Report for HPC LAB

Name: Bhaskar R
Roll No: CED18I009
Programming Environment: CUDA (Google Colab)
Problem: Matrix Multiplication
Date: 20th 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;

int main(int argc, char *argv[])
{
   int i, j, k, n = 512;
   double a[n][n], b[n][n], c[n][n];
   for (i = 0; i < n; i++)
   {
       for (j = 0; j < n; j++)
       {
           a[i][j] = i + j;
           b[i][j] = i + j;
           c[i][j] = 0;
           for (k = 0; k < n; k++)
               c[i][j] += a[i][k] * b[k][j];
       }
   }
   return 0;
}
```

Parallel Code :

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

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

int main()
{
    srand(time(0));
    int blocks[] = {1, 1, 1, 1, 1, 1, 1, 10, 20, 30, 40, 50, M / 8, M /
4, M / 2, M, M, M, M, M};
    int threads[] = {1, 10, 20, 30, 40, 50, M, 10, 10, 10, 10, 10, M,
M, M, M / 8, M / 4, M / 2, 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);
        matmul<<<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("\nProduct :\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 = 0.33712
Blocks =    1 and Threads per Block =   10 Time = 0.06826
Blocks =    1 and Threads per Block =   20 Time = 0.06941
Blocks =    1 and Threads per Block =   30 Time = 0.06627
Blocks =    1 and Threads per Block =   40 Time = 0.06416
Blocks =    1 and Threads per Block =   50 Time = 0.07898
Blocks =    1 and Threads per Block = 1024 Time = 0.07853
Blocks =   10 and Threads per Block =   10 Time = 0.06266
Blocks =   20 and Threads per Block =   10 Time = 0.06982
Blocks =   30 and Threads per Block =   10 Time = 0.08154
Blocks =   40 and Threads per Block =   10 Time = 0.11987
Blocks =   50 and Threads per Block =   10 Time = 0.06835
Blocks =  128 and Threads per Block = 1024 Time = 0.06650
Blocks =  256 and Threads per Block = 1024 Time = 0.10205
Blocks =  512 and Threads per Block = 1024 Time = 0.06794
Blocks = 1024 and Threads per Block =  128 Time = 0.07533
Blocks = 1024 and Threads per Block =  256 Time = 0.06086
Blocks = 1024 and Threads per Block =  512 Time = 0.06026
Blocks = 1024 and Threads per Block = 1024 Time = 0.06272

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

Product :
101.892 124.642 147.392 170.142 192.892 215.642 238.392
124.628 154.378 184.128 213.878 243.628 273.378 303.128
147.364 184.114 220.864 257.614 294.364 331.114 367.864
170.1   213.85  257.6   301.35  345.1   388.85  432.6
192.836 243.586 294.336 345.086 395.836 446.586 497.336
215.572 273.322 331.072 388.822 446.572 504.322 562.072
238.308 303.058 367.808 432.558 497.308 562.058 626.808
```

Observations:

Number of Blocks	Threads per Block	Execution Time	Speed-up	Parallelization Fraction
1	1	0.337	1.0	
1	10	0.068	4.9559	88.6911
1	20	0.069	4.8841	83.7109
1	30	0.066	5.1061	83.1885
1	40	0.064	5.2656	83.086
1	50	0.078	4.3205	78.423
1	1024	0.078	4.3205	76.9297
10	10	0.062	5.4355	90.6694
20	10	0.069	4.8841	88.3616
30	10	0.081	4.1605	84.4049
40	10	0.119	2.8319	71.8756
50	10	0.068	4.9559	88.6911
128	1024	0.067	5.0299	80.1972
256	1024	0.102	3.3039	69.8009
512	1024	0.066	5.1061	80.4942
1024	128	0.060	5.6167	82.8432
1024	256	0.060	5.6167	82.5183
1024	512	0.075	4.4933	77.8968
1024	1024	0.062	5.4355	81.6822

Speed up can be found using the following formula,

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

where, $S(n)$ = Speedup for thread count 'n'

