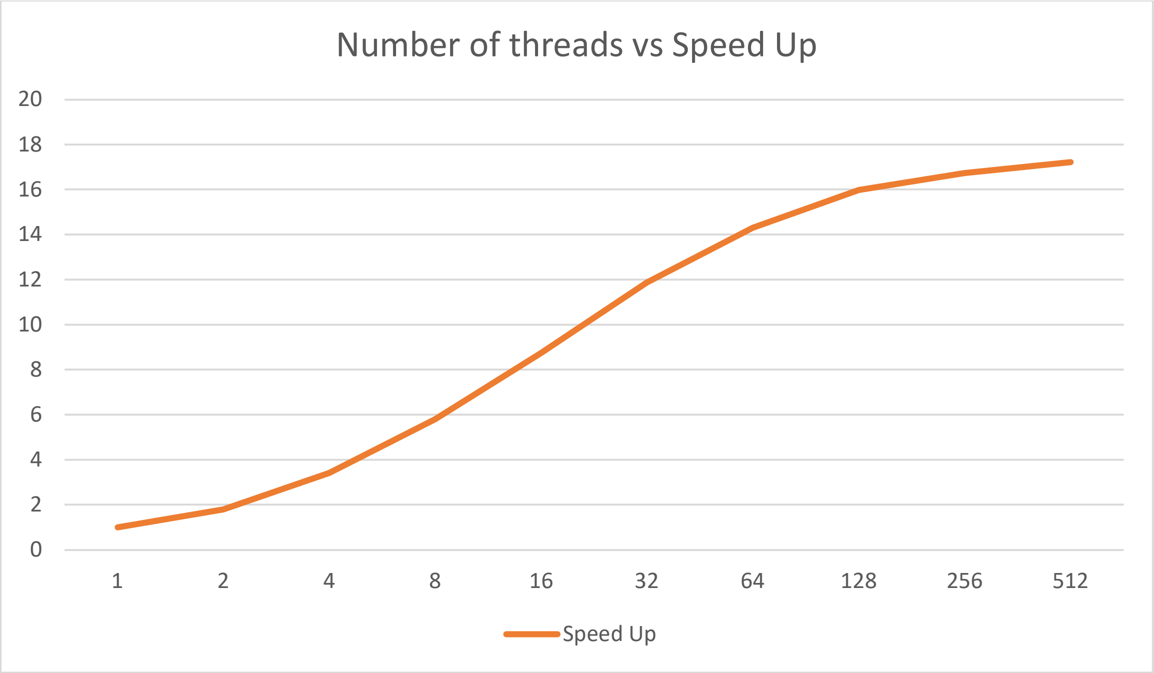


```
shyam@shyam-Inspiron-14-5408:~/Academics/Labs/PDC/Lab3$ gcc -fopenmp three.c
shyam@shyam-Inspiron-14-5408:~/Academics/Labs/PDC/Lab3$ ./a.out
Name: Shyam Sundaram
Reg num: 19BCE1560
PDC Lab:

Thread count: 1 Time taken is: 6.472656   PF = -nan  Speedup = -nan
Thread count: 2 Time taken is: 0.729736   PF = 0.887259  Speedup = 1.797363
Thread count: 4 Time taken is: 0.367920   PF = 0.943158  Speedup = 3.417265
Thread count: 8 Time taken is: 0.351318   PF = 0.945723  Speedup = 5.797349
Thread count: 16 Time taken is: 0.358398   PF = 0.944629  Speedup = 8.740459
Thread count: 32 Time taken is: 0.353760   PF = 0.945346  Speedup = 11.876975
Thread count: 64 Time taken is: 0.357178   PF = 0.944817  Speedup = 14.296876
Thread count: 128 Time taken is: 0.357422   PF = 0.944780  Speedup = 15.974100
Thread count: 256 Time taken is: 0.363037   PF = 0.943912  Speedup = 16.729416
Thread count: 512 Time taken is: 0.363770   PF = 0.943799  Speedup = 17.228210
```

PLOTS





INFERENCE

The addition of the extra for loop increased the workload. Thus, as a greater number of threads work on it, the lower the execution time is and higher the Speed-Up, but up to a certain point, after which it is near constant.

