High Performance Computer Architecture (E0 - 243)
Assignment 2 (Part - B)
MTech - CSA (IISC Bangalore)

Submitted By:

Anushka Pateriya (23130)

Sahil Lathiya (22691)

Part B

1. What is CUDA?

CUDA stands for Compute Unified Device Architecture, and it is a parallel computing platform and programming model developed by NVIDIA. CUDA allows developers to use NVIDIA GPUs (Graphics Processing Units) for general-purpose computing tasks, not just graphics processing. The primary goal of CUDA is to enable developers to harness the parallel processing power of GPUs to accelerate a wide range of applications.

Key features of CUDA:

- **Parallel Programming Model:**

  CUDA provides a parallel programming model that allows developers to write programs that can be executed in parallel on NVIDIA GPUs. This is achieved using a C-like programming language with extensions for parallelism.

- **GPU Acceleration:**

  CUDA allows developers to offload certain parts of their application's computation to the GPU, taking advantage of the massive parallel processing capabilities of modern GPUs. This is particularly useful for tasks that can be parallelized, such as matrix operations, simulations, and deep learning.

- **CUDA Toolkit:**

  NVIDIA provides a CUDA Toolkit, which includes a compiler, libraries, and development tools for building GPU-accelerated applications. The toolkit includes libraries for linear algebra, signal processing, and other common numerical tasks.

- **GPU-Accelerated Libraries:**

  CUDA supports GPU-accelerated libraries that provide optimized implementations of common algorithms for tasks like linear algebra, signal processing, and image processing.

- **Integration with Popular Languages:**

  While CUDA itself uses a C-like language, it also integrates with popular programming languages like C++, Python, and Fortran through language bindings and extensions.

