Report for HPC LAB

Name: Bhaskar R
Roll No: CED18I009
Programming Environment: CUDA (Google Colab)
Problem: Vector Addition
Date: 17th November 2021

Hardware Configuration:

CPU NAME : Intel(R) Xeon(R) CPU @ 2.30GHz
RAM : 12.69 GB

Serial Code:

```
#include <bits/stdc++.h>
using namespace std;
#define N 1024
int main()
{
   srand(time(0));
   double a[N], b[N], c[N];
   for (int i = 0; i < N; i++)
   {
       a[i] = rand() % 100 + i + 0.250;
       b[i] = rand() % 100 * i + 0.248;
       c[i] = a[i] + b[i];
   }

   for (auto i : c)
       cout << i << endl;
   return 0;
}
```

Parallel Code :

```
%%cu
#include <bits/stdc++.h>
using namespace std;
#define N 25
#define M 1024

__global__ void vector_add(double *a, double *b, double *c)
{
   int id = blockIdx.x * blockDim.x + threadIdx.x;
   if (id < N)
       c[id] = a[id] + b[id];
}

int main()
{
   srand(time(0));
   int blocks[] = {1, 1, 1, 1, 1, 1, 1, 10, 20, 30, 40, 50, M / 2, M /
4, M / 8, M, M, M, M, M};
   int threads[] = {1, 10, 20, 30, 40, 50, M, 10, 10, 10, 10, 10, M, M,
M, M / 2, M / 4, M / 8, M};
   double a[N], b[N], c[N];
   double *d_a, *d_b, *d_c;
   double size = N * sizeof(double);

   // Allocate space for device copies of a, b, c
   cudaMalloc((void **)&d_a, size);
   cudaMalloc((void **)&d_b, size);
   cudaMalloc((void **)&d_c, size);

   for (int i = 0; i < N; i++)
   {
       a[i] = rand() % 100 + i + 0.250;
       b[i] = rand() % 100 * i + 0.248;
   }

   cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);
```

```
    for (int k = 0; k < 19; k++)
    {
        float elapsed = 0;
        cudaEvent_t start, stop;
        cudaEventCreate(&start);
        cudaEventCreate(&stop);

        cudaEventRecord(start, 0);
        vector_add<<<blocks[k], threads[k]>>>(d_a, d_b, d_c);

        // Copy result back to host
        cudaError err = cudaMemcpy(&c, d_c, size,
cudaMemcpyDeviceToHost);
        if (err != cudaSuccess)
            cout << "CUDA Error copying to Host :" <<
cudaGetErrorString(err) << endl;

        cudaEventRecord(stop, 0);
        cudaEventSynchronize(stop);

        cudaEventElapsedTime(&elapsed, start, stop);

        cudaEventDestroy(start);
        cudaEventDestroy(stop);

        printf("Blocks = %4d and Threads per Block = %4d Time = %.5f\n",
blocks[k], threads[k], elapsed);
    }
    cout << "\nSum of Vectors " << endl;
    for (int i = 0; i < N; i++)
        cout << a[i] << " + " << b[i] << " = " << c[i] << endl;

    // Cleanup
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    return 0;
}
```

Output :

```
Blocks =    1 and Threads per Block =    1 Time = 0.42102
Blocks =    1 and Threads per Block =   10 Time = 0.03331
Blocks =    1 and Threads per Block =   20 Time = 0.02893
Blocks =    1 and Threads per Block =   30 Time = 0.03539
Blocks =    1 and Threads per Block =   40 Time = 0.03120
Blocks =    1 and Threads per Block =   50 Time = 0.03133
Blocks =    1 and Threads per Block = 1024 Time = 0.02973
Blocks =   10 and Threads per Block =   10 Time = 0.03184
Blocks =   20 and Threads per Block =   10 Time = 0.05238
Blocks =   30 and Threads per Block =   10 Time = 0.02976
Blocks =   40 and Threads per Block =   10 Time = 0.02765
Blocks =   50 and Threads per Block =   10 Time = 0.04042
Blocks =  512 and Threads per Block = 1024 Time = 0.03562
Blocks =  256 and Threads per Block = 1024 Time = 0.03030
Blocks =  128 and Threads per Block = 1024 Time = 0.03184
Blocks = 1024 and Threads per Block =  512 Time = 0.04250
Blocks = 1024 and Threads per Block =  256 Time = 0.03187
Blocks = 1024 and Threads per Block =  128 Time = 0.03168
Blocks = 1024 and Threads per Block = 1024 Time = 0.05168

 Sum of Vectors
11.25 + 0.248 = 11.498
66.25 + 4.248 = 70.498
97.25 + 156.248 = 253.498
73.25 + 267.248 = 340.498
76.25 + 304.248 = 380.498
6.25 + 175.248 = 181.498
29.25 + 402.248 = 431.498
55.25 + 168.248 = 223.498
9.25 + 336.248 = 345.498
93.25 + 216.248 = 309.498
37.25 + 760.248 = 797.498
68.25 + 484.248 = 552.498
104.25 + 408.248 = 512.498
83.25 + 923.248 = 1006.5
83.25 + 1260.25 = 1343.5
27.25 + 480.248 = 507.498
92.25 + 1232.25 = 1324.5
53.25 + 1207.25 = 1260.5
25.25 + 1062.25 = 1087.5
79.25 + 608.248 = 687.498
55.25 + 280.248 = 335.498
40.25 + 1218.25 = 1258.5
55.25 + 1474.25 = 1529.5
57.25 + 1978.25 = 2035.5
86.25 + 1680.25 = 1766.5
```

