# Computer Architecture
# Lab Three


Prepared By

Ahmed Aly Gamal El-Din El-Ghannam

*Electronics and Communications - Level 4*
*ID: 19015292*

Yahia Walid El-Dakhakhny

*Electronics and Communications - Level 4*
*ID: 19016891*


May 2024

# Contents

# List of Figures

# Listings

# 1 Introduction

This report contains code—and rigorous analysis of said code—written using normal C++ and using the Cuda framework in an effort to build a solid understanding on the differences between vector processors and ordinary processors. A program that calculates the matrix equation shown in 1 is run on both at different matrix sizes. Additionally, profiling has been used to observe each function's timing to know what kind of operation consumes time in each version of the program. All code, profiling output files, and screenshots can be found in the project's Github repository.

$$\boldsymbol{R} = \boldsymbol{C}(\boldsymbol{A} \times \boldsymbol{B} + \boldsymbol{B} \times \boldsymbol{A}) \tag{1}$$

# 2 Prerequisites

This section contains the host system specifications used to run the two versions of the code, as well as a quick summary of the steps I followed in order to get my setup ready to compile Cuda code.

## 2.1 Host System Specifications

My setup is as shown in figure 1.

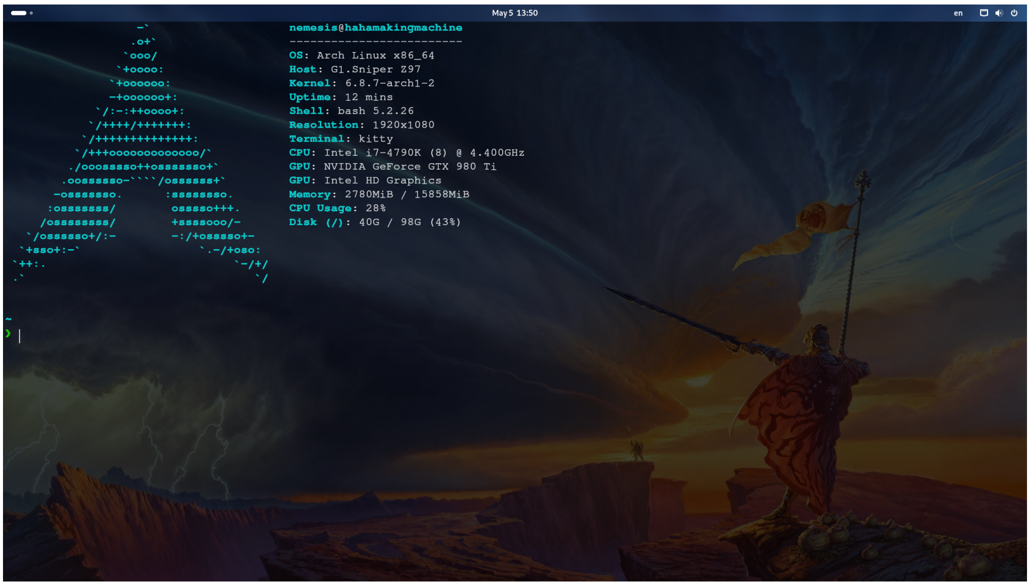



Figure 1: Host System Specifications

The GPU used is a GTX980ti: a flagship GPU released in 2015. The specifications relevant to this report are:

- Streaming Multiprocessor Count (SMC): 22 (each clocked at 1216MHz)
- CUDA Core Count (CCC): 2816 (SMC $\times 128$)
- Cache: 48 KB (L1 – per SM) & 3 MB (L2)
- Memory: 6GB of DDR5 (clocked at 1800 MHz)
- Memory Bus: 384 bit
- Memory Bandwidth: 345.6 GB/s

Understanding the GPU specifications is crucial for optimizing the block size and number of threads passed to the kernel—more on that later.

## 2.2 Installation Process

The process of installing Cuda was as follows:

1. Install cuda using

```
sudo pacman -S cuda
```

2. To make sure it was installed successfully:

```
nvcc --version
```

3. Export Cuda library directory into '.bashrc' as an environment variable.

4. Add the following library in the .cu program:

```
#include <cuda_runtime.h>
```

## 2.3 Compilation Settings

Since this lab's main purpose is to compare CPU and GPU performance in executing the same operation, CPU memory allocation has been avoided when defining the variables holding the matrix and the stack is instead used to make the comparison as fair as possible. Bare in mind that the default stack size would not be able to hold the matrices at large values of N which would result in a segmentation fault; so, the following command was used to extend the stack size—which is not a best practice but it is purely for testing purposes:

```
ulimit -s 4096000
```

When compiling the C++ version, I used gprof as a profiling tool to get a good estimate for how long does each function take to execute. So, the C++ code is compiled as follows:

```
g++ -pg nocuda.cpp -o nocuda
```

The program then have to at least be run once in order to extract timing information using the following command:

```
gprof nocuda gmon.out > analysis.txt
```

When compiling the Cuda version, nvcc is used as the compiler and, luckily, Nvidia's provided nvprof tool grants way more accurate timing information even without extra flags. To compile, simply run:

```
nvcc cuda.cu -o cuda
```

To analyse using nvprof, run:

```
time nvprof ./cuda
```

In the previous command, the shell command 'time' is used to give me the program's total runtime.

# 3 CPU Version of The Program

In this section, the CPU version of the program will be explained function-by-function. Additionally, runtime and profiling information will observed at different values of N. ***Note that gprof is a bit inaccurate since its lowest time unit is 0.01s; ergo, at low values of N, information gathered by gprof will not be provided.***

## 3.1 Program Code

The program simply generates 3 square matrices—as well as 2 empty matrices: one as a temporary variable and one to store the final result—of size NxN and populates them with random numbers between 0 and 49. The generated matrices, as well as the matrix resulting from the operation are all written in comma separated (csv) files for debugging purposes.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <fstream>

using namespace std;

#define N 5000 // matrix row/col
```

Code Snippet 1: Included Libraries & Macro

```
void populateMatrix(unsigned int (&matrix)[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            matrix[i][j] = rand() % 50; // Generate random numbers
               between 0 and 49 (inclusive)
        }
    }
}
```

Code Snippet 2: Function to Populate Matrices with Random Values

```
void printMatrix(const unsigned int (&matrix)[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}
```

Code Snippet 3: Function to Print Matrix (For Testing Only)

