ECE 60827 Assignment 1

Shuting Du, du335@purdue.edu

Part A: Single-precision AX Plus Y (SAXPY)

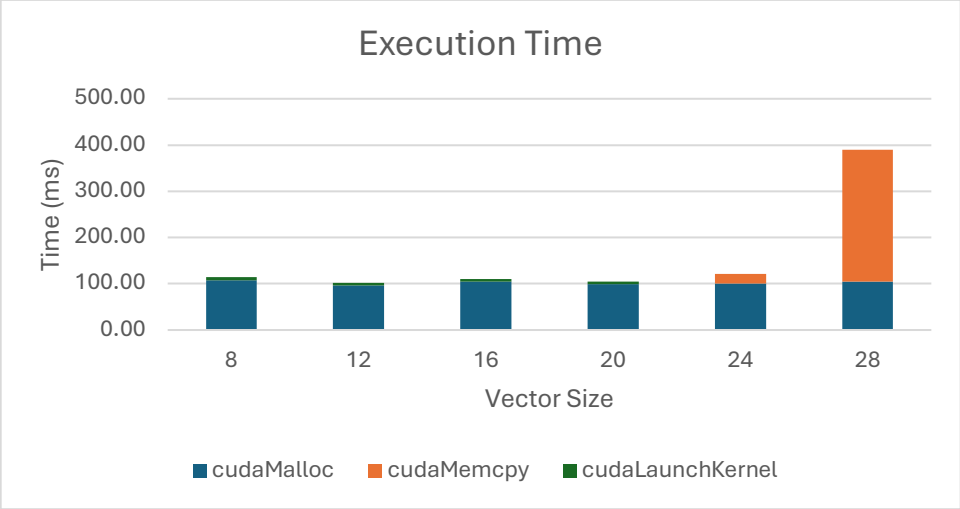

Figure 1

Figure 1 shows the execution time for different vector sizes. As the vector size increases, the total execution time remains relatively constant for smaller sizes. At vector size 24, there is a slight increase in execution time due to *cudaMemcpy* overhead. At vector size 28, execution time dramatically increases, primarily due to the *cudaMemcpy* operation. This suggests that memory transfer becomes the major bottleneck when handling larger vector sizes.

The execution time is divided into three main components: *cudaMalloc, cudaMemcpy, cuda KernelLaunch.* Memory allocation time is relatively constant across all vector sizes, and it remains the dominant cost for smaller vector sizes, the cost does not scale significantly with vector size. For small vector sizes, memory copy overhead is negligible. At vector size 24, memory copy overhead becomes noticeable. At vector size 28, memory copy dominates the execution time, indicating that memory transfer becomes the main bottleneck. Kernel launch time is negligible compared to memory operations. It suggests that kernel execution is efficient, and most overhead comes from memory operations.

Part B: Monte Carlo estimation of the value of Pi

Figure2~Figure4 illustrate the execution time breakdown for different CUDA operations (*cudaMalloc, cudaLaunchKernel,* and *cudaDeviceSynchronize*) while varying key computational parameters: Generate Block Number, Sample Size, and Reduce Size.