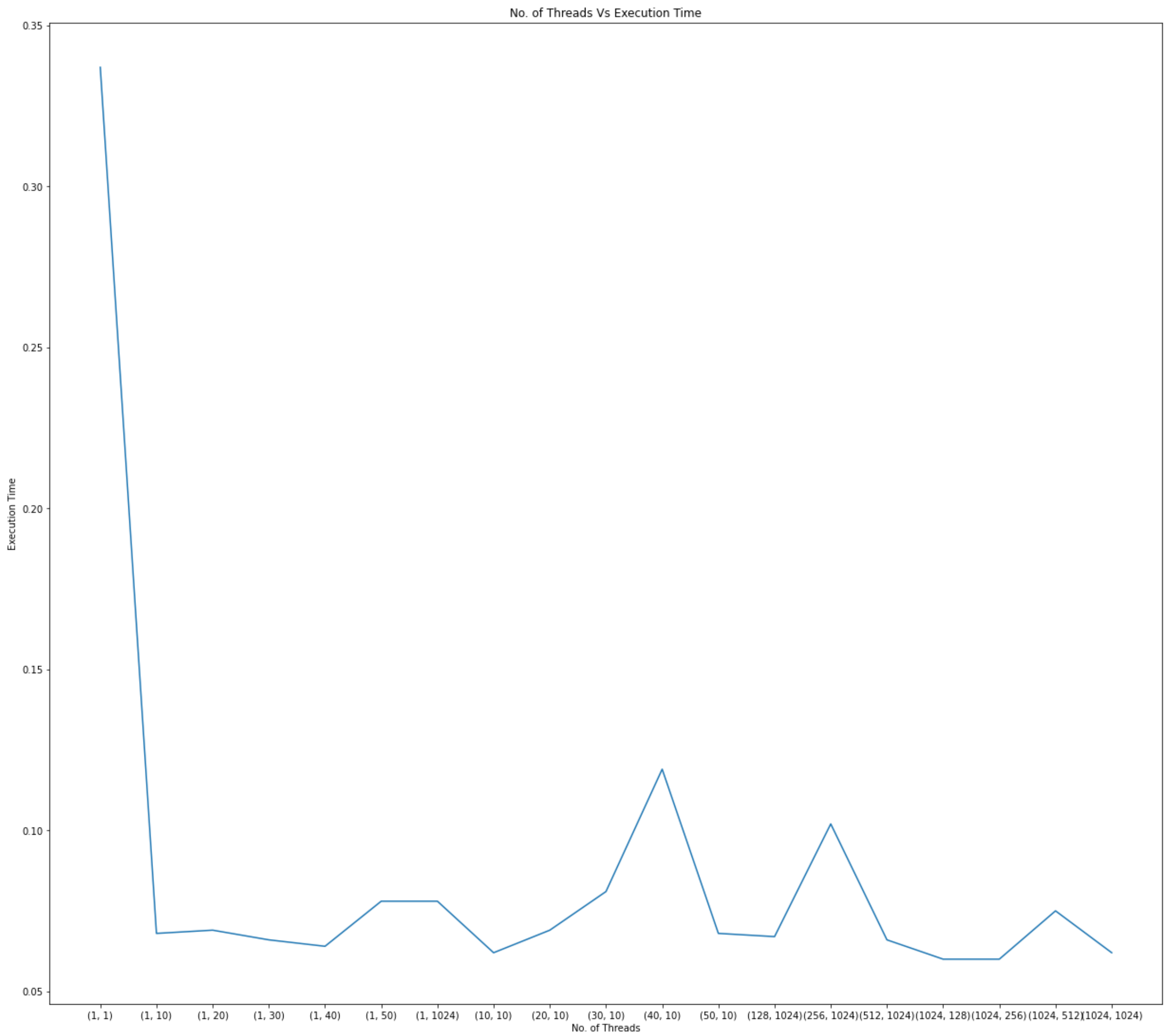
$T(1)$ = Execution Time for Thread count '1' (serial code)

$T(n)$ = Execution Time for Thread count 'n' (serial code)