```
void addMatrix(const unsigned int (&matrixA)[N][N], const unsigned int
    (&matrixB)[N][N], unsigned int (&matrixC)[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            matrixC[i][j] = matrixA[i][j] + matrixB[i][j];
        }
    }
}
```

Code Snippet 4: Function to Add Two Matrices

```
void multMatrix(const unsigned int (&matrixA)[N][N], const unsigned int
    (&matrixB)[N][N], unsigned int (&matrixC)[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            matrixC[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                matrixC[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }
}
```

Code Snippet 5: unction to Multiply Two Matrices

```
void csvMatrix(const unsigned int (&matrix)[N][N], const char *filename)
{
    std::ofstream file(filename);
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            file << matrix[i][j];
            if (j < N - 1)
            {
                file << ",";
            }
        }
        file << "\n";
    }
    file.close();
```

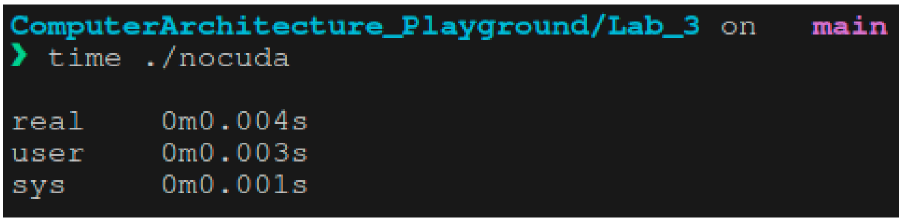
Code Snippet 6: Write Matrix in CSV File

```
int main()
{
    // random number generation shenanigans
    srand(time(NULL));

    // define matrices
    unsigned int matA[N][N];
    unsigned int matB[N][N];
    unsigned int matC[N][N];
    unsigned int matRes[N][N];
    unsigned int matTemp[N][N];

    // populate matrix A && B
    populateMatrix(matA);
    populateMatrix(matB);
    populateMatrix(matC);

    // output matrix A && B as csv files for references
    csvMatrix(matA, "MatrixA.csv");
    csvMatrix(matB, "MatrixB.csv");
    csvMatrix(matC, "MatrixC.csv");

    // C * ((A * B) + (B * A))
    multMatrix(matA, matB, matRes); // A * B = res

    multMatrix(matB, matA, matTemp); // B * A = temp

    addMatrix(matRes, matTemp, matTemp); // res + temp = temp

    multMatrix(matC, matTemp, matRes); // C * temp = res

    // output matrix C result for reference
    csvMatrix(matRes, "Result.csv");

    return 0;
}
```

Code Snippet 7: Main Function Implementation

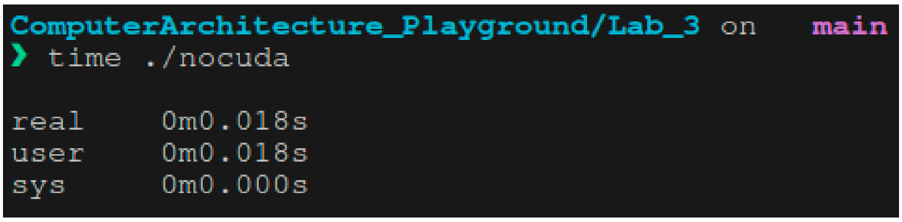
## 3.2 Runtime & Profiling Results

In this section, runtime and profiling information will be observed for 6 values of N: 50, 100, 500, 1000, 2000, and 5000.



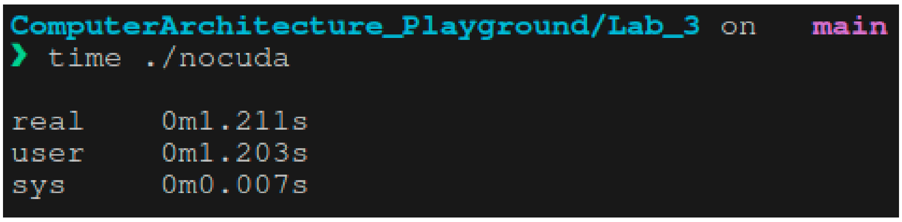
```
ComputerArchitecture_Playground/Lab_3 on  main
❯ time ./nocuda

real    0m0.004s
user    0m0.003s
sys     0m0.001s
```

Figure 2: Runtime at N = 50



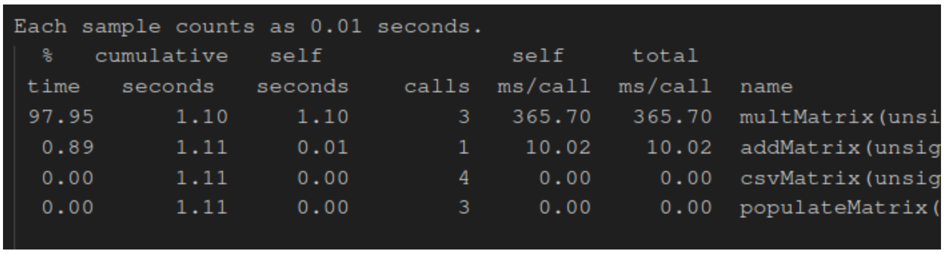
```
ComputerArchitecture_Playground/Lab_3 on  main
❯ time ./nocuda

real    0m0.018s
user    0m0.018s
sys     0m0.000s
```

Figure 3: Runtime at N = 100



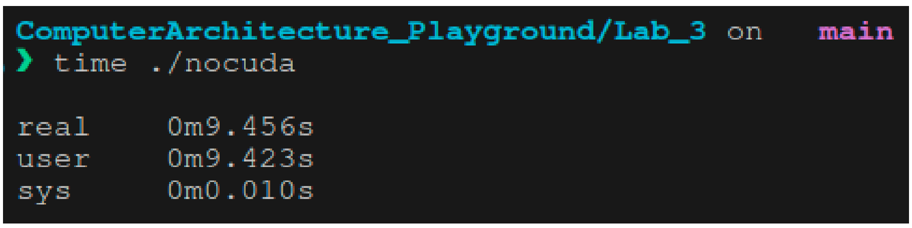
```
ComputerArchitecture_Playground/Lab_3 on  main
❯ time ./nocuda

real    0m1.211s
user    0m1.203s
sys     0m0.007s
```

Figure 4: Runtime at N = 500



