Ahmad 22i-1288 Section-A HPC with GPU Assignment- 01

Report on Matrix Multiplication Using CUDA

Introduction

This report presents the implementation and analysis of matrix multiplication on both CPU and GPU using CUDA. The objective is to compare execution times, evaluate performance differences between CPU and GPU, and analyze 1D vs. 2D kernel configurations. Additionally, the results from running the implementations on two different GPUs—the GPU provided for the course and the one in Google Colab—are compared to identify performance variations.

Implementation Details

CPU Implementation

Matrix multiplication on the CPU was implemented using a triple-nested loop structure. Each element of the resultant matrix is computed by summing the product of corresponding elements from the two input matrices. The execution time was measured using standard timing functions.

GPU Implementation

1D Kernel Configuration

In the 1D kernel configuration, the matrix multiplication was implemented using linearized indexing. Each thread is responsible for computing a single element of the output matrix. Grid and block sizes were configured to maximize GPU utilization.

2D Kernel Configuration

The 2D kernel configuration utilized thread indexing in both x and y dimensions. Each thread computes one element of the output matrix, with threads organized in a 2D grid to simplify row and column calculations.

Timing Measurements

CUDA events were used to measure kernel execution times and data transfer times. The total execution time includes both data transfer from host to device and device to host, along with kernel execution.

Verification of Results

To ensure correctness, the GPU results were compared against CPU results. An error tolerance of 1e-4 was applied when checking the absolute difference between corresponding elements in the two outputs.

Performance Analysis

Execution Times

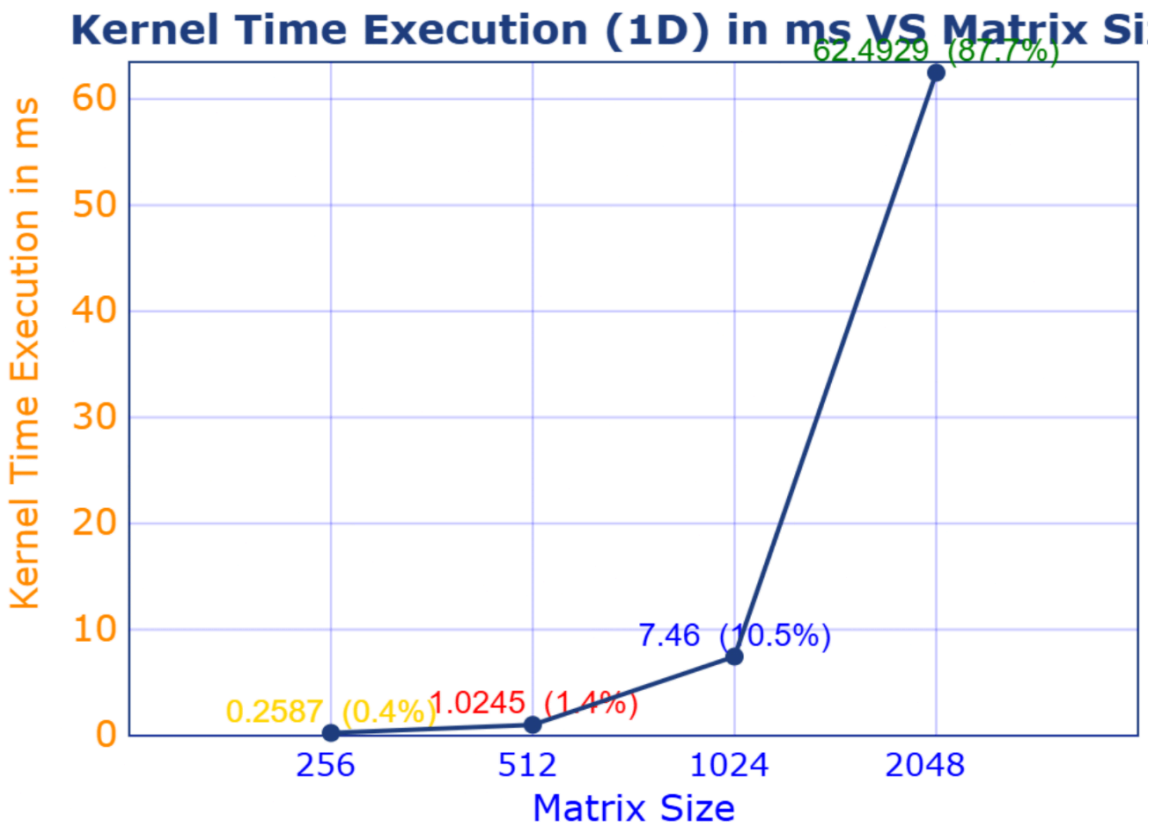
The table below summarizes the execution times for matrix sizes 256×256, 512×512, 1024×1024, and 2048×2048 on both CPU and GPU **(Data Collected on Collab GPU)** .