MATRIX ADDITION

CODE

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

#define ROWS 2500
#define COLS 250

int main()
{
   float a[ROWS][COLS],b[ROWS][COLS],c[ROWS][COLS];
   printf("Name: Shyam Sundaram\nReg num: 19BCE1560\nPDC Lab:\n\n");

   for(int i=0;i<ROWS;++i)
   for(int j=0;j<COLS;++j)
   {
      a[i][j]=i*10+j;
      b[i][j]=j*10+i;
   }
   int thread[]={1,2,4,8,16,32,64,128,256,512};
   float serial;
   for(int t=0;t<10;++t)
```

```
    {
        omp_set_num_threads(thread[t]);
        float start=omp_get_wtime();

        #pragma omp parallel for shared(a,b,c) //reduction(+: c)
        for(int i=0;i<ROWS;++i)
        for(int j=0;j<COLS;++j)
        {
            for(int j=0;j<1000;++j)
            c[i][j]=a[i][j]+b[i][j];
        }
        float end=omp_get_wtime();
        float exec=end-start;
        if(t==0) serial=exec;

        printf("Thread count: %d Time taken is: %f",thread[t],exec);
        float pf=(1-(exec/serial))/(1-(1/thread[t]));
        printf(" PF = %f ",pf);
        float s=1-pf;
        float speedup=1/(s+(pf/thread[t]));
        printf(" Speedup = %f\n",speedup);
    }
    return 0;
}
```

COMPILATION AND EXECUTION

```
gcc -fopenmp matadd.c
./a.out
```

OBSERVATIONS

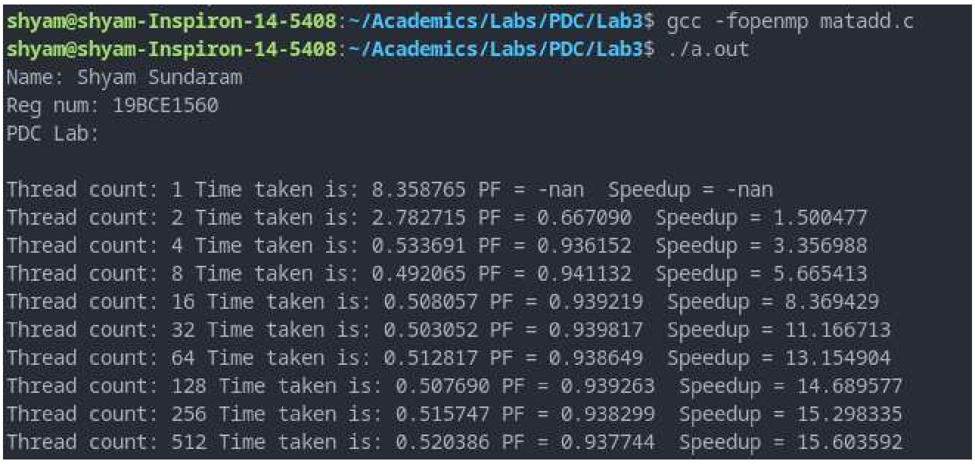
NUMBER OF THREADS	EXECUTION TIME	SPEED-UP
1	8.358765	1
2	2.782715	1.500477
4	0.533691	3.356988
8	0.492065	5.665413
16	0.508057	8.369429
32	0.503052	11.166713
64	0.512817	13.154904
128	0.507690	14.689577
256	0.515747	15.298335
512	0.520386	15.603592

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];
```