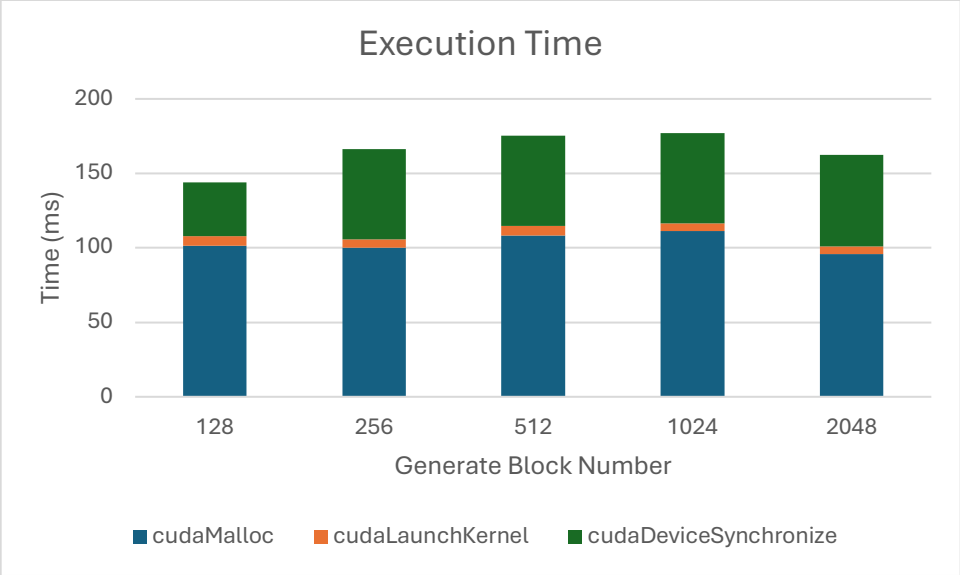

Figure 2

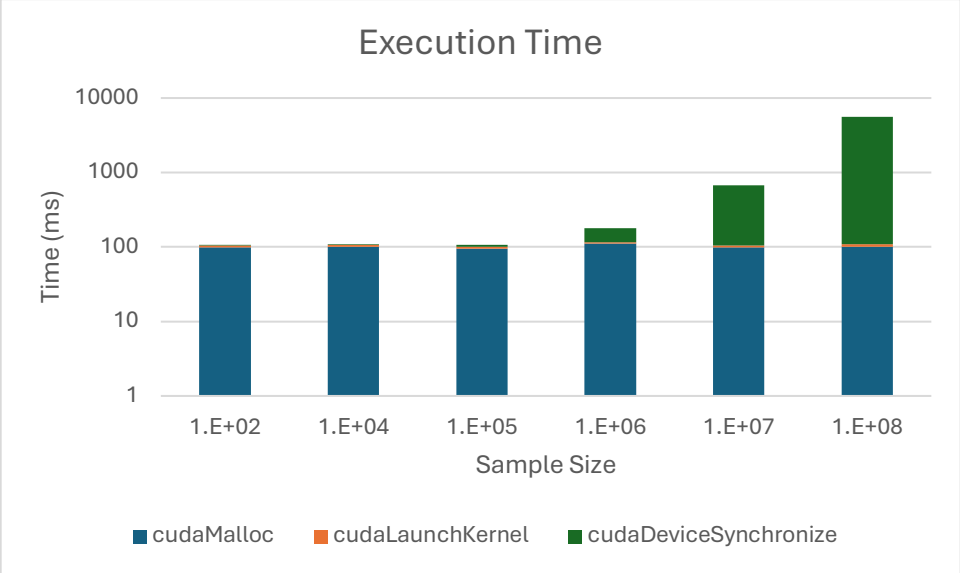

Figure 3

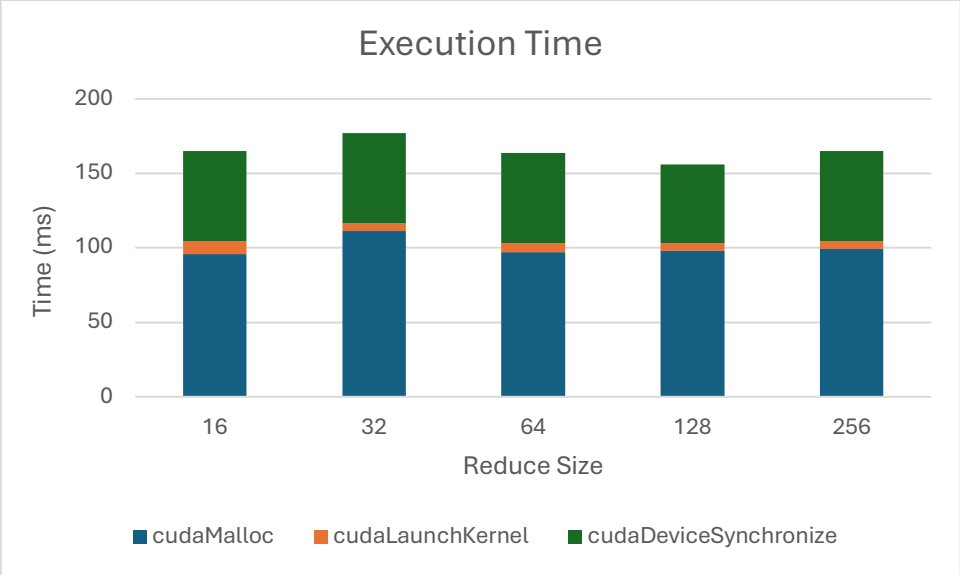

Figure 4

When varying the **Generate Block Number**, execution time remains relatively stable, with *cudaMalloc* being the primary time consumer. However, *cudaDeviceSynchronize* shows a slight increase, indicating that excessive blocks may introduce additional synchronization overhead. To optimize, it is important to balance the block count to avoid unnecessary synchronization costs.

For **Sample Size**, execution time remains stable for small values but increases sharply beyond 1E06, primarily due to *cudaDeviceSynchronize* becoming the dominant bottleneck. This suggests that memory transfer and synchronization overhead significantly impact performance at large scales.

In the case of **Reduce Size**, execution time is mostly unaffected, with *cudaMalloc* being the primary contributor to execution time, while *cudaDeviceSynchronize* shows minor fluctuations. This indicates that reduction operations do not heavily impact overall execution time. To optimize, it is beneficial to focus on reducing memory allocation overhead and utilizing shared memory for more efficient reductions.

Overall, the key optimizations involve minimizing synchronization overhead for large sample sizes, balancing the number of blocks to avoid excessive synchronization, and optimizing memory management to reduce allocation time. By applying these strategies, CUDA performance can be significantly improved.