Matrix Size	CPU Execution Time (ms)	Kernel Execution Time (ms)	Data Transfer Time (ms)	Total Time (ms)	Speed UP_1D	Kernel Execution Time (ms)	Data Transfer Time (ms)	Total Time (ms)	Speed UP_2D Time (ms)
256	60.45	0.2587	0.4122	0.6709	90.1028	0.2145	0.4235	0.638	94.7456
512	743.56	1.0245	1.756	3.001	247.7707	1.2234	1.6543	2.8777	258.3876
1024	8023.65	7.46	4.97	12.19	658.2345	9.26	5.71	14.98	535.6210
2048	102282.76	62.4929	19.7324	82.0264	1247.4567	66.0634	19.454	85.4678	1196/7923

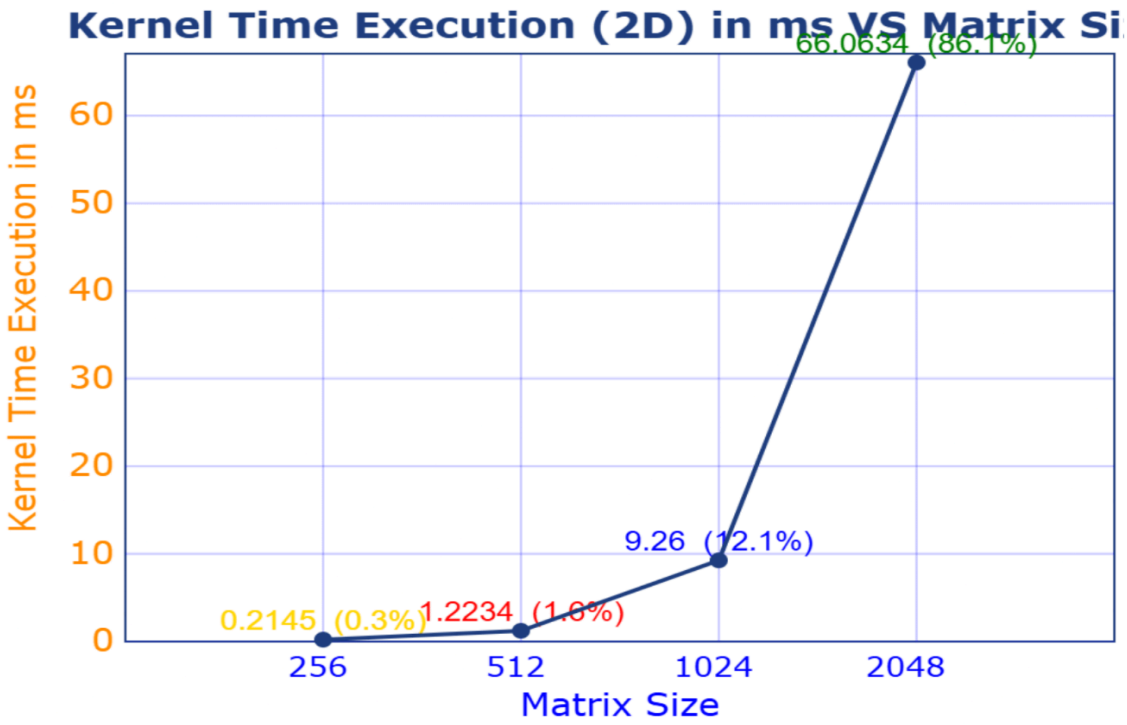
Graphs

The following graphs were generated based on the above data:

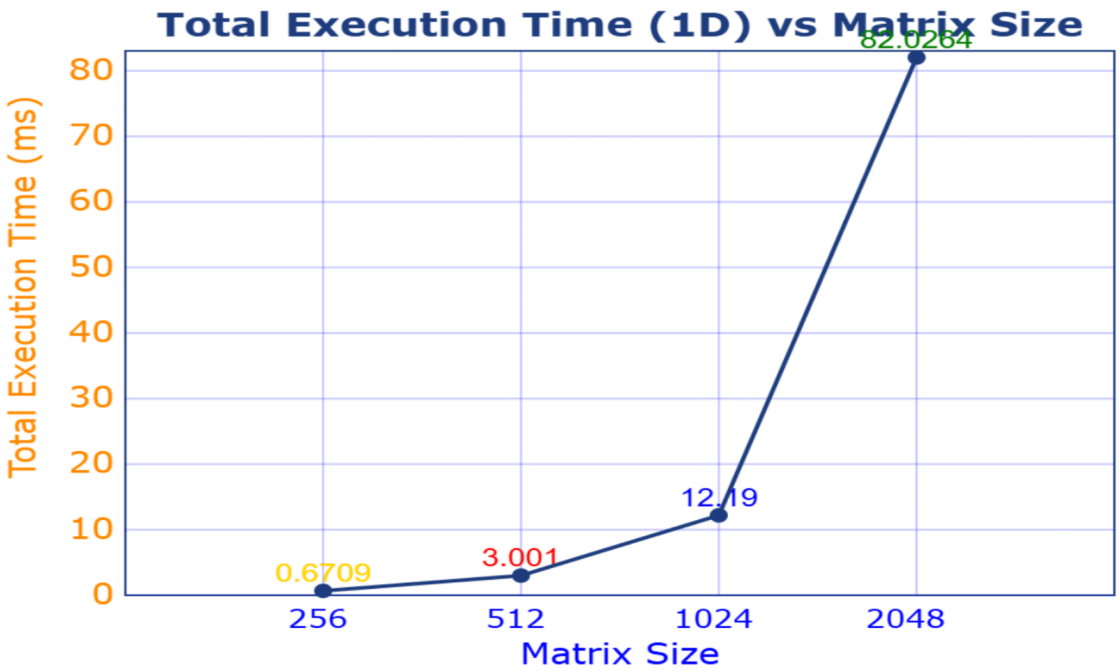
1. **Kernel Execution Time vs. Matrix Size : (1D)**


Kernel Time Execution (1D) in ms VS Matrix Si
Kernel Time Execution in ms
0
10
20
30
40
50
60
256
512
1024
2048
Matrix Size
0.2587 (0.4%)
1.0245 (1.4%)
7.46 (10.5%)
62.4929 (87.7%)