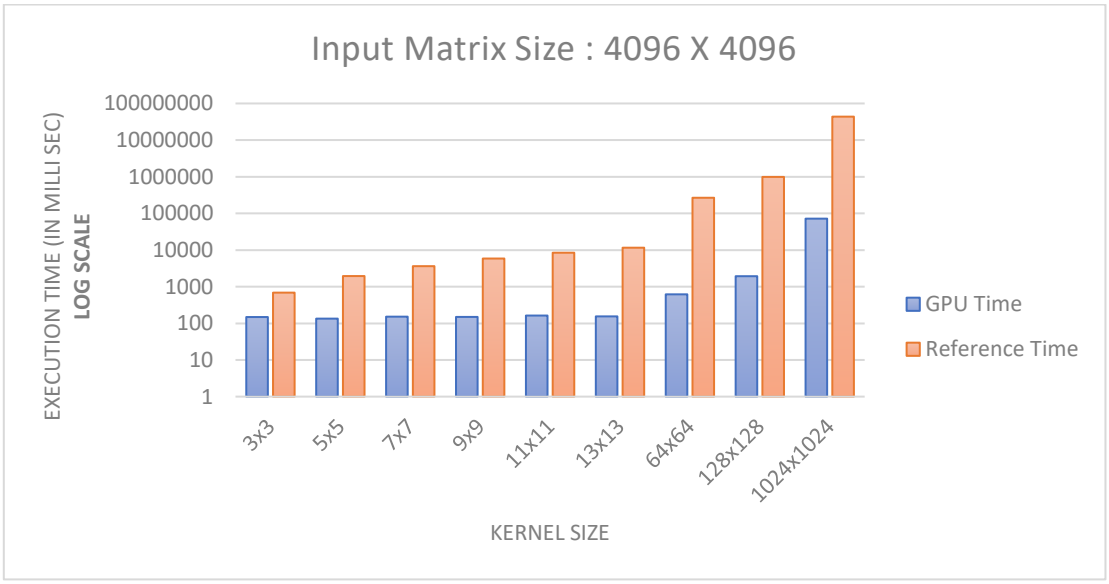
- **Parallelism and Scalability:**

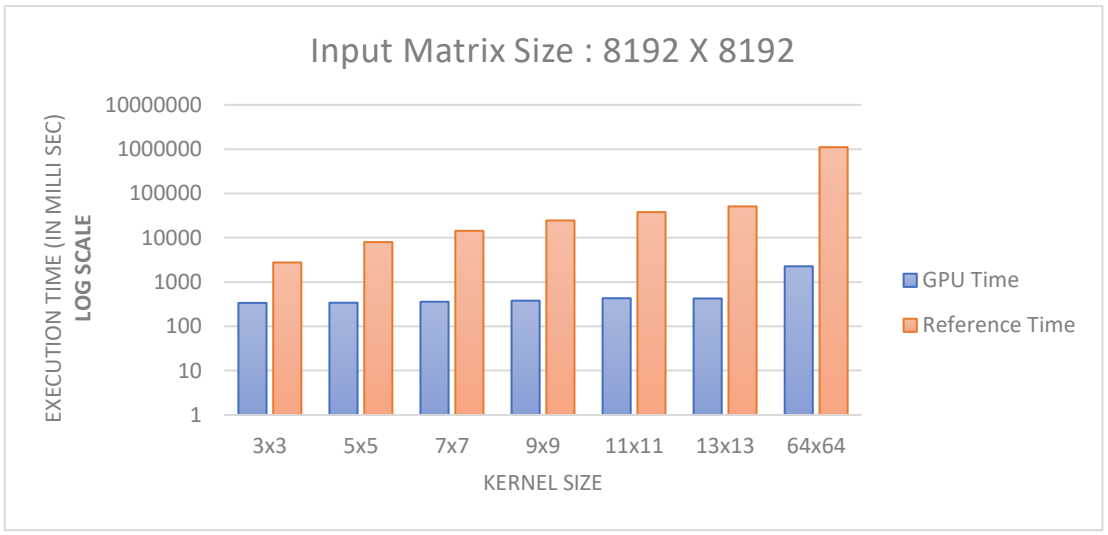
  CUDA allows developers to express both task parallelism and data parallelism, making it suitable for a wide range of parallel computing tasks. It also provides features for managing data transfer between the CPU and GPU.