Observations:

Number of Blocks	Threads per Block	Execution Time	Speed-up	Parallelization Fraction
1	1	0.421	1.0	
1	10	0.033	12.7576	102.4017
1	20	0.028	15.0357	98.2623
1	30	0.035	12.0286	94.8481
1	40	0.031	13.5806	95.0119
1	50	0.031	13.5806	94.5271
1	1024	0.029	14.5172	93.2026
10	10	0.031	13.5806	102.9295
20	10	0.052	8.0962	97.3873
30	10	0.029	14.5172	103.4574
40	10	0.027	15.5926	103.9852
50	10	0.040	10.525	100.5542
512	1024	0.035	12.0286	91.7761
256	1024	0.030	14.0333	92.9649
128	1024	0.031	13.5806	92.7271
1024	512	0.042	10.0238	90.1999
1024	256	0.031	13.5806	92.9998
1024	128	0.031	13.5806	93.366
1024	1024	0.051	8.2549	87.9719

Speed up can be found using the following formula,

S(n)=T(1)/T(n)

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

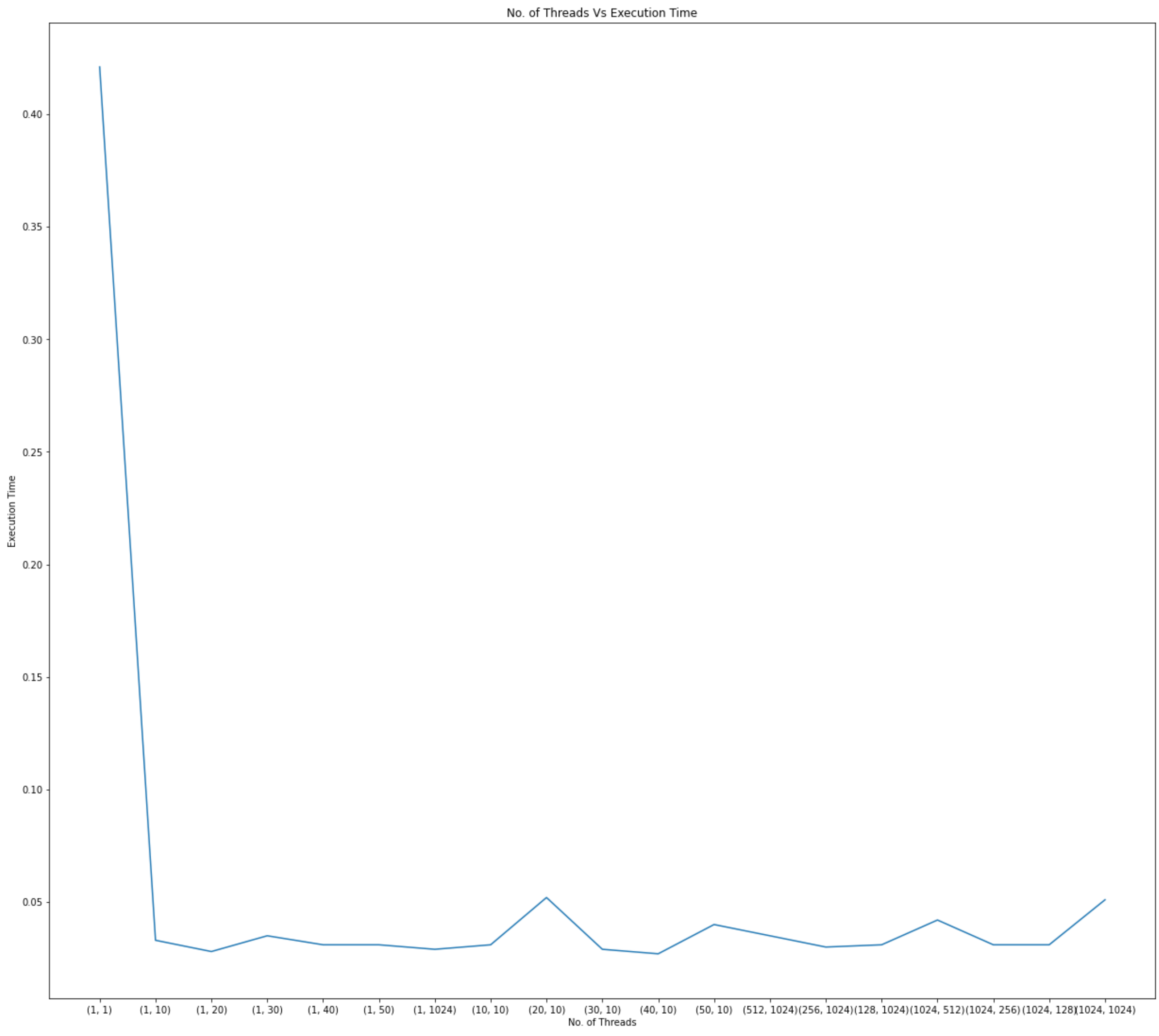
T(1) = Execution Time for Thread count ‘1’ (serial code)