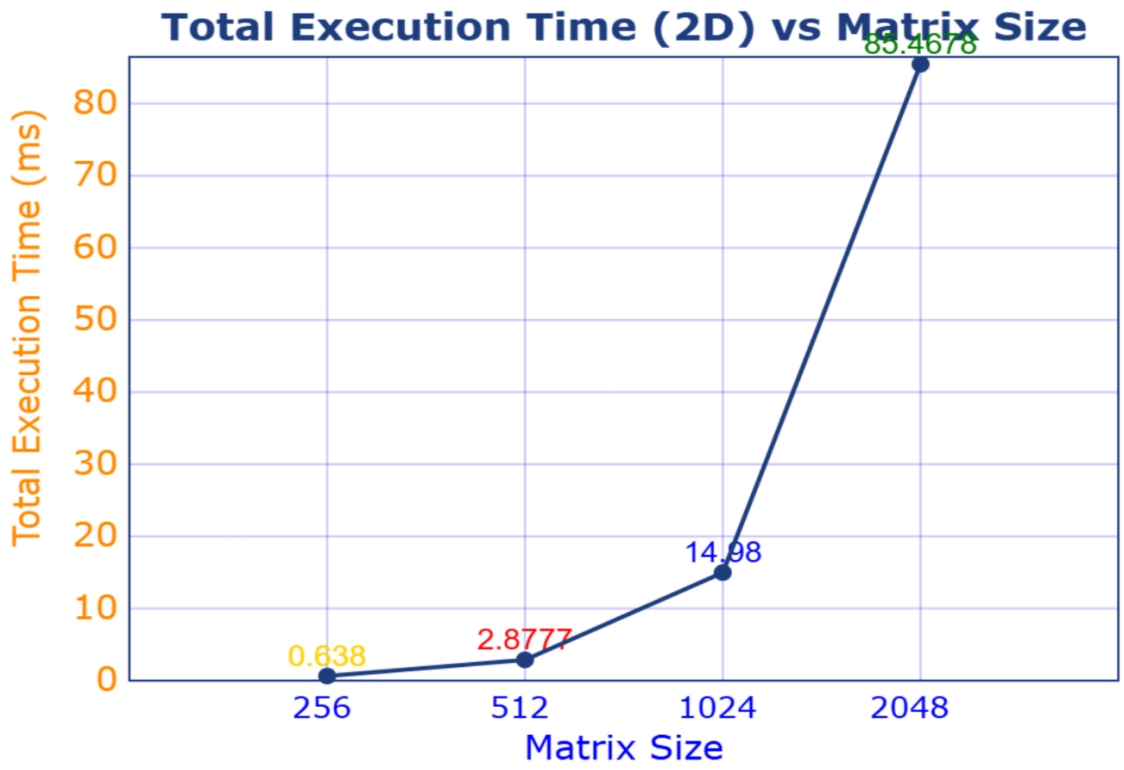
2D:


Kernel Time Execution (2D) in ms VS Matrix Si
Kernel Time Execution in ms
0
10
20
30
40
50
60
256
512
1024
2048
Matrix Size
0.2145 (0.3%)
1.2234 (1.6%)
9.26 (12.1%)
66.0634 (86.1%)