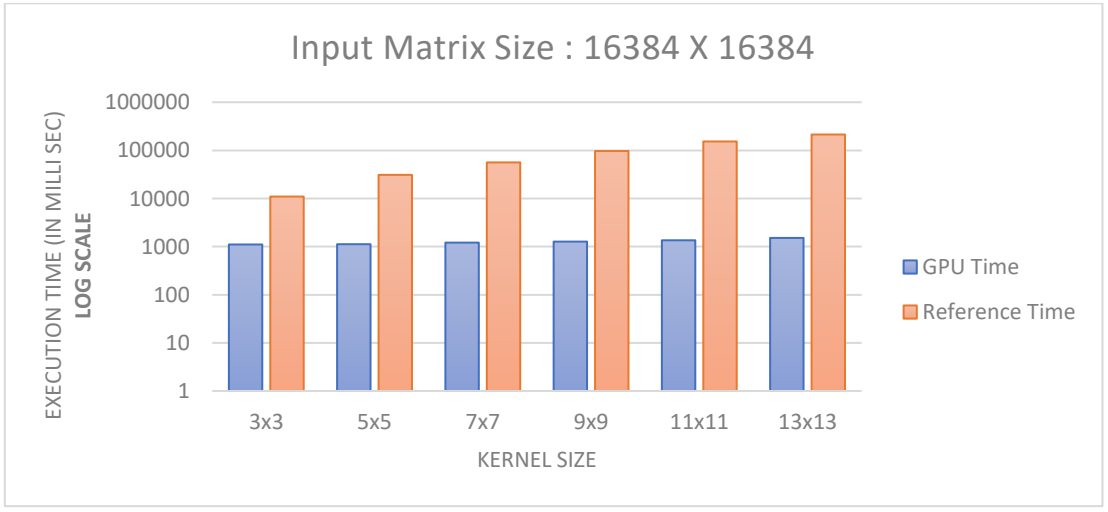
2. Observation with CUDA

Execution time of Reference (Naive) code VS Execution time of GPU

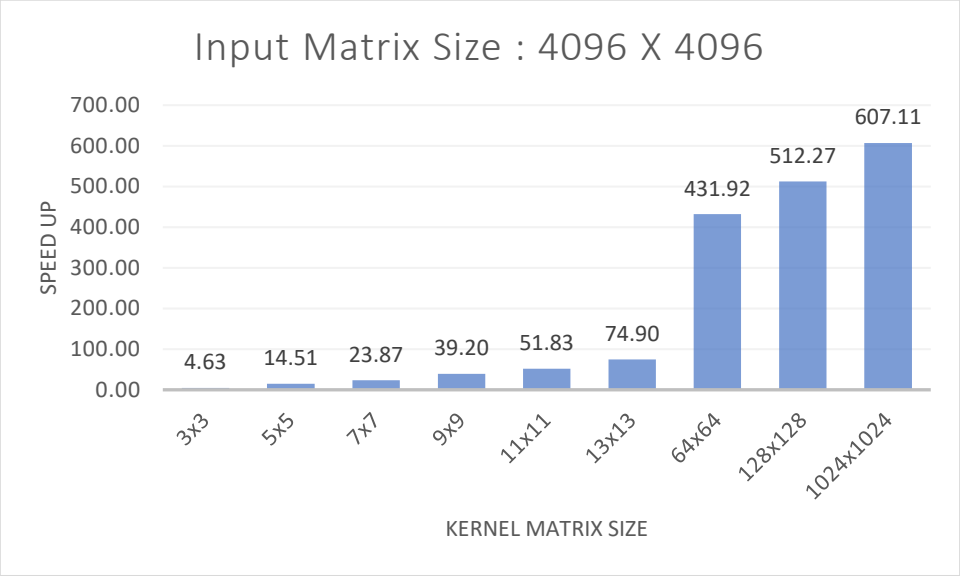
Note: In graphs log scale is used to represent analysis.