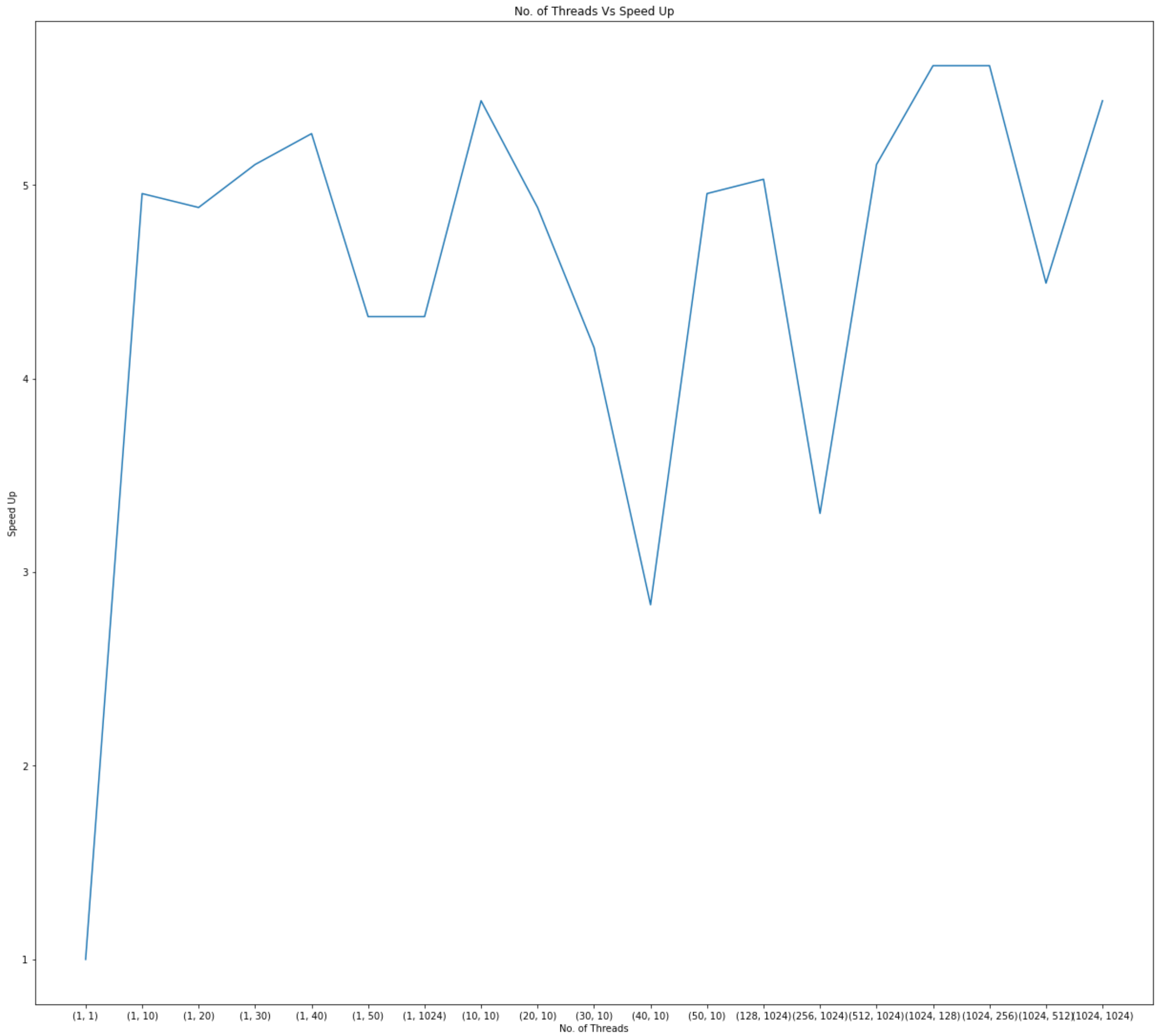
Parallelization Fraction can be found using the following formula, **S(n)=1/((1 - p) + p/n)**

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 matmul function for the different combinations of no. of blocks and no. of threads.
- The Maximum speedup was for 1024 number blocks with 256 threads per block combination. This is because it has reasonably fewer communication overheads and also a good amount of parallelization