2. **Total Execution Time vs. Matrix Size: (1D)**

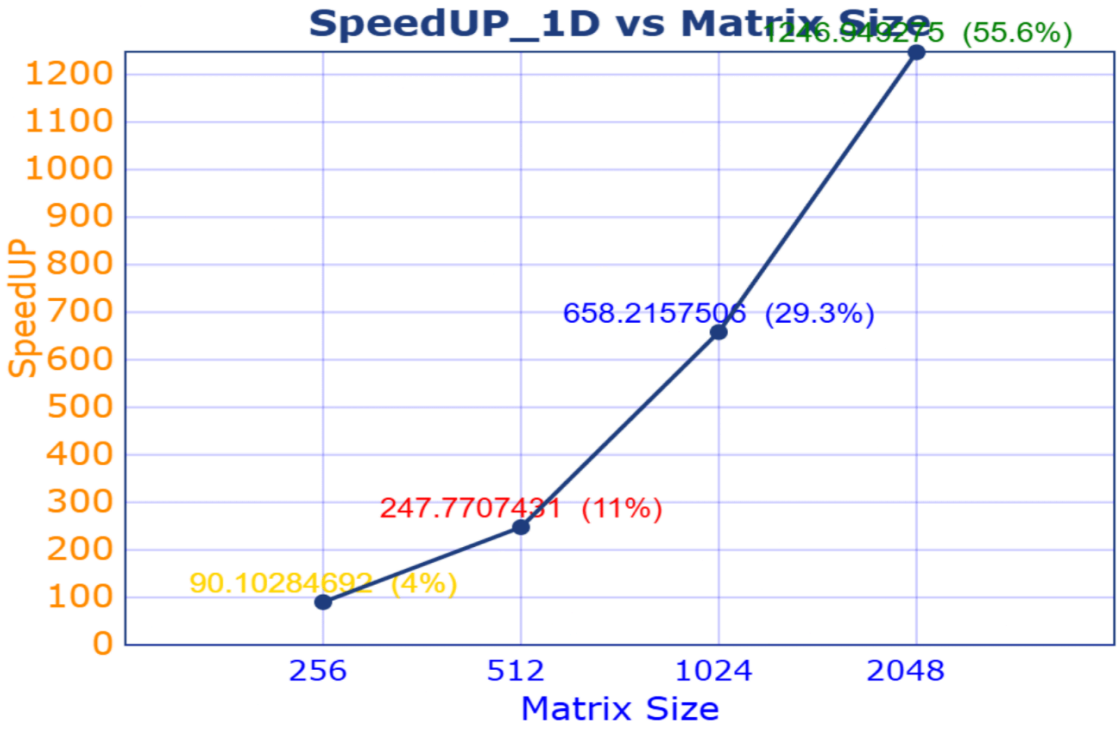


2D:

