Report for HPC LAB

Name: Bhaskar R
Roll No: CED18I009
Programming Environment: CUDA (Google Colab)
Problem: Matrix 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 10
int main()
{
   srand(time(0));
   int a[N][N], b[N][N], c[N][N];
   for (int i = 0; i < N; i++)
       for (int j = 0; j < N; j++)
           a[i][j] = rand() % 10 + i * j;

   for (int i = 0; i < N; i++)
       for (int j = 0; j < N; j++)
           b[i][j] = rand() % 10 + i * j;

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

   for (int i = 0; i < N; i++)
   {
       for (int j = 0; j < N; j++)
           cout << c[i][j] << "\t";
       cout << endl;
   }
   return 0;
}
```

Parallel Code :

```
%%cu
#include <bits/stdc++.h>
using namespace std;
#define N 7
#define M 1024

__global__ void matadd(double a[][N], double b[][N], double c[][N])
{
    int id = gridDim.x * blockIdx.y + blockIdx.x;
    if (id < N)
    {
        for (int i = 0; i < N; i++)
            c[id][i] = a[id][i] + b[id][i];
    }
}

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][N], b[N][N], c[N][N] = {{0}};
    double(*d_a)[N], (*d_b)[N], (*d_c)[N];
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            a[i][j] = i + j + 0.250;
            b[i][j] = i + j + 0.248;
        }
    }
    cudaMalloc((void **)&d_a, N * N * sizeof(double));
    cudaMalloc((void **)&d_b, N * N * sizeof(double));
    cudaMalloc((void **)&d_c, N * N * sizeof(double));
```

```
    cudaMemcpy(d_a, a, N * N * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, N * N * sizeof(double), cudaMemcpyHostToDevice);
    dim3 grid(N, N);

    for (int k = 0; k < 19; k++)
    {
        float elapsed = 0;
        cudaEvent_t start, stop;
        cudaEventCreate(&start);
        cudaEventCreate(&stop);
        cudaEventRecord(start, 0);
        matadd<<<blocks[k], threads[k]>>>(d_a, d_b, d_c);

        cudaError err = cudaMemcpy(c, d_c, N * N * sizeof(double),
cudaMemcpyDeviceToHost);
        if (err != cudaSuccess)
            cout << "CUDA Error copying to Host: " <<
cudaGetErrorString(err);
        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);
    }

    printf("\nMatrix A:\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)

            cout << a[i][j] << "\t";

        cout << endl;
    }
    printf("\nMatrix B:\n");
```

```
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            cout << b[i][j] << "\t";
        cout << endl;
    }
    printf("\nSum :\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            cout << c[i][j] << "\t";
        cout << endl;
    }
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    return 0;
}
```

Output :

```
Blocks =    1 and Threads per Block =    1 Time = 0.19187
Blocks =    1 and Threads per Block =   10 Time = 0.07840
Blocks =    1 and Threads per Block =   20 Time = 0.04045
Blocks =    1 and Threads per Block =   30 Time = 0.03354
Blocks =    1 and Threads per Block =   40 Time = 0.03056
Blocks =    1 and Threads per Block =   50 Time = 0.04525
Blocks =    1 and Threads per Block = 1024 Time = 0.02554
Blocks =   10 and Threads per Block =   10 Time = 0.03050
Blocks =   20 and Threads per Block =   10 Time = 0.03066
Blocks =   30 and Threads per Block =   10 Time = 0.02992
Blocks =   40 and Threads per Block =   10 Time = 0.02998
Blocks =   50 and Threads per Block =   10 Time = 0.03011
Blocks =  512 and Threads per Block = 1024 Time = 0.04966
Blocks =  256 and Threads per Block = 1024 Time = 0.04954
Blocks =  128 and Threads per Block = 1024 Time = 0.03712
Blocks = 1024 and Threads per Block =  512 Time = 0.04938
Blocks = 1024 and Threads per Block =  256 Time = 0.05040
Blocks = 1024 and Threads per Block =  128 Time = 0.03917
Blocks = 1024 and Threads per Block = 1024 Time = 0.07773

Matrix A:
0.25    1.25    2.25    3.25    4.25    5.25    6.25
1.25    2.25    3.25    4.25    5.25    6.25    7.25
2.25    3.25    4.25    5.25    6.25    7.25    8.25
3.25    4.25    5.25    6.25    7.25    8.25    9.25
4.25    5.25    6.25    7.25    8.25    9.25    10.25
5.25    6.25    7.25    8.25    9.25    10.25   11.25
6.25    7.25    8.25    9.25    10.25   11.25   12.25

Matrix B:
0.248   1.248   2.248   3.248   4.248   5.248   6.248
1.248   2.248   3.248   4.248   5.248   6.248   7.248
2.248   3.248   4.248   5.248   6.248   7.248   8.248
3.248   4.248   5.248   6.248   7.248   8.248   9.248
4.248   5.248   6.248   7.248   8.248   9.248   10.248
5.248   6.248   7.248   8.248   9.248   10.248  11.248
6.248   7.248   8.248   9.248   10.248  11.248  12.248

Sum :
0.498   2.498   4.498   6.498   8.498   10.498  12.498
2.498   4.498   6.498   8.498   10.498  12.498  14.498
4.498   6.498   8.498   10.498  12.498  14.498  16.498
6.498   8.498   10.498  12.498  14.498  16.498  18.498
8.498   10.498  12.498  14.498  16.498  18.498  20.498
10.498  12.498  14.498  16.498  18.498  20.498  22.498
12.498  14.498  16.498  18.498  20.498  22.498  24.498
```

Observations:

Number of Blocks	Threads per Block	Execution Time	Speed-up	Parallelization Fraction
1	1	0.191	1.0	
1	10	0.078	2.4487	65.7356
1	20	0.040	4.775	83.2185
1	30	0.033	5.7879	85.5751
1	40	0.030	6.3667	86.4546
1	50	0.045	4.2444	77.9995
1	1024	0.025	7.64	86.996
10	10	0.030	6.3667	93.6592
20	10	0.030	6.3667	93.6592
30	10	0.029	6.5862	94.2408
40	10	0.029	6.5862	94.2408
50	10	0.030	6.3667	93.6592
512	1024	0.049	3.898	74.4185
256	1024	0.049	3.898	74.4185
128	1024	0.037	5.1622	80.7072
1024	512	0.049	3.898	74.4913
1024	256	0.050	3.82	74.1115
1024	128	0.039	4.8974	80.2076
1024	1024	0.077	2.4805	59.7439

Speed up can be found using the following formula,

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

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

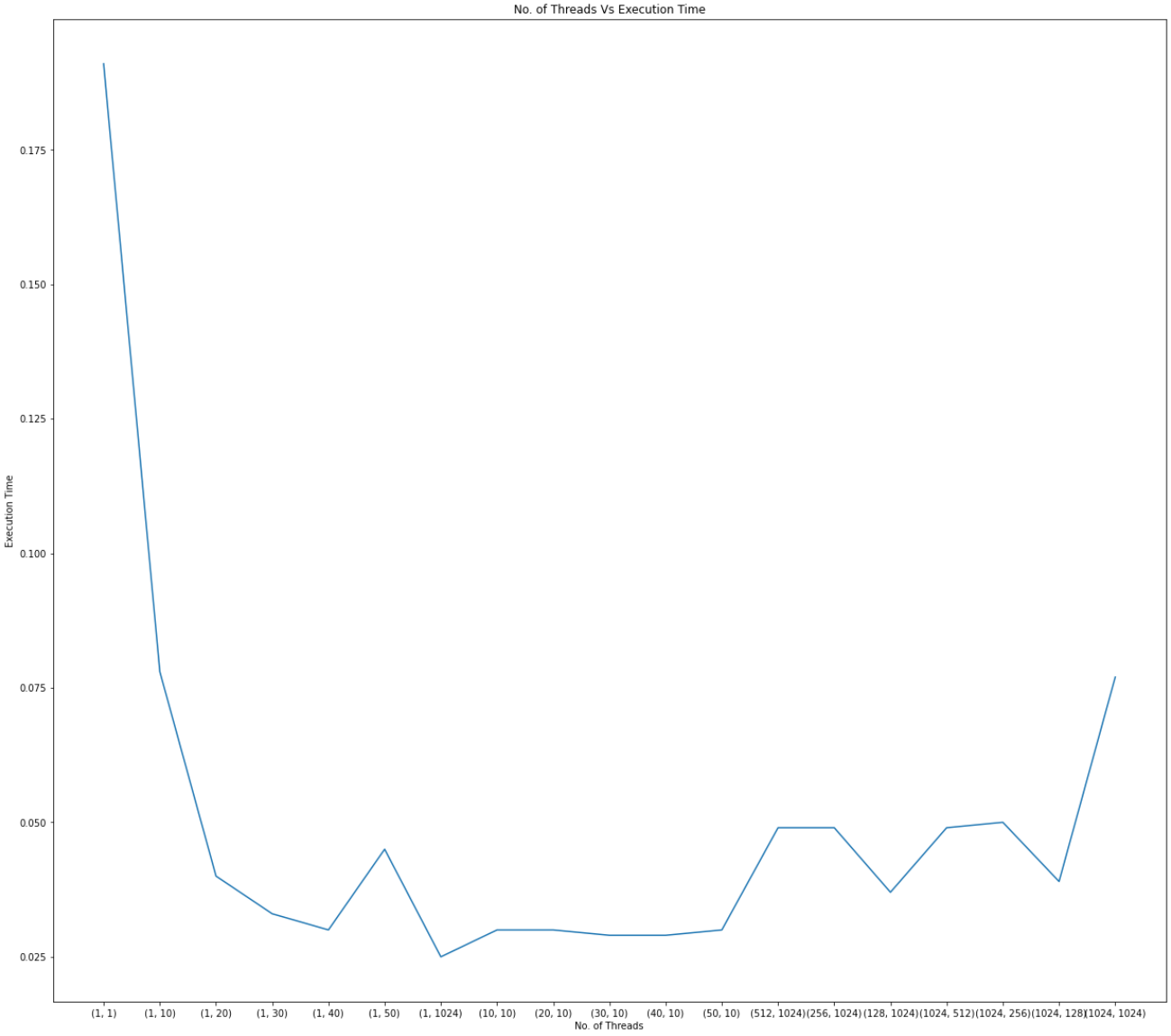
T(1) = Execution Time for Thread count ‘1’ (serial code)

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

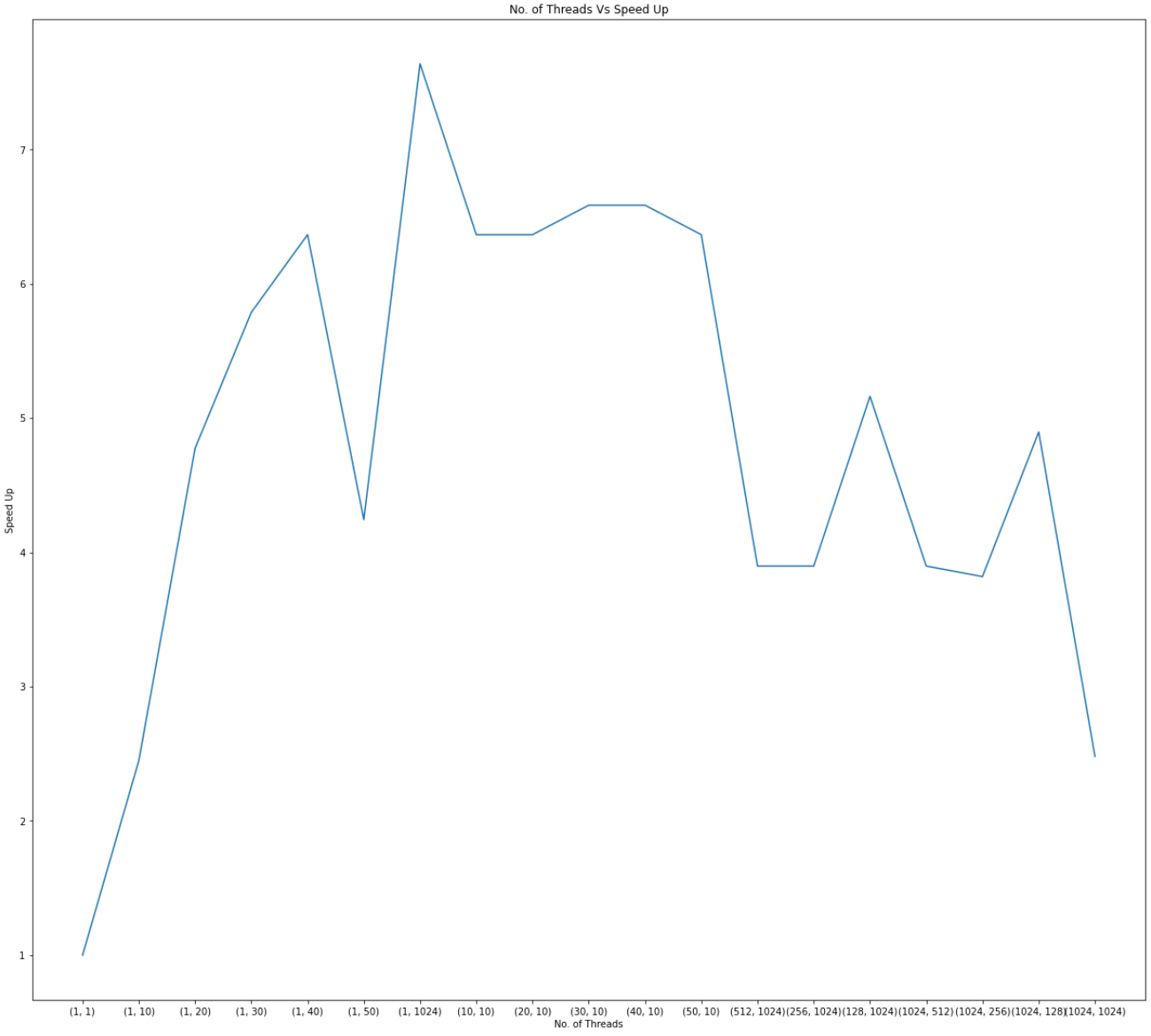
Parallelization Fraction can be found using the following formula, **S(n)=1/((1 - p) + p/n)**

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 matadd function for the different combinations of no. of blocks and no. of threads.
- The Maximum speedup was for 1 number block with 1024 threads per block combination. This is because it has reasonably fewer communication overheads and also a good amount of parallelization