```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 97.95      1.10     1.10        3   365.70   365.70  multMatrix(unsi
  0.89      1.11     0.01        1    10.02    10.02  addMatrix(unsig
  0.00      1.11     0.00        4     0.00     0.00  csvMatrix(unsig
  0.00      1.11     0.00        3     0.00     0.00  populateMatrix(
```

Figure 5: Function Timing at N = 500 (gprof)



```
ComputerArchitecture_Playground/Lab_3 on  main
❯ time ./nocuda

real    0m9.456s
user    0m9.423s
sys     0m0.010s
```

Figure 6: Runtime at N = 1000

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 98.45      9.08     9.08        3     3.03     3.03  multMatrix(unsi
  0.11      9.09     0.01        4     0.00     0.00  csvMatrix(unsig
  0.11      9.10     0.01        1     0.01     0.01  addMatrix(unsig
  0.00      9.10     0.00        3     0.00     0.00  populateMatrix(
```

Figure 7: Function Timing at N = 1000 (gprof)

```
ComputerArchitecture_Playground/Lab_3 on  main
❯ time ./nocuda

real    1m29.660s
user    1m29.257s
sys     0m0.175s
```

Figure 8: Runtime at N = 2000

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.30     87.95    87.95        3    29.32    29.32  multMatrix(unsi
  0.03     87.98     0.03        4     0.01     0.01  csvMatrix(unsig
  0.02     88.00     0.02        3     0.01     0.01  populateMatrix(
  0.01     88.01     0.01        1     0.01     0.01  addMatrix(unsig
```

Figure 9: Function Timing at N = 2000 (gprof)

```
ComputerArchitecture_Playground/Lab_3 on  main
❯ time ./nocuda

real    35m43.403s
user    35m39.282s
sys     0m0.613s
```

Figure 10: Runtime at N = 5000

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.46   1470.95  1470.95        3   490.32   490.32  multMatrix(unsig
  0.01   1471.12     0.17        4     0.04     0.04  csvMatrix(unsign
  0.01   1471.26     0.14        3     0.05     0.05  populateMatrix(u
  0.00   1471.32     0.06        1     0.06     0.06  addMatrix(unsign
  0.00   1471.35     0.03                             _init
```

Figure 11: Function Timing at N = 5000 (gprof)

## 3.3 Observation

At low values of N, the CPU code is blazing fast and outperforms the GPU in everything. But, once N crosses the 500 mark, the performance shifts in favour of GPU because the multMatrix function simply takes too long to execute. Additionally, the time taken at values of N higher than 2000 is immense. Figure 12 shows the exponential growth that happens at very high values of N[1].

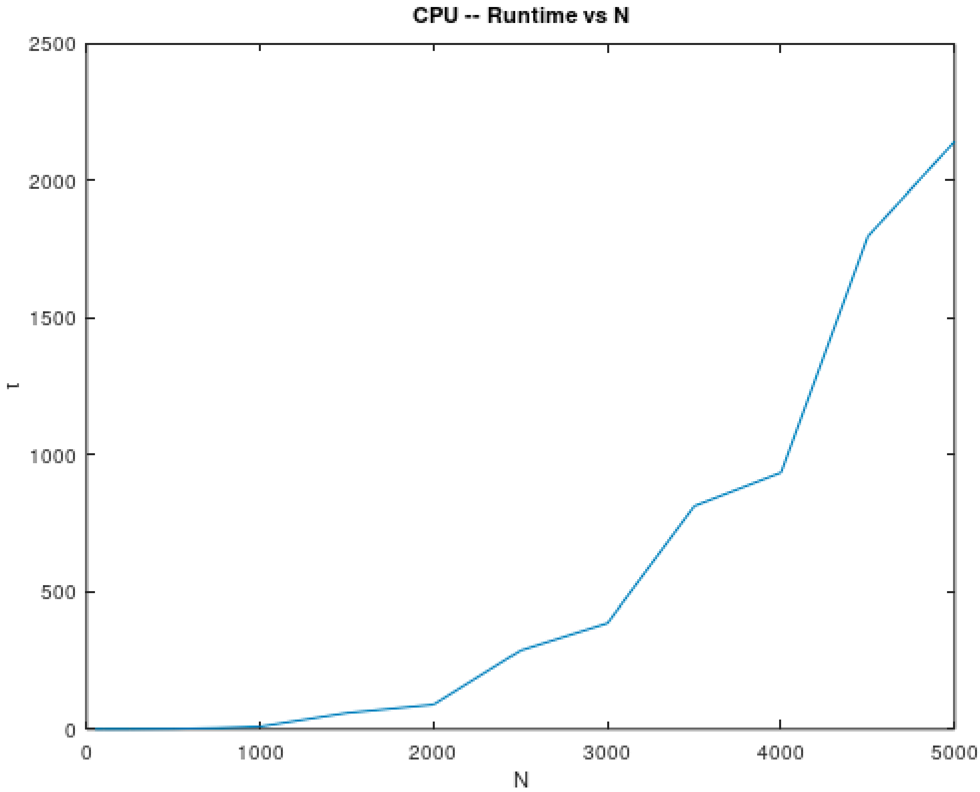



Figure 12: CPU – Runtime vs N

[1] Graph was plotted using Matlab at N = 50, 100, 250, 500, 750, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000.

# 4 GPU Version of The Program

In this section, the GPU version of the program will be explained function-by-function. Additionally, runtime and profiling information will observed at different values of N.

## 4.1 Program Code

The program is mostly the same in many areas to its CPU counterpart—same `csvMatrix` and `populateMatrix`. But, the main difference is in how the GPU handles addition and multiplication, which will be explained later.

```
#include <cuda_runtime.h>
```

Code Snippet 8: Included Cuda Library

An addition to the code is determining the operation used based on 'enums' defined to ease code readability and simplify the used logic.

```
// available matrix operations
typedef enum
{
    ADD = 1,
    MUL = 2
} matOp;
```

Code Snippet 9: Defined Operations as Enums

The main change in the main function implementation is an entirely rewritten function to perform either addition of multiplication based on the passed enum operation. This function contains all the preparations for the GPU kernel functions for addition and multiplication.

```
    // C * ((A * B) + (B * A))
    matrixOperationCudaWrapper(matA, matB, matRes, MUL); // A * B = res

    matrixOperationCudaWrapper(matB, matA, matTemp, MUL); // B * A = temp

    matrixOperationCudaWrapper(matRes, matTemp, matTemp, ADD); // res + temp = temp

    matrixOperationCudaWrapper(matC, matTemp, matRes, MUL); // C * temp = res
```

