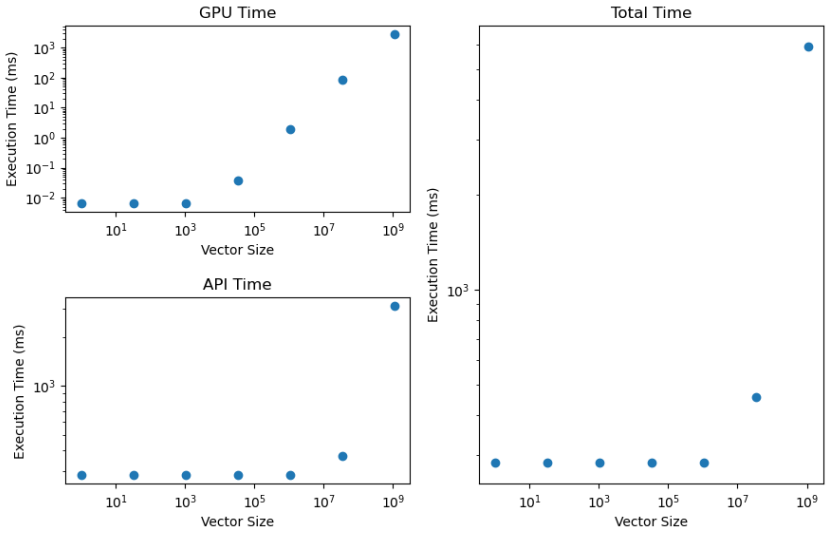
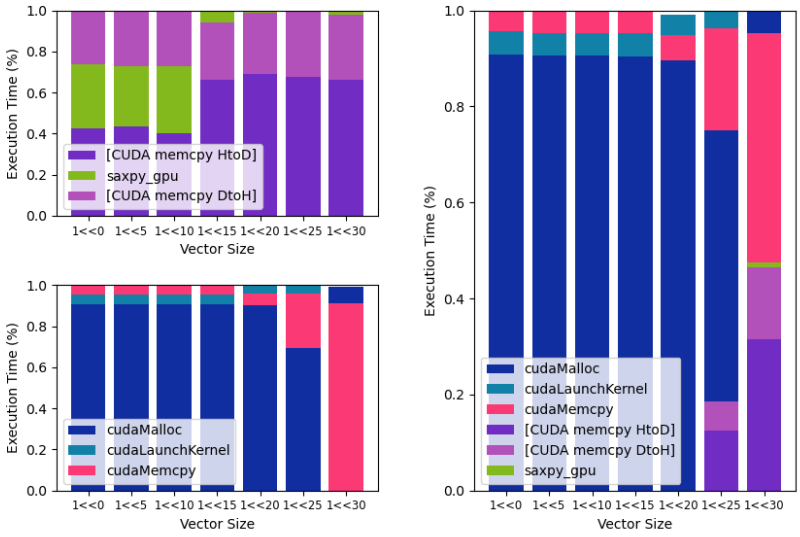
Eric Rodenkirch
ECE 60827
2/1/25

Lab 1 Report

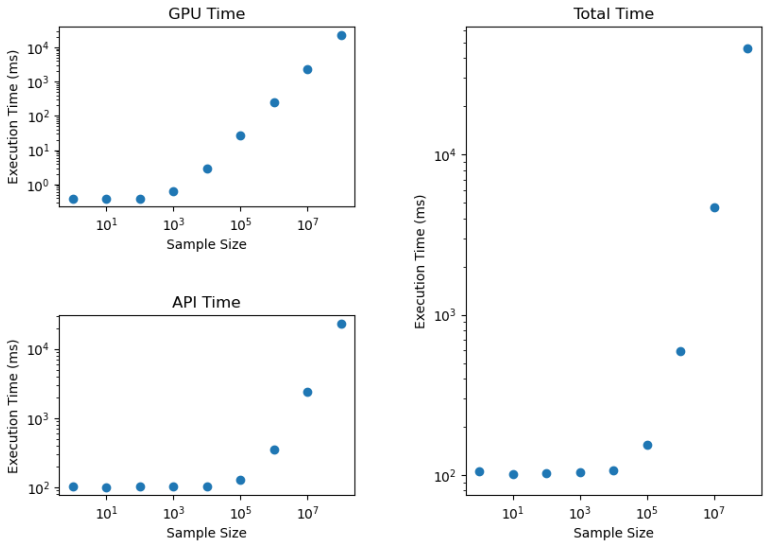


In the graph above we can see the execution time for the SAXPY GPU kernel across various vector sizes. With vector size on a log-scale we can see that execution time is linear with relation to the vector size with the exception of 1, 32, and 1024 elements. From this it appears that there may be some minimum time overhead when running a GPU kernel, regardless of the size the data is processing.

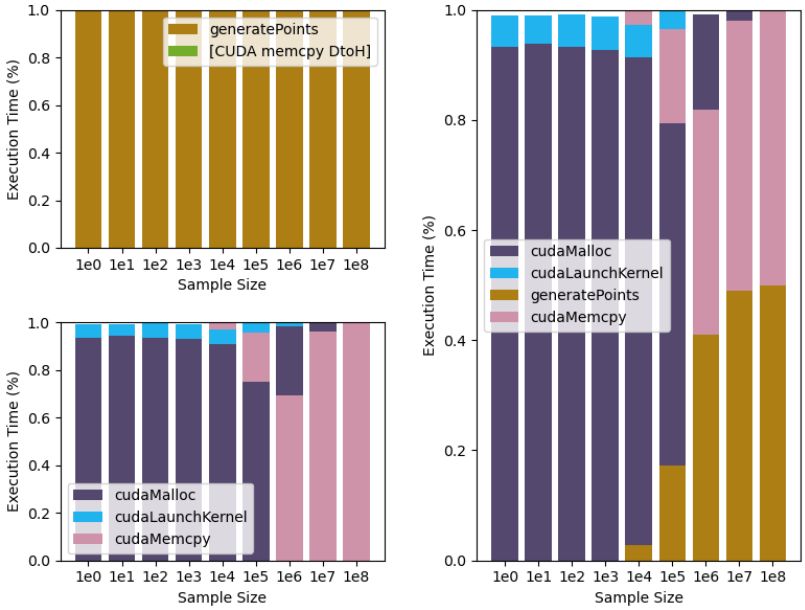


As the vector size increases, the SAXPY kernel becomes increasingly CPU bound. For GPU activity, we can see that memcpy becomes nearly 100% of the activity beyond 2^{15} vector size. Memcpy Host to Device takes 2/3rds of the time, and Device to Host takes 1/3rd of the time. This

is because 2 vectors are copied from host to device, but only one vector is copied back. Similarly, memcpy takes over most of the percentage time for API calls for large vectors. The SAXPY kernel execution time is also generally dominated by the CUDA API calls rather than GPU work. This is a result of SAXPY being an incredibly simple calculation for the GPU. Only for extremely large vectors does the GPU activity become a significant portion of the total execution time.



For Monte Carlo execution time (seen above), we see the GPU execution time scaling linearly with the sample size, once it's passed the minimum overