Data Analysis of CUDA Programming Project

A. Execution Time

Table 1. 2D Element-Wise Product Multiplication (C Version)

Number of Elements	Average Execution Time (uS)
1024	3523.30
2048	15183.86
4096	89321.61

Table 2. 1D Element-Wise Product Multiplication (CUDA)

Vector Size , Threads per Block	Average Execution Time (uS)
2^{20} , 256	1032.0
2^{22} , 512	4258.1
2^{24} , 1024	8199.2

Table 3. 2D Element-Wise Product Multiplication (CUDA)

Vector Size , Threads per Block	Average Execution Time (uS)
1024x1024 , 8x8	225.65
2048x2048 , 16x16	1242.6
4096x4096 , 32x32	7122

We initialized all the values of X and Y with 2 and 1 respectively for 1D and 2D CUDA versions. By comparing the execution times of each version, we can clearly see that the slowest out of all of them is the C version. As we compare the average execution time between the 1D and 2D CUDA versions, we can see that the recorded times for the 2D CUDA version are faster.

B. Difference/Effect of using 1D and 2D

In 1D matrix programming in CUDA, the matrices are represented in one dimension only, which is much simpler to implement and comprehend compared to a 2D matrix. A 1D matrix can also be accessed easier since each element can be accessed using a single index. This can increase the performance, since the easy access of each element can reduce memory latency. Although, 1D programming has limited parallelism

compared to 2D programming. This can lead to a decrease in performance, especially if the matrices are large.

In 2D matrix programming in CUDA, the matrices are represented as two dimensional arrays. Unlike 1D programming, 2D programming has increased parallelism, which means that it allows faster computation since multiple threads can be executed simultaneously. Although, the memory access pattern of a 2D matrix isn't as straightforward or as simple as a 1D matrix. The performance could also be affected negatively as the sizes of the matrices increase. In contrast with 1D, it is not as easy to comprehend and it is more complex to handle.

C. Effect of Block Sizes

Based on the data that we have recorded, we observed that the execution time increases as the block size increases. As the block size gets too large, it can lead to an increase in memory contention, which could cause the execution time to increase.

A smaller block size will have reduced memory contention, which improves the memory access patterns and performance. Although, it has limited instruction-level parallelism compared to higher block size, which means that it is less efficient since the ability to access multiple instructions simultaneously is limited.

A larger block size has better instruction-level parallelism. SInce more threads are available to execute instructions simultaneously, it can lead to more efficiency. Although, a larger block size has increased memory contention as mentioned earlier, which impacts the performance negatively.