Code Snippet 10: Main Function Implementation

The CUDA wrapper function takes two 2D arrays, one 2D array to store the result, and an enum that tells the function which operation to execute.

```
void matrixOperationCudaWrapper(const unsigned int (&h_matrixA)[N][N], const unsigned int (&h_matrixB)[N][N], unsigned int (&h_matrixRes)[N][N], unsigned char operation)
```

Code Snippet 11: Matrix Operation Wrapper Function

The first thing this function does is to define pointers that will be used to store memory addresses.

```
    // create pointers to gpu
    unsigned int* d_cudaA = 0;
    unsigned int* d_cudaB = 0;
    unsigned int* d_cudaRes = 0;
```

Code Snippet 12: Pointers to GPU Memory

```
    // defining size
    size_t sizeInBytes = N * N * sizeof(unsigned int);
```

Code Snippet 13: Required GPU Memory Size

There are two ways to allocate GPU memory using CUDA: cudaMalloc and cudaMallocManaged. cudaMalloc was ultimately chosen because it was easier to understand its syntax and the benefits cudaMallocManaged brought was too advanced to understand.

```
    // allocate memory in gpu
    cudaMalloc((void**)(&d_cudaA), sizeInBytes);
    cudaMalloc((void**)(&d_cudaB), sizeInBytes);
    cudaMalloc((void**)(&d_cudaRes), sizeInBytes);
```

Code Snippet 14: GPU Memory Allocation

Here, the passed matrices to the function are copied to the allocated GPU memory. This copying process represents a huge overhead in the program and can be optimized by using cudaMallocManaged which eliminates the need to use cudaMemcpy completely.

```
    // copy vectors into gpu
    cudaMemcpy(d_cudaA, h_matrixA, sizeInBytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_cudaB, h_matrixB, sizeInBytes, cudaMemcpyHostToDevice);
```

Code Snippet 15: Copying Data From CPU Memory to GPU Memory

The number of threads and block size are chosen based on the GPU's hardware. Through trial and error, the best performance was achieved at a thread count of 32 and a block size of 128. It is important to keep both numbers a power of two to optimize performance. The type dim3 outputs The vector thread indices (X, Y, Z) suitable for the defined block size and thread count; this will be used in the kernel function.

```
    // defining threads and block size
    int threads = 32;
    int blocks = 128;

    // setting up kernel launch parameters
    dim3 BLOCKS(blocks, blocks);
    dim3 THREADS(threads, threads);
```

Code Snippet 16: Choosing Number of Threads and Block Size

Depending on the chosen operation, the corresponding kernel is launched by this obscurely complex syntax.

```
    // launch kernel for chosen operation
    if (operation == ADD)
        matrixAddCUDA<<<BLOCKS, THREADS>>>(d_cudaA, d_cudaB, d_cudaRes);
    else if (operation == MUL)
        matrixMulCUDA<<<BLOCKS, THREADS>>>(d_cudaA, d_cudaB, d_cudaRes);
    else
    {
        cout << "chotto matte!" << endl;
        return; // do not continue this mess!
    }
```

Code Snippet 17: Launching Appropriate Kernel Based on Passed Operation

The kernel function for addition is fairly simple. The vector thread indices will be used to index the memory locations to perform summation. The tricky part is to index a 2D array correctly using 1D array indexing. It is important to make sure that a thread operates within the array bounds.

```
__global__ void matrixAddCUDA(unsigned int* matrixA, unsigned int*
   matrixB, unsigned int* matrixRes)
{
    // Compute each thread's global row and column index
    int rowIndex = blockIdx.y * blockDim.y + threadIdx.y;
    int colIndex = blockIdx.x * blockDim.x + threadIdx.x;

    if (rowIndex < N && colIndex < N)
    {
        // simply add
        matrixRes[rowIndex * N + colIndex] = matrixA[rowIndex * N +
           colIndex] + matrixB[rowIndex * N + colIndex];
    }
}
```

Code Snippet 18: Matrix Addition CUDA Kernel Function

The kernel function for multiplication is a bit more complex. Like the previous function, it uses the vector thread indices to correctly index the arrays and perform a boundary check before executing any operation. It is important to initialize the result's element to 0 before accumulating the results of multiplying A and B.

```
__global__ void matrixMulCUDA(unsigned int* matrixA, unsigned int*
   matrixB, unsigned int* matrixRes)
{
    // Compute each thread's global row and column index
    int rowIndex = blockIdx.y * blockDim.y + threadIdx.y;
    int colIndex = blockIdx.x * blockDim.x + threadIdx.x;

    // Iterate over row, and down column
    if (rowIndex < N && colIndex < N)
    {
        matrixRes[rowIndex * N + colIndex] = 0;
        for (int k = 0; k < N; k++)
        {
            // Accumulate results for a single element
            matrixRes[rowIndex * N + colIndex] += matrixA[rowIndex * N +
               k] * matrixB[k * N + colIndex];
        }
    }
}
```

Code Snippet 19: Matrix Multiplication CUDA Kernel Function

After the operation is done, the results are copied from GPU memory to CPU memory.

```
    // copy result from gpu memory
    cudaMemcpy(h_matrixRes, d_cudaRes, sizeInBytes,
       cudaMemcpyDeviceToHost);
```

Code Snippet 20: Copying Data From GPU Memory to CPU Memory

Finally, free all the allocated GPU memory to avoid any memory leaks.

```
    // free allocated gpu memory
    cudaFree(d_cudaA);
    cudaFree(d_cudaB);
    cudaFree(d_cudaRes);
```

Code Snippet 21: Free Allocated GPU Memory

## 4.2 Runtime & Profiling Results

In this section, runtime and profiling information will be observed for 6 values of N at two thread counts (16 and 32): 50, 100, 500, 1000, 2000, and 5000.

