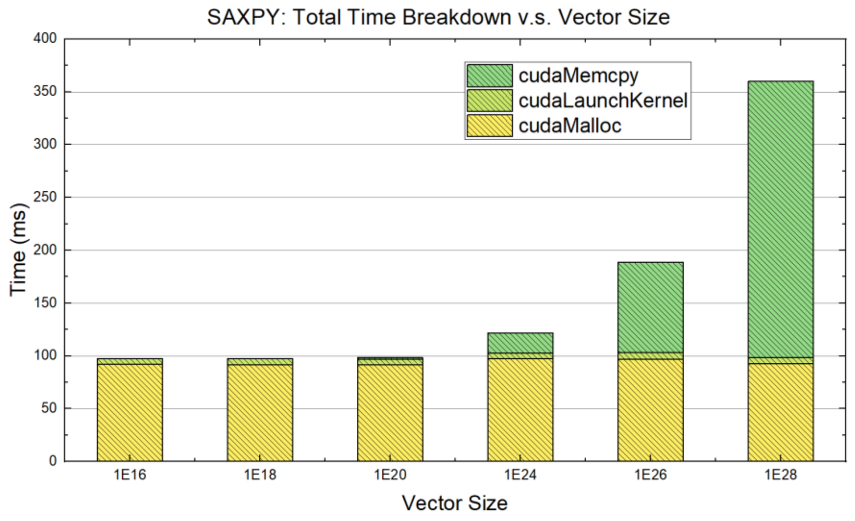
CUDA Programming Assignment #1 Report

Name: Luqi Zheng

1) SAXPY Workload:



The above figure shows that the total time breakdown of the SAXPY workload operation relative to vector size. The time components are divided into three parts: cudaMalloc, cudaLaunchKernel, and cudaMemcpy. Among them, cudaMalloc remains relatively constant across all vector sizes, indicating that memory allocation time on the GPU does not significantly depend on the vector size. cudaLaunchKernel contributes minimally to the total time. However, cudaMemcpy shows a significant increase as the vector size grows, especially after the vector size approachs1E24, which suggests that data transfer between host and GPU becomes the primary bottleneck for larger vectors.

2) Monte Carlo Workload:

(a)

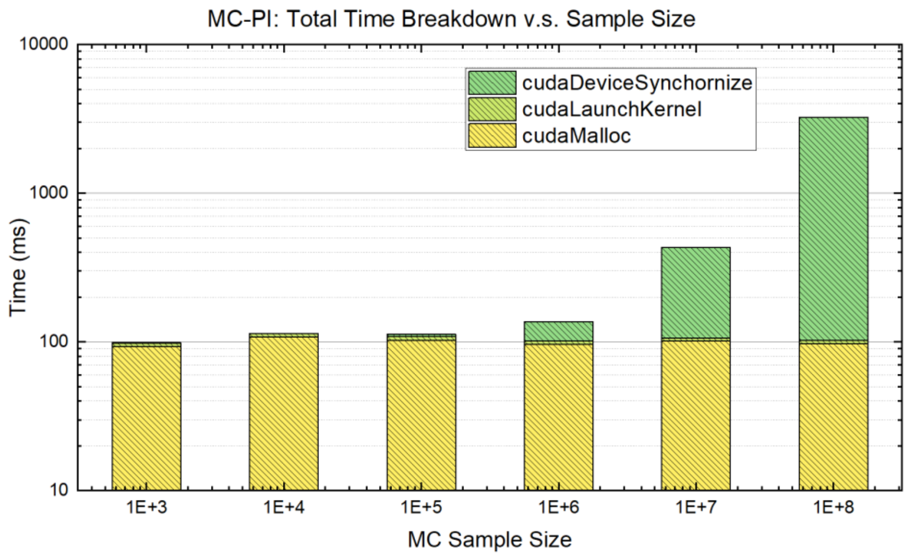


Figure (a) shows the total time breakdown for a Monte Carlo PI computation as a function of the sample size on a logarithmic scale. The total time is divided into three components: cudaMalloc, cudaLaunchKernel, and cudaDeviceSynchronize. cudaMalloc remains nearly constant across all sample sizes, indicating that GPU memory allocation time is independent of the sample size, which aligns with our previous observations in SAXPY case. cudaLaunchKernel continues to have a negligible contribution to the total time, suggesting that launching the kernel incurs minimal overhead. Nevertheless, cudaDeviceSynchronize increases significantly with larger sample sizes. For smaller sample sizes (up to 1E5), synchronization time is minimal, but beyond 1E6, it becomes the dominant contributor to total time, especially for 1E7 and 1E8.