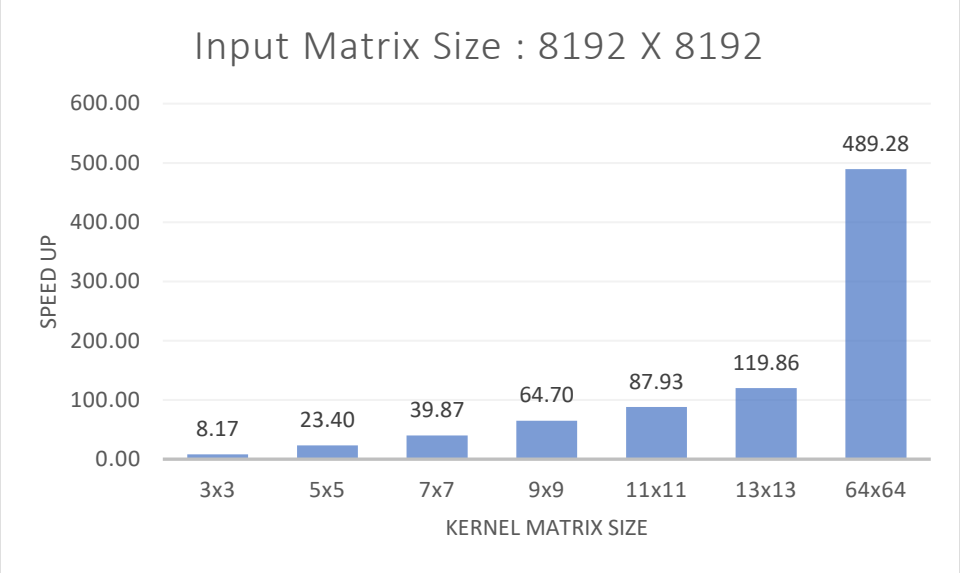


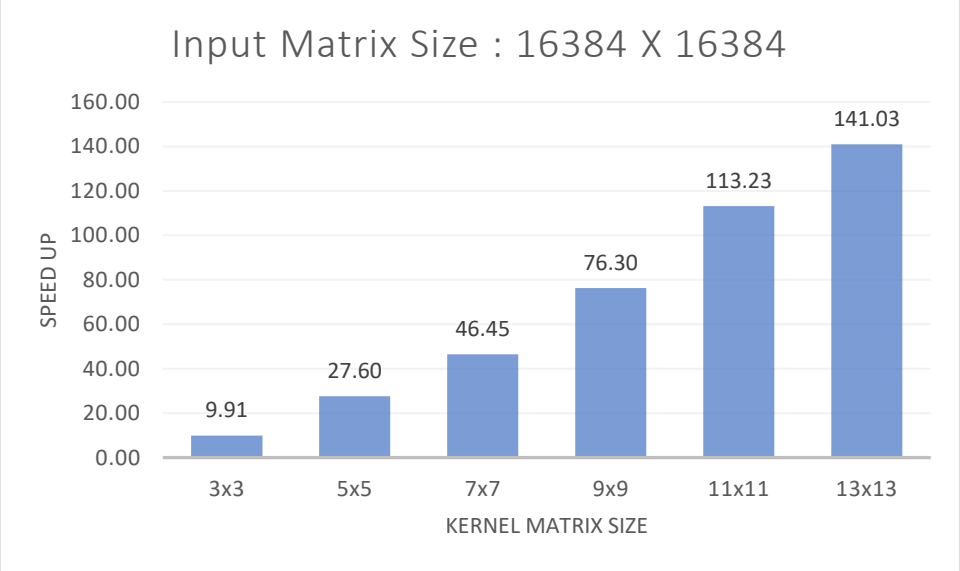




GPU Speed-Up


Input Matrix Size : 4096 X 4096
SPEED UP
700.00
600.00
500.00
400.00
300.00
200.00
100.00
0.00
4.63
14.51
23.87
39.20
51.83
74.90
431.92
512.27
607.11
3x3
5x5
7x7
9x9
11x11
13x13
64x64
128x128
1024x1024
KERNEL MATRIX SIZE


Input Matrix Size : 8192 X 8192
SPEED UP
600.00
500.00
400.00
300.00
200.00
100.00
0.00
8.17
23.40
39.87
64.70
87.93
119.86
489.28
3x3
5x5
7x7
9x9
11x11
13x13
64x64
KERNEL MATRIX SIZE

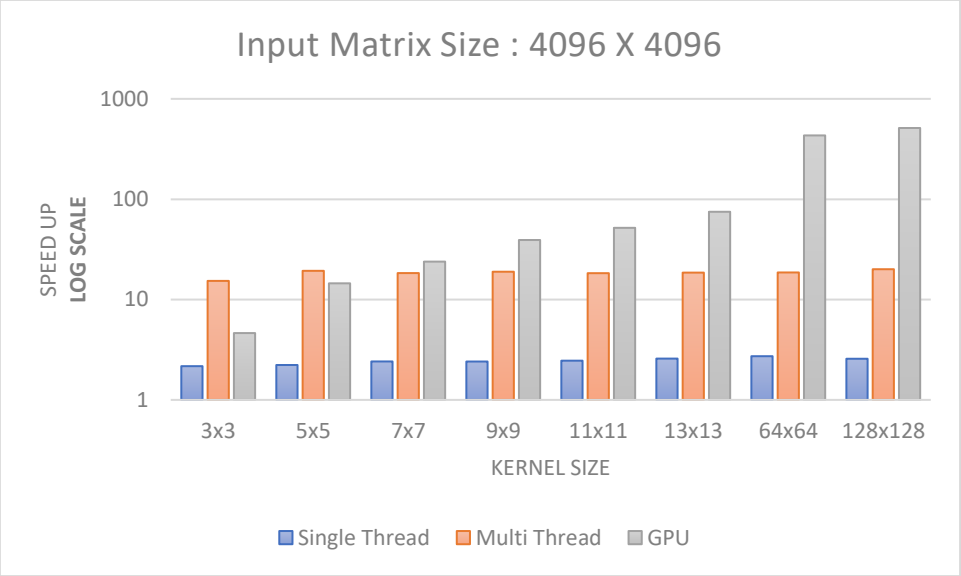

Input Matrix Size : 16384 X 16384
SPEED UP
160.00
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
9.91
27.60
46.45
76.30
113.23
141.03
3x3
5x5
7x7
9x9
11x11
13x13
KERNEL MATRIX SIZE