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==4300== NVPROF is profiling process 4300, command: ./cuda
==4300== Profiling application: ./cuda
==4300== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   67.05%  43.170us         3  14.390us  13.888us  15.105us  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                   17.89%  11.521us         8  1.4400us  1.4080us  1.5040us  [CUDA memcpy HtoD]
                   11.48%  7.3920us         4  1.8480us  1.7600us  2.0800us  [CUDA memcpy DtoH]
                    3.58%  2.3040us         1  2.3040us  2.3040us  2.3040us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   96.66%  54.205ms        12  4.5171ms  1.5420us  53.973ms  cudaMalloc
                    1.74%  976.11us         4  244.03us  6.6380us  953.73us  cudaLaunchKernel
                    0.99%  555.81us        12  46.317us  4.9020us  409.69us  cudaMemcpy
                    0.36%  201.28us        12  16.773us  1.7180us  65.051us  cudaFree
                    0.20%  113.73us       114     997ns      73ns  47.850us  cuDeviceGetAttribute
                    0.03%  14.160us         1  14.160us  14.160us  14.160us  cuDeviceGetName
                    0.01%  6.6100us         1  6.6100us  6.6100us  6.6100us  cuDeviceGetPCIBusId
                    0.00%  1.0710us         3     357ns     104ns     804ns  cuDeviceGetCount
                    0.00%     457ns         2     228ns     110ns     347ns  cuDeviceGet
                    0.00%     454ns         1     454ns     454ns     454ns  cuDeviceTotalMem
                    0.00%     230ns         1     230ns     230ns     230ns  cuModuleGetLoadingMode
                    0.00%     229ns         1     229ns     229ns     229ns  cuDeviceGetUuid

real    0m0.400s
user    0m0.022s
sys     0m0.102s
```

Figure 13: Program Timing Information at N = 50 (Thread Count of16)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==4480== NVPROF is profiling process 4480, command: ./cuda
==4480== Profiling application: ./cuda
==4480== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   58.92%  86.370us         3  28.790us  28.160us  29.697us  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                   27.90%  40.896us         8  5.1120us  5.0880us  5.1520us  [CUDA memcpy HtoD]
                   11.48%  16.832us         4  4.2080us  4.0000us  4.4160us  [CUDA memcpy DtoH]
                    1.70%  2.4960us         1  2.4960us  2.4960us  2.4960us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   95.34%  44.635ms        12  3.7196ms  1.5880us  44.404ms  cudaMalloc
                    3.18%  1.4873ms       114  13.046us     117ns  1.3712ms  cuDeviceGetAttribute
                    0.73%  341.18us        12  28.431us  10.714us  99.393us  cudaMemcpy
                    0.42%  198.05us        12  16.504us  1.8380us  62.790us  cudaFree
                    0.25%  119.03us         4  29.757us  6.5100us  96.605us  cudaLaunchKernel
                    0.05%  22.120us         1  22.120us  22.120us  22.120us  cuDeviceGetName
                    0.02%  9.4010us         1  9.4010us  9.4010us  9.4010us  cuDeviceGetPCIBusId
                    0.00%  1.3190us         3     439ns     184ns     950ns  cuDeviceGetCount
                    0.00%     706ns         2     353ns     159ns     547ns  cuDeviceGet
                    0.00%     507ns         1     507ns     507ns     507ns  cuDeviceTotalMem
                    0.00%     388ns         1     388ns     388ns     388ns  cuModuleGetLoadingMode
                    0.00%     316ns         1     316ns     316ns     316ns  cuDeviceGetUuid

real    0m0.221s
user    0m0.025s
sys     0m0.076s
```

Figure 14: Program Timing Information at N = 100 (Thread Count of 16)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==4665== NVPROF is profiling process 4665, command: ./cuda
==4665== Profiling application: ./cuda
==4665== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   88.82%  7.7082ms         3  2.5694ms  2.5515ms  2.5835ms  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                    7.42%  644.11us         8  80.514us  79.746us  82.306us  [CUDA memcpy HtoD]
                    3.66%  317.32us         4  79.329us  79.138us  79.650us  [CUDA memcpy DtoH]
                    0.11%  9.1840us         1  9.1840us  9.1840us  9.1840us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   78.60%  43.353ms        12  3.6127ms  43.213us  42.774ms  cudaMalloc
                   19.16%  10.569ms        12  880.75us  102.58us  3.0784ms  cudaMemcpy
                    1.70%  935.99us        12  77.998us  44.118us  89.788us  cudaFree
                    0.30%  163.62us         4  40.904us  8.4900us  124.50us  cudaLaunchKernel
                    0.20%  112.63us       114     987ns      71ns  47.385us  cuDeviceGetAttribute
                    0.02%  13.199us         1  13.199us  13.199us  13.199us  cuDeviceGetName
                    0.01%  7.0780us         1  7.0780us  7.0780us  7.0780us  cuDeviceGetPCIBusId
                    0.00%  1.3600us         3     453ns     110ns  1.1350us  cuDeviceGetCount
                    0.00%     531ns         2     265ns     140ns     391ns  cuDeviceGet
                    0.00%     444ns         1     444ns     444ns     444ns  cuDeviceTotalMem
                    0.00%     231ns         1     231ns     231ns     231ns  cuDeviceGetUuid
                    0.00%     218ns         1     218ns     218ns     218ns  cuModuleGetLoadingMode

real    0m0.365s
user    0m0.088s
sys     0m0.074s
```

Figure 15: Program Timing Information at N = 500 (Thread Count of 16)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==4841== NVPROF is profiling process 4841, command: ./cuda
==4841== Profiling application: ./cuda
==4841== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   87.77%  53.223ms         3  17.741ms  17.603ms  17.998ms  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                    7.60%  4.6064ms         8  575.79us  539.09us  682.45us  [CUDA memcpy HtoD]
                    4.55%  2.7614ms         4  690.35us  596.94us  797.46us  [CUDA memcpy DtoH]
                    0.08%  48.065us         1  48.065us  48.065us  48.065us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   54.82%  62.542ms        12  5.2119ms  599.07us  18.908ms  cudaMemcpy
                   42.50%  48.491ms        12  4.0409ms  43.158us  47.327ms  cudaMalloc
                    2.32%  2.6482ms        12  220.68us  80.914us  281.63us  cudaFree
                    0.20%  228.79us         4  57.197us  26.745us  138.31us  cudaLaunchKernel
                    0.13%  151.14us       114  1.3250us     111ns  65.033us  cuDeviceGetAttribute
                    0.01%  14.995us         1  14.995us  14.995us  14.995us  cuDeviceGetName
                    0.01%  7.3960us         1  7.3960us  7.3960us  7.3960us  cuDeviceGetPCIBusId
                    0.00%  1.2520us         3     417ns     152ns     867ns  cuDeviceGetCount
                    0.00%     777ns         2     388ns     138ns     639ns  cuDeviceGet
                    0.00%     576ns         1     576ns     576ns     576ns  cuDeviceTotalMem
                    0.00%     383ns         1     383ns     383ns     383ns  cuModuleGetLoadingMode
                    0.00%     280ns         1     280ns     280ns     280ns  cuDeviceGetUuid

real    0m0.601s
user    0m0.309s
sys     0m0.087s
```

