1. Assignment 5: CUDA Matrix Multiplication with Streams

1.1 Performance Analysis

We implement CUDA matrix multiplication with streams. The goal of the implementation is to compare the performance of matrix multiplication with and without CUDA streams using two methods: (i) cuBLAS and (ii) custom CUDA kernel.

The cuBLAS library (cublasSgemm) is used for a highly optimized GPU matrix multiplication. A custom kernel implemented a tiled matrix multiplication using shared memory. Two CUDA streams were created with cuBLAS launched on one stream and kernel matrix multiplication launched on the second stream. Both operations executed concurrently on separate streams. We also launch both cuBLAS and the kernel sequentially without using any streams. The execution times are measured using CUDA events and results are shown in Table 0. We also graph the results in Figure 1 and Figure 2.

Table 0: Comparison of execution times with and without CUDA streams

	Execution Time (ms)			
Matrix Size	With streams	Without streams	Stream1 (cuBLAS)	Stream2 (kernel)
100x100	0.065	206.013	0.031	0.022
500x500	0.189	144.541	0.078	0.143
1000x1000	1.007	1.791	0.249	0.972
5000x5000	89.632	113.396	25.502	89.572
10000x10000	527.185	539.647	115.44	527.061

We observe for small matrix sizes, the total time with streams is much faster than without streams, even though both individual streams are very fast. For medium and large matrix sizes,

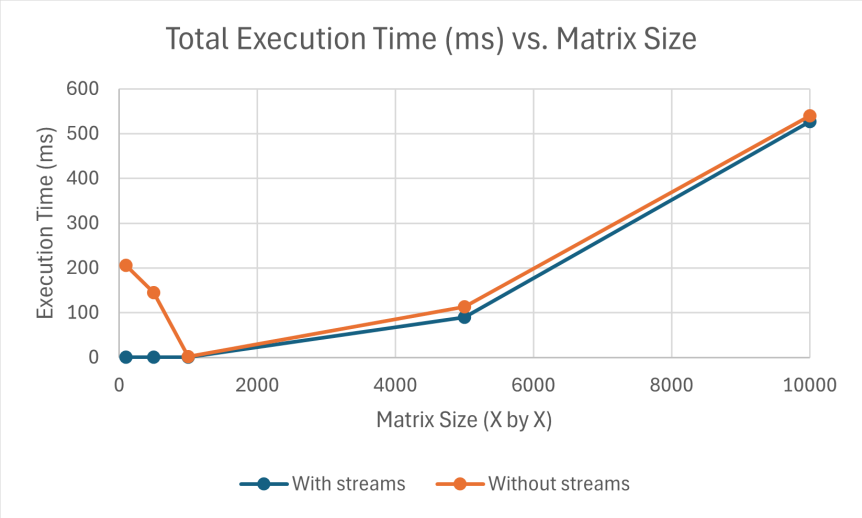

Figure 1: Comparison of total execution times with and without CUDA streams

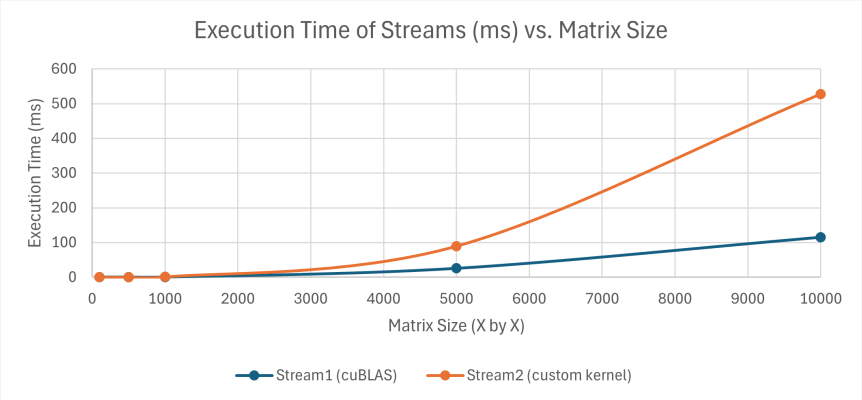

Figure 2: Comparison of individual stream exection times vs matrix size

we observe that the performance with streams remains only slightly better. One reason may be the custom kernel bottleneck (shown in Figure 2), i.e. concurrency will not help much when one task is much slower than the other. Although this may not be the case here, a limit of stream concurrency is the GPU resources, meaning streams are most beneficial when the individual tasks don't saturate the GPU.

Concurrent execution using CUDA streams can improve GPU utilization by overlapping independent tasks. In this case, streams only improve implementations with small matrix sizes, as

this is when the execution times of both tasks are similar. However, as the matrix size grows, the custom kernel is much slower than cuBLAS, thus concurrency does not increase execution time by much.