T(n) = Execution Time for Thread count ‘n’ (serial code)

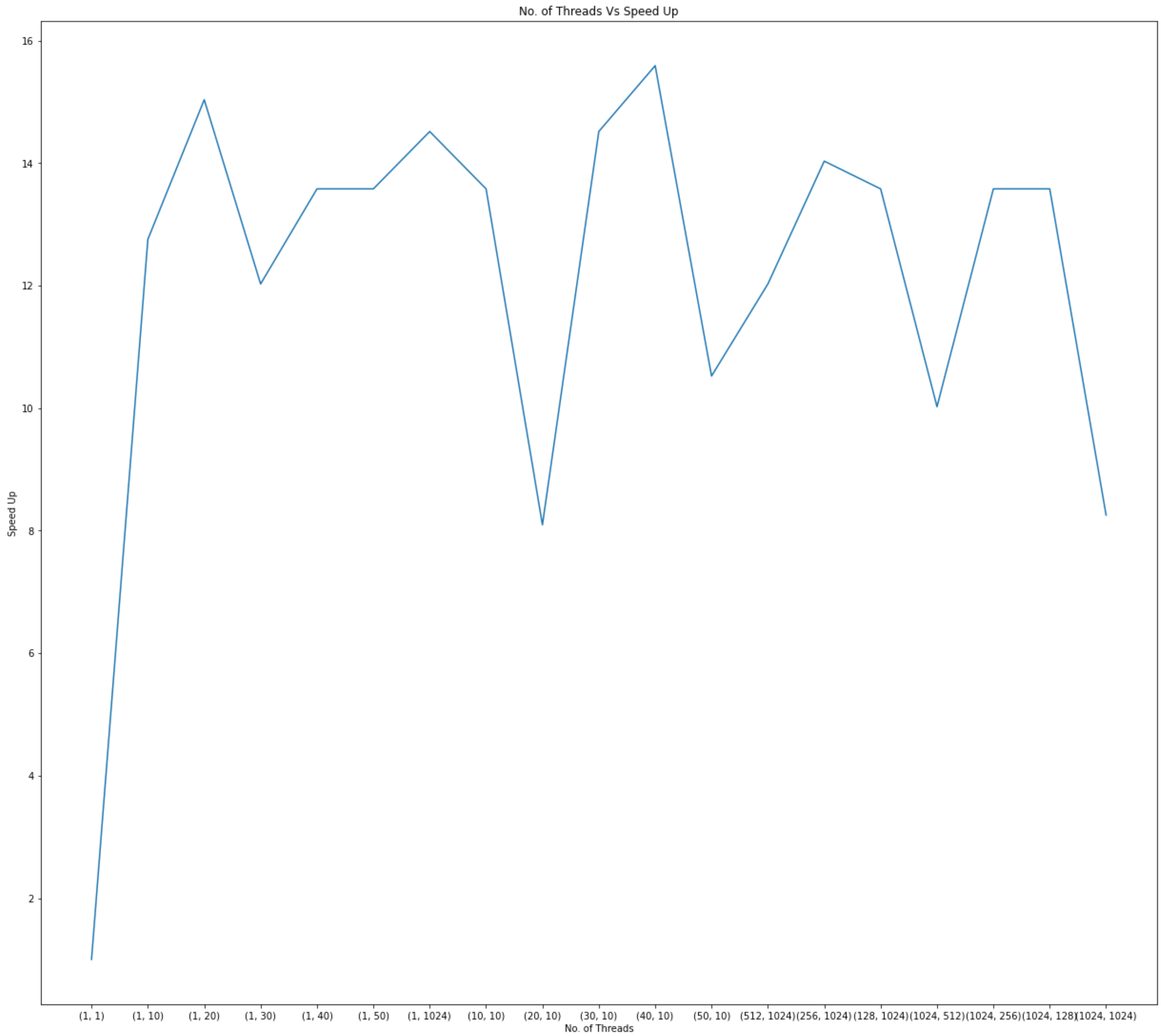
Parallelization Fraction can be found using the following formula, **S(n)=1/((1 - p) + p/n)**

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

No. of Threads Vs Execution Time



No. of Threads Vs Speed Up



Inference:

- For (1,1) the execution time is maximum, i.e poor performance. This is because there is no parallel execution.
- The Striding technique was used in the vector_add function for the different combinations of no. of blocks and no. of threads.
- The Maximum speedup was for 40 number blocks with 10 threads per block combination. This is because it has reasonably fewer communication overheads and also a good amount of parallelization