Figure 16: Program Timing Information at N = 1000 (Thread Count of 16)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==5006== NVPROF is profiling process 5006, command: ./cuda
==5006== Profiling application: ./cuda
==5006== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   86.68%  218.70ms         3  72.900ms  70.105ms  78.336ms  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                    8.07%  20.361ms         8  2.5451ms  2.4427ms  2.7509ms  [CUDA memcpy HtoD]
                    5.18%  13.066ms         4  3.2664ms  2.5716ms  4.0852ms  [CUDA memcpy DtoH]
                    0.07%  185.51us         1  185.51us  185.51us  185.51us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   83.99%  254.01ms        12  21.168ms  2.5143ms  82.581ms  cudaMemcpy
                   13.38%  40.452ms        12  3.3710ms  45.970us  37.421ms  cudaMalloc
                    2.51%  7.5998ms        12  633.32us  114.24us  900.01us  cudaFree
                    0.08%  255.89us         4  63.972us  31.279us  149.41us  cudaLaunchKernel
                    0.03%  98.207us       114     861ns      72ns  41.795us  cuDeviceGetAttribute
                    0.00%  12.147us         1  12.147us  12.147us  12.147us  cuDeviceGetName
                    0.00%  6.8830us         1  6.8830us  6.8830us  6.8830us  cuDeviceGetPCIBusId
                    0.00%     792ns         3     264ns     104ns     559ns  cuDeviceGetCount
                    0.00%     459ns         2     229ns      74ns     385ns  cuDeviceGet
                    0.00%     380ns         1     380ns     380ns     380ns  cuDeviceTotalMem
                    0.00%     245ns         1     245ns     245ns     245ns  cuDeviceGetUuid
                    0.00%     233ns         1     233ns     233ns     233ns  cuModuleGetLoadingMode

real    0m1.559s
user    0m1.146s
sys     0m0.110s
```

Figure 17: Program Timing Information at N = 2000 (Thread Count of 16)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==5236== NVPROF is profiling process 5236, command: ./cuda
==5236== Profiling application: ./cuda
==5236== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   96.67%  5.93430s         3  1.97810s  1.97639s  1.98026s  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                    2.03%  124.76ms         8  15.595ms  15.224ms  16.522ms  [CUDA memcpy HtoD]
                    1.28%  78.686ms         4  19.672ms  15.681ms  24.180ms  [CUDA memcpy DtoH]
                    0.02%  1.1758ms         1  1.1758ms  1.1758ms  1.1758ms  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   98.37%  6.14064s        12  511.72ms  15.333ms  2.00212s  cudaMemcpy
                    1.07%  66.864ms        12  5.5720ms  64.555us  44.219ms  cudaMalloc
                    0.55%  34.605ms        12  2.8837ms  137.46us  4.2751ms  cudaFree
                    0.00%  286.64us         4  71.659us  27.984us  194.10us  cudaLaunchKernel
                    0.00%  115.26us       114  1.0110us      75ns  49.556us  cuDeviceGetAttribute
                    0.00%  11.585us         1  11.585us  11.585us  11.585us  cuDeviceGetName
                    0.00%  6.6900us         1  6.6900us  6.6900us  6.6900us  cuDeviceGetPCIBusId
                    0.00%     843ns         3     281ns     146ns     532ns  cuDeviceGetCount
                    0.00%     531ns         1     531ns     531ns     531ns  cuDeviceTotalMem
                    0.00%     456ns         2     228ns      90ns     366ns  cuDeviceGet
                    0.00%     228ns         1     228ns     228ns     228ns  cuDeviceGetUuid
                    0.00%     225ns         1     225ns     225ns     225ns  cuModuleGetLoadingMode

real    0m12.674s
user    0m11.270s
sys     0m0.445s
```

Figure 18: Program Timing Information at N = 5000 (Thread Count of 16)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==18316== NVPROF is profiling process 18316, command: ./cuda
==18316== Profiling application: ./cuda
==18316== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   73.87%  876.12us         3  292.04us  291.37us  292.87us  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                   24.52%  290.76us         1  290.76us  290.76us  290.76us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
                    0.97%  11.520us         8  1.4400us  1.4080us  1.4720us  [CUDA memcpy HtoD]
                    0.64%  7.5530us         4  1.8880us  1.7600us  2.2400us  [CUDA memcpy DtoH]
      API calls:   96.24%  44.556ms        12  3.7130ms  1.7090us  44.323ms  cudaMalloc
                    2.80%  1.2941ms        12  107.84us  7.0840us  308.03us  cudaMemcpy
                    0.43%  201.24us        12  16.769us  1.6970us  63.574us  cudaFree
                    0.25%  114.61us         4  28.651us  6.7540us  91.930us  cudaLaunchKernel
                    0.24%  109.68us       114     962ns      75ns  46.394us  cuDeviceGetAttribute
                    0.02%  11.510us         1  11.510us  11.510us  11.510us  cuDeviceGetName
                    0.01%  6.8040us         1  6.8040us  6.8040us  6.8040us  cuDeviceGetPCIBusId
                    0.00%     935ns         3     311ns     116ns     651ns  cuDeviceGetCount
                    0.00%     519ns         2     259ns     134ns     385ns  cuDeviceGet
                    0.00%     473ns         1     473ns     473ns     473ns  cuDeviceTotalMem
                    0.00%     296ns         1     296ns     296ns     296ns  cuModuleGetLoadingMode
                    0.00%     142ns         1     142ns     142ns     142ns  cuDeviceGetUuid