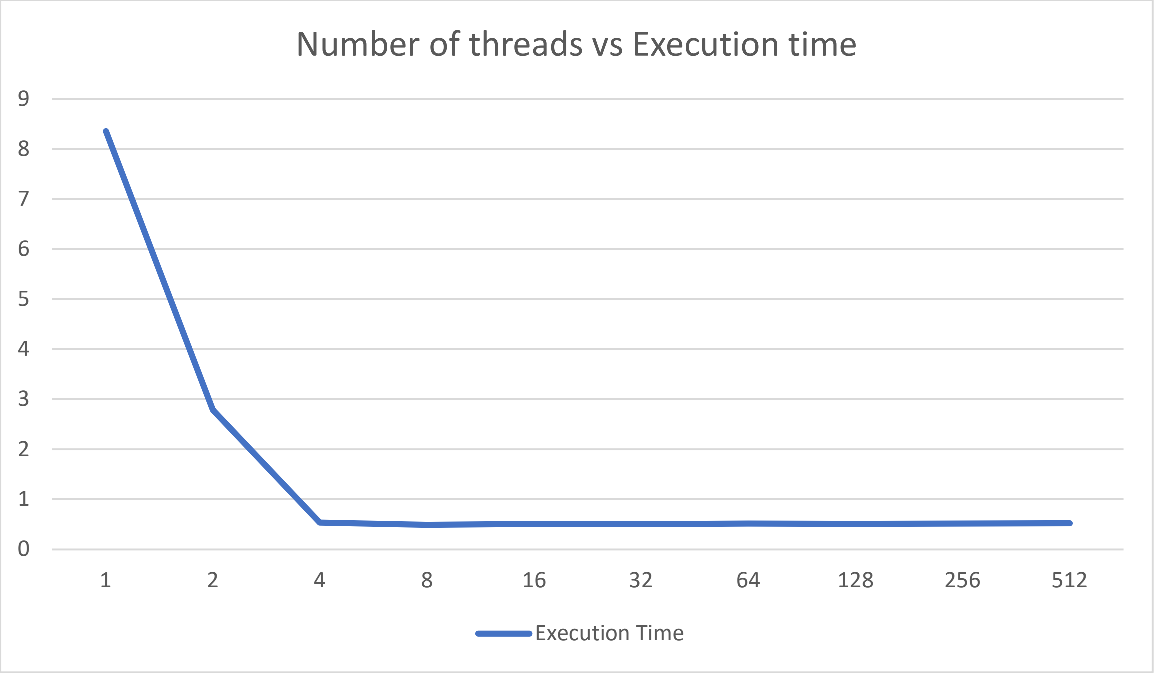
SCREENSHOT

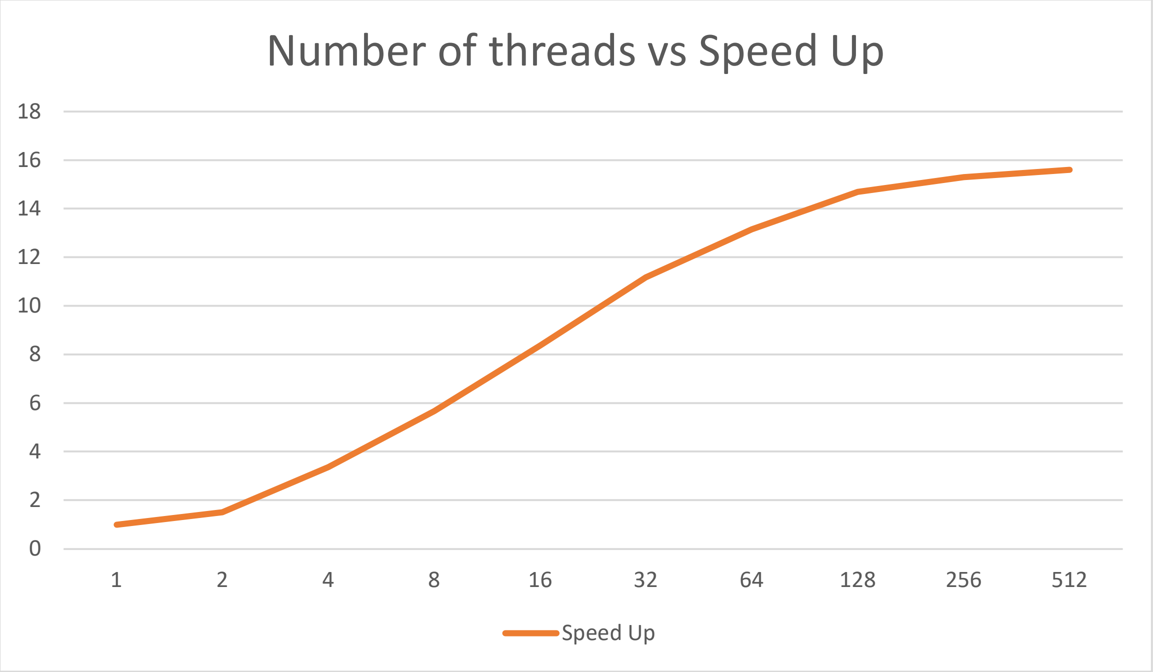


```
shyam@shyam-Inspiron-14-5408:~/Academics/Labs/PDC/Lab3$ gcc -fopenmp matadd.c
shyam@shyam-Inspiron-14-5408:~/Academics/Labs/PDC/Lab3$ ./a.out
Name: Shyam Sundaram
Reg num: 19BCE1560
PDC Lab:

Thread count: 1 Time taken is: 8.358765 PF = -nan  Speedup = -nan
Thread count: 2 Time taken is: 2.782715 PF = 0.667090  Speedup = 1.500477
Thread count: 4 Time taken is: 0.533691 PF = 0.936152  Speedup = 3.356988
Thread count: 8 Time taken is: 0.492065 PF = 0.941132  Speedup = 5.665413
Thread count: 16 Time taken is: 0.508057 PF = 0.939219  Speedup = 8.369429
Thread count: 32 Time taken is: 0.503052 PF = 0.939817  Speedup = 11.166713
Thread count: 64 Time taken is: 0.512817 PF = 0.938649  Speedup = 13.154904
Thread count: 128 Time taken is: 0.507690 PF = 0.939263  Speedup = 14.689577
Thread count: 256 Time taken is: 0.515747 PF = 0.938299  Speedup = 15.298335
Thread count: 512 Time taken is: 0.520386 PF = 0.937744  Speedup = 15.603592
```

PLOTS





INFERENCE

The addition of the extra for loop increased the workload. Thus, as a greater number of threads work on it, the lower the execution time is and higher the Speed-Up, but up to a certain point, after which it is near constant.