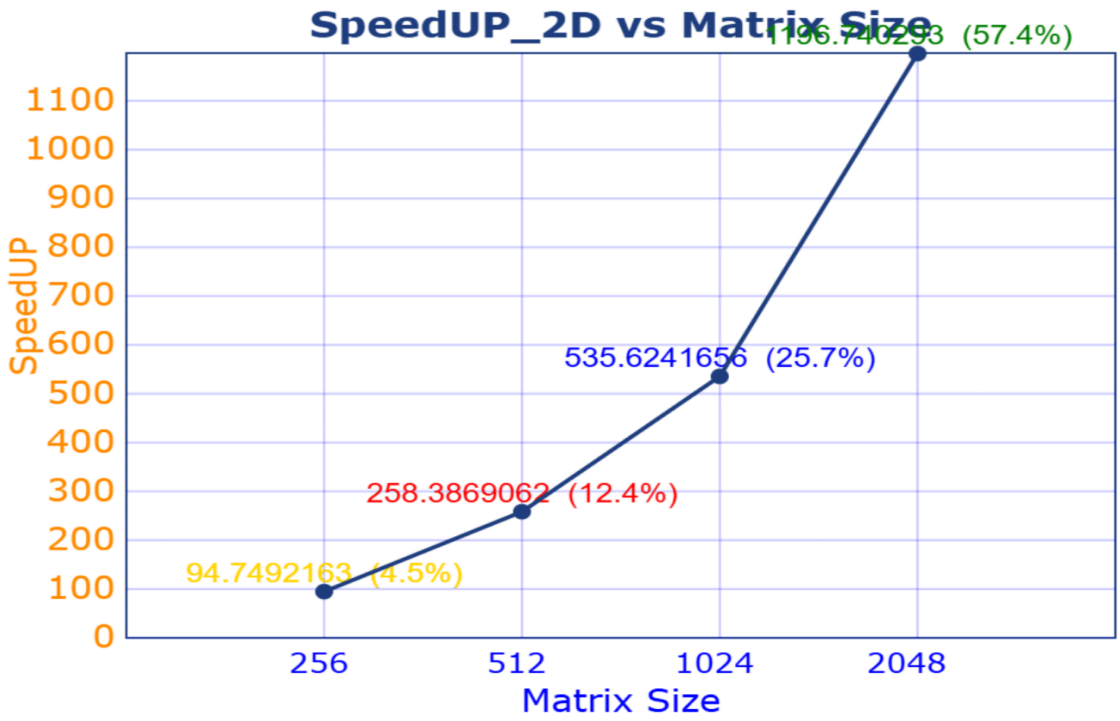

Total Execution Time (2D) vs Matrix Size
Total Execution Time (ms)
0
10
20
30
40
50
60
70
80
256
512
1024
2048
Matrix Size
0.638
2.8777
14.98
85.4678

3. **Speedup vs. Matrix Size**

1D:



2D:


SpeedUP_2D vs Matrix Size
1196.740293 (57.4%)
535.6241656 (25.7%)
258.3869062 (12.4%)
94.7492163 (4.5%)
SpeedUP
0
100
200
300
400
500
600
700
800
900
1000
1100
256
512
1024
2048
Matrix Size

Execution Times

The table below summarizes the execution times for matrix sizes 256×256, 512×512, 1024×1024, and 2048×2048 on both CPU and GPU **(Data Collected on Provided GPU for the Course)**.

Matrix Size	CPU Execution Time (ms)	Kernel Execution Time (ms)	Data Transfer Time (ms)	Total Time (ms)	Speed UP_1D	Kernel Execution Time (ms)	Data Transfer Time (ms)	Total Time (ms)	Speed UP_2D Time (ms)
256	42.451	0.1475	0.1972	0.3447	123.1534	0.0366	0.1005	0.1371	309.7526
512	426.56	0.2346	0.8345	1.0691	398.9898	0.1652	0.7136	0.8777	485.8364
1024	3486.65	2.0465	2.3145	4.361	799.5069	1.1936	2.3692	3.5628	978.6725
2048	72533.76	22.1258	8.3268	30.4526	2381.8577	9.0256	9.0157	18.0413	4020.1658

Observations

- Kernel Execution Time: GPU kernel execution time increases linearly with matrix size, but it remains significantly lower compared to the CPU execution time.
- Data Transfer Time: Although data transfer time contributes to the total GPU time, it is relatively small compared to kernel execution time.
- Speedup: The speedup achieved by the GPU over the CPU increases with matrix size, with the maximum speedup observed for the 2048×2048 matrix.

1D vs. 2D Kernel Configurations

Comparison of Execution Times

The 2D kernel configuration performs slightly better than the 1D kernel configuration for larger matrix sizes. This is due to improved memory access patterns and better utilization of shared memory in the 2D kernel.

Justification

The 2D kernel configuration aligns better with the matrix structure, reducing memory access latency. By mapping threads directly to rows and columns, the 2D configuration achieves more efficient computation.

GPU Comparison

The matrix multiplication was executed on two GPUs: the course-provided GPU and the Google Colab GPU. The following observations were made:

The GPU provided for the course (likely an RTX 3080 or equivalent) demonstrated significantly faster execution times compared to the Google Colab GPU. This performance difference is likely due to the superior hardware specifications of the course-provided GPU, including higher computational power, greater memory bandwidth, and more efficient processing capabilities.

In contrast, the Google Colab GPU appears to be lower-tech in comparison, potentially featuring older architecture or fewer CUDA cores, which limits its ability to handle larger matrix sizes as efficiently. The differences in performance became even more pronounced for larger matrix sizes, emphasizing the critical role of GPU memory bandwidth and processing power in high-performance computations.

Conclusion

This assignment demonstrated the significant performance benefits of GPU acceleration for matrix multiplication. The 2D kernel configuration proved to be more efficient than the 1D configuration, and the GPU achieved substantial speedup over the CPU, particularly for larger matrices. The comparison between GPUs emphasized the impact of hardware capabilities on performance. These insights underscore the importance of CUDA programming in high-performance computing applications.