real    0m0.230s
user    0m0.023s
sys     0m0.088s
```

Figure 19: Program Timing Information at N = 50 (Thread Count of 32)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==18501== NVPROF is profiling process 18501, command: ./cuda
==18501== Profiling application: ./cuda
==18501== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   72.24%  906.84us         3  302.28us  301.38us  302.79us  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                   23.17%  290.89us         1  290.89us  290.89us  290.89us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
                    3.26%  40.962us         8  5.1200us  5.0890us  5.1840us  [CUDA memcpy HtoD]
                    1.32%  16.578us         4  4.1440us  4.0320us  4.3530us  [CUDA memcpy DtoH]
      API calls:   96.10%  45.939ms        12  3.8282ms  1.7320us  45.707ms  cudaMalloc
                    2.94%  1.4074ms        12  117.28us  10.368us  332.62us  cudaMemcpy
                    0.44%  209.19us        12  17.432us  1.8500us  64.018us  cudaFree
                    0.25%  119.05us         4  29.763us  7.0810us  95.293us  cudaLaunchKernel
                    0.23%  110.01us       114     964ns      69ns  47.447us  cuDeviceGetAttribute
                    0.02%  8.8700us         1  8.8700us  8.8700us  8.8700us  cuDeviceGetName
                    0.01%  7.1220us         1  7.1220us  7.1220us  7.1220us  cuDeviceGetPCIBusId
                    0.00%     865ns         3     288ns      99ns     645ns  cuDeviceGetCount
                    0.00%     545ns         2     272ns      87ns     458ns  cuDeviceGet
                    0.00%     489ns         1     489ns     489ns     489ns  cuDeviceTotalMem
                    0.00%     238ns         1     238ns     238ns     238ns  cuModuleGetLoadingMode
                    0.00%     215ns         1     215ns     215ns     215ns  cuDeviceGetUuid

real    0m0.221s
user    0m0.022s
sys     0m0.077s
```

Figure 20: Program Timing Information at N = 100 (Thread Count of 32)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==19080== NVPROF is profiling process 19080, command: ./cuda
==19080== Profiling application: ./cuda
==19080== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   82.04%  5.7252ms         3  1.9084ms  1.8435ms  2.0255ms  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                    9.20%  641.91us         8  80.238us  80.066us  80.514us  [CUDA memcpy HtoD]
                    4.54%  316.71us         4  79.177us  79.106us  79.265us  [CUDA memcpy DtoH]
                    4.23%  294.92us         1  294.92us  294.92us  294.92us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   82.09%  45.365ms        12  3.7804ms  42.782us  44.789ms  cudaMalloc
                   15.73%  8.6945ms        12  724.54us  71.539us  2.3589ms  cudaMemcpy
                    1.68%  926.20us        12  77.183us  44.960us  93.592us  cudaFree
                    0.27%  147.93us         4  36.981us  8.0840us  121.23us  cudaLaunchKernel
                    0.20%  110.55us       114     969ns      72ns  47.870us  cuDeviceGetAttribute
                    0.02%  9.8620us         1  9.8620us  9.8620us  9.8620us  cuDeviceGetName
                    0.01%  7.2150us         1  7.2150us  7.2150us  7.2150us  cuDeviceGetPCIBusId
                    0.00%     762ns         3     254ns     102ns     525ns  cuDeviceGetCount
                    0.00%     485ns         2     242ns     107ns     378ns  cuDeviceGet
                    0.00%     449ns         1     449ns     449ns     449ns  cuDeviceTotalMem
                    0.00%     244ns         1     244ns     244ns     244ns  cuModuleGetLoadingMode
                    0.00%     221ns         1     221ns     221ns     221ns  cuDeviceGetUuid

real    0m0.374s
user    0m0.088s
sys     0m0.086s
```

Figure 21: Program Timing Information at N = 500 (Thread Count of 32)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==19505== NVPROF is profiling process 19505, command: ./cuda
==19505== Profiling application: ./cuda
==19505== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   81.91%  33.988ms         3  11.329ms  11.023ms  11.744ms  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                   10.83%  4.4933ms         8  561.67us  526.70us  658.29us  [CUDA memcpy HtoD]
                    6.51%  2.6998ms         4  674.95us  570.35us  810.71us  [CUDA memcpy DtoH]
                    0.76%  313.96us         1  313.96us  313.96us  313.96us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   50.23%  46.553ms        12  3.8794ms  44.931us  45.382ms  cudaMalloc
                   46.57%  43.159ms        12  3.5966ms  581.54us  12.812ms  cudaMemcpy
                    2.78%  2.5737ms        12  214.47us  59.938us  282.19us  cudaFree
                    0.21%  191.81us         4  47.953us  12.391us  125.52us  cudaLaunchKernel
                    0.18%  163.63us       114  1.4350us     117ns  64.025us  cuDeviceGetAttribute
                    0.02%  23.123us         1  23.123us  23.123us  23.123us  cuDeviceGetName
                    0.01%  10.452us         1  10.452us  10.452us  10.452us  cuDeviceGetPCIBusId
                    0.00%  1.6960us         3     565ns     173ns  1.3330us  cuDeviceGetCount
                    0.00%     698ns         2     349ns     164ns     534ns  cuDeviceGet
                    0.00%     569ns         1     569ns     569ns     569ns  cuDeviceTotalMem
                    0.00%     374ns         1     374ns     374ns     374ns  cuModuleGetLoadingMode
                    0.00%     311ns         1     311ns     311ns     311ns  cuDeviceGetUuid

real    0m0.584s
user    0m0.275s
sys     0m0.089s
```

Figure 22: Program Timing Information at N = 1000 (Thread Count of 32)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==19853== NVPROF is profiling process 19853, command: ./cuda
==19853== Profiling application: ./cuda
==19853== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   87.14%  218.54ms         3  72.846ms  70.569ms  75.371ms  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                    7.81%  19.577ms         8  2.4471ms  2.3605ms  2.7227ms  [CUDA memcpy HtoD]
                    4.91%  12.313ms         4  3.0783ms  2.4502ms  3.8287ms  [CUDA memcpy DtoH]
                    0.14%  347.72us         1  347.72us  347.72us  347.72us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   83.22%  252.46ms        12  21.038ms  2.4909ms  79.256ms  cudaMemcpy
                   14.11%  42.812ms        12  3.5677ms  46.080us  39.727ms  cudaMalloc
                    2.54%  7.6922ms        12  641.02us  104.44us  957.35us  cudaFree
                    0.09%  280.05us         4  70.013us  29.073us  186.00us  cudaLaunchKernel
                    0.03%  96.141us       114     843ns      77ns  41.132us  cuDeviceGetAttribute
                    0.00%  10.740us         1  10.740us  10.740us  10.740us  cuDeviceGetName
                    0.00%  7.2890us         1  7.2890us  7.2890us  7.2890us  cuDeviceGetPCIBusId
                    0.00%     886ns         3     295ns      94ns     645ns  cuDeviceGetCount
                    0.00%     503ns         1     503ns     503ns     503ns  cuDeviceTotalMem
                    0.00%     501ns         2     250ns     100ns     401ns  cuDeviceGet
                    0.00%     296ns         1     296ns     296ns     296ns  cuModuleGetLoadingMode
                    0.00%     233ns         1     233ns     233ns     233ns  cuDeviceGetUuid

real    0m1.518s
user    0m1.133s
sys     0m0.090s
```

Figure 23: Program Timing Information at N = 2000 (Thread Count of 32)