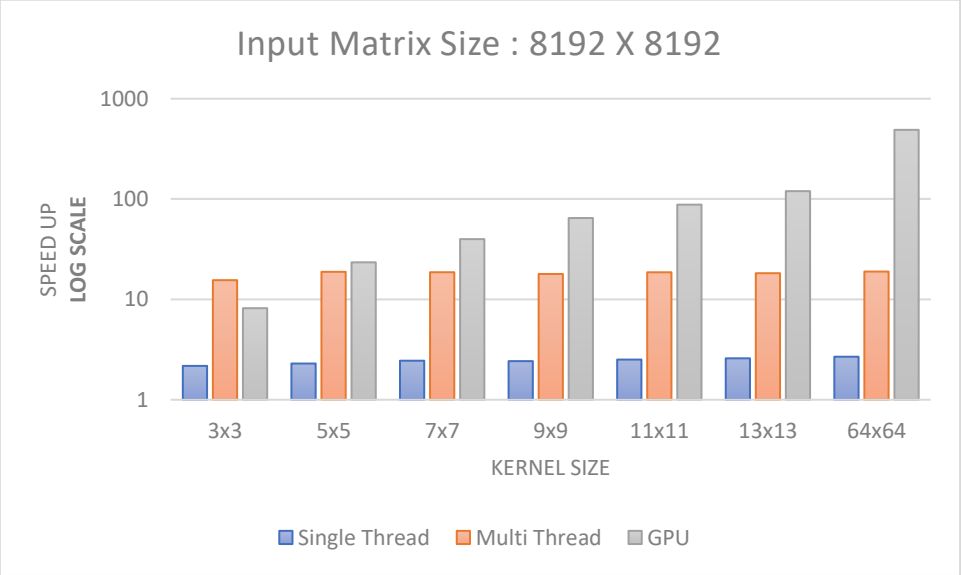
The acceleration attained through the utilization of CUDA demonstrates a substantial speed-up, reaching a factor as high as 607.11. This noteworthy improvement is observed when comparing the performance of CUDA against a naive CPU implementation. Specifically, the comparison is conducted for an input matrix size of 4096 and a kernel matrix size of 1024, showcasing the exceptional efficiency gains achieved by leveraging CUDA for parallelized computation on a GPU architecture in contrast to the sequential execution on a conventional CPU.

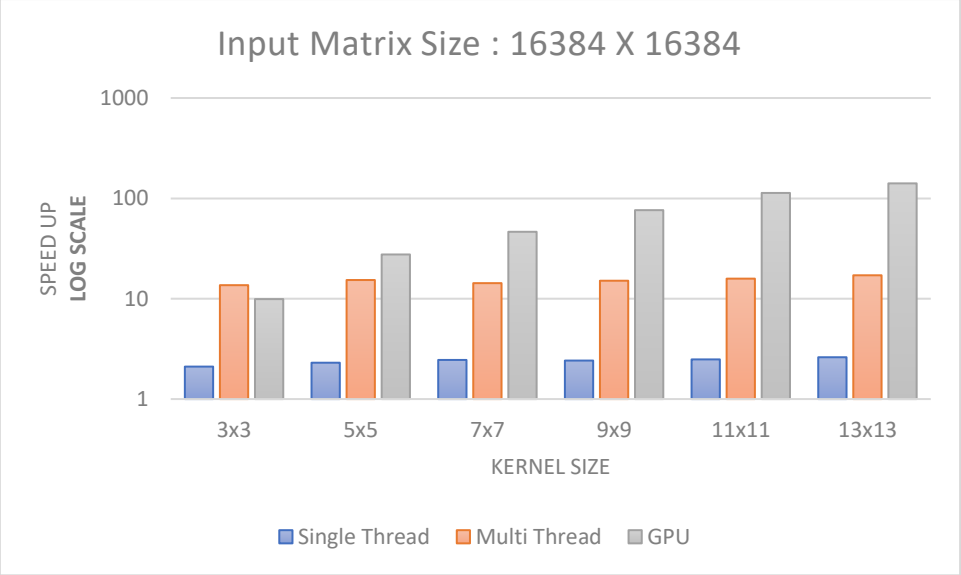
3. Conclusion

Comparison of Speed-up

Note: In graphs log scale is used to represent analysis.







- In the initial naive approach, parallelism and unrolling and other techniques were not incorporated, leading to the highest execution time due to the sequential nature of the computation. Subsequently, in the single-threaded implementation, we introduced unrolling and other techniques to enhance performance, resulting in a noticeable reduction in running time.

- The implementation of multi-threading yields a substantial enhancement in comparison to single-threaded execution, highlighting the efficacy of parallelization as a strategy for optimizing overall system performance.

- However, the most significant speedup was achieved through CUDA programming. By harnessing the power of the GPU, CUDA maximizes the utilization of a vast number of parallel threads, enabling a highly efficient and concurrent execution of the computation. This utilization of parallelism at the thread level allows CUDA to process a substantial amount of data concurrently, leading to a remarkable reduction in execution time. The observed speedup with CUDA underscores the pivotal role of parallel processing in optimizing performance for computationally intensive tasks, showcasing the advantages of leveraging specialized hardware for efficient and accelerated computations.