(b)

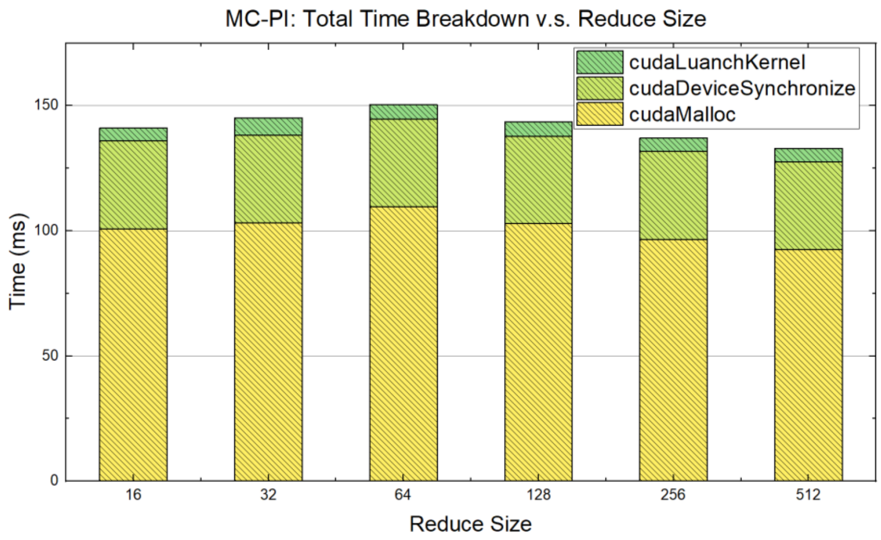
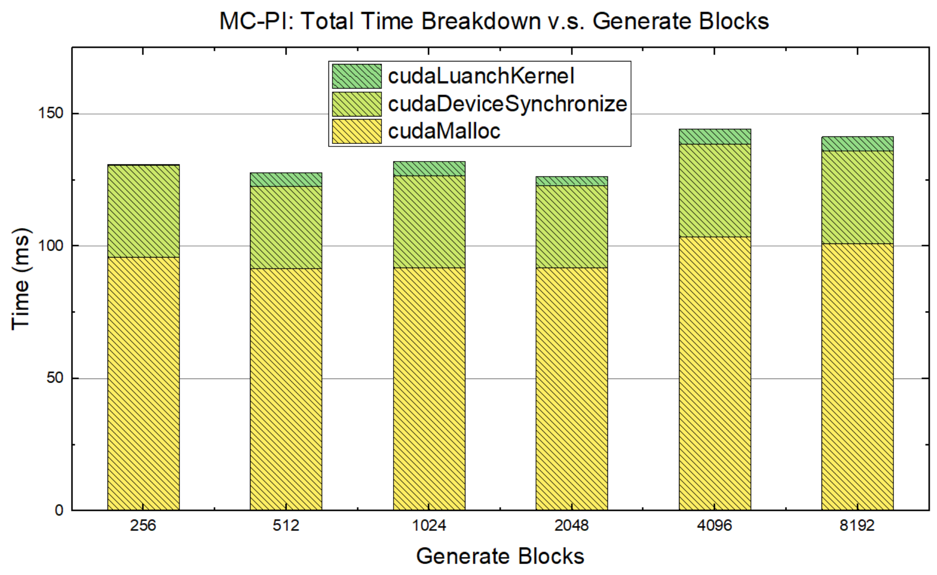


Figure (b) shows the total time breakdown for a Monte Carlo PI computation as a function of the sample size on a logarithmic scale. The total time is divided into three components: cudaMalloc, cudaDeviceSynchronize, and cudaLaunchKernel. cudaMalloc is the dominant contributor across all reduce sizes, and its time remains relatively consistent, indicating that memory allocation time is largely independent of the reduction size. cudaDeviceSynchronize shows slight variations, which suggests that synchronization overhead is mildly sensitive to reduce size but does not scale significantly. cudaLaunchKernel contributes the least to the total time, and its impact remains almost negligible across all reduce sizes.

(c)

Figure (c) shows the total time breakdown for a Monte Carlo PI computation as a function of the sample size on a logarithmic scale. The total time is divided into three components: cudaMalloc, cudaDeviceSynchronize, and cudaLaunchKernel. cudaMalloc remains the dominant contributor across all block sizes, with slight fluctuations but generally consistent performance. This suggests that memory allocation time is largely unaffected by the number of generate blocks. cudaDeviceSynchronize shows more variability, peaking at 256 and 4096 blocks. This indicates that synchronization overhead can be



influenced by the number of blocks, possibly due to load balancing or thread divergence issues. However, cudaLaunchKernel consistently contributes the least to total execution time, indicating that the kernel launch overhead is minimal regardless of block count.

3) Summary:

In this CUDA programming assignment, we evaluate the performance breakdown of SAXPY and MC-PI computations on the GPU. Across all cases, cudaMalloc remains a consistent time contributor, especially for smaller data sizes. As vector or sample sizes increase, cudaMemcpy (in SAXPY) and cudaDeviceSynchronize (in MC-PI) become the dominant bottlenecks, highlighting the growing cost of data transfer and synchronization with larger workloads. cudaLaunchKernel consistently shows minimal overhead, indicating efficient kernel launching. Overall, optimizing memory allocation, data transfer, and synchronization is key to improving GPU performance.