```
ComputerArchitecture_Playground/Lab_3 on  main [✘!?]
❯ time nvprof ./cuda
==20774== NVPROF is profiling process 20774, command: ./cuda
==20774== Profiling application: ./cuda
==20774== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   91.92%  2.34635s         3  782.12ms  770.43ms  793.02ms  matrixMulCUDA(unsigned int*, unsigned int*, unsigned int*)
                    4.87%  124.38ms         8  15.548ms  15.259ms  16.584ms  [CUDA memcpy HtoD]
                    3.18%  81.145ms         4  20.286ms  15.640ms  25.112ms  [CUDA memcpy DtoH]
                    0.03%  791.67us         1  791.67us  791.67us  791.67us  matrixAddCUDA(unsigned int*, unsigned int*, unsigned int*)
      API calls:   96.15%  2.55463s        12  212.89ms  15.326ms  818.46ms  cudaMemcpy
                    2.53%  67.259ms        12  5.6050ms  67.490us  44.622ms  cudaMalloc
                    1.30%  34.642ms        12  2.8868ms  141.28us  4.3188ms  cudaFree
                    0.01%  291.16us         4  72.788us  29.494us  194.73us  cudaLaunchKernel
                    0.00%  109.03us       114     956ns      74ns  46.894us  cuDeviceGetAttribute
                    0.00%  9.8600us         1  9.8600us  9.8600us  9.8600us  cuDeviceGetName
                    0.00%  6.7940us         1  6.7940us  6.7940us  6.7940us  cuDeviceGetPCIBusId
                    0.00%     990ns         3     330ns     101ns     700ns  cuDeviceGetCount
                    0.00%     499ns         2     249ns     106ns     393ns  cuDeviceGet
                    0.00%     465ns         1     465ns     465ns     465ns  cuDeviceTotalMem
                    0.00%     236ns         1     236ns     236ns     236ns  cuDeviceGetUuid
                    0.00%     224ns         1     224ns     224ns     224ns  cuModuleGetLoadingMode

real    0m8.916s
user    0m7.682s
sys     0m0.394s
```

Figure 24: Program Timing Information at N = 5000 (Thread Count of 32)

## 4.3 Observation

At low values of N, the GPU code is ridiculously slow: the amount of time taken to copy data between GPU and CPU memory outweighs any gains in execution time of actual operations. Once N passes the 500 mark, actual gains start to appear. At values of N higher than 2000, the performance difference is staggering and the memory overhead is out shined by the speedup obtained in the multiplication operation. Figures 25 and 26 show the runtime at different values of N[2] at thread count 16 and 32 respectively. It is noticed that the slope of the second graph is more forgiving and that the runtime is considerably lower at the higher thread count.

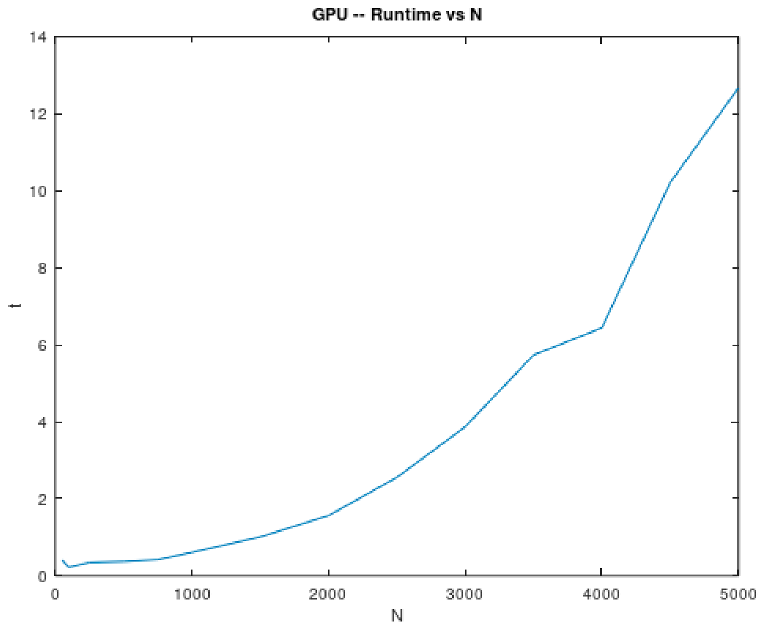



Figure 25: GPU – Runtime vs N (Thread Count of 16)

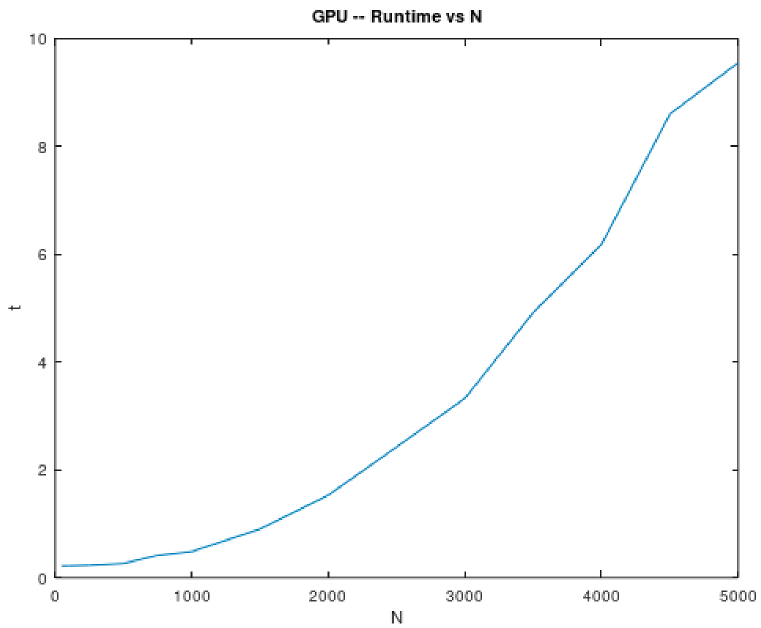



Figure 26: GPU – Runtime vs N (Thread Count of 32)

[2]Graph One and graph two were plotted using Matlab at N = 50, 100, 250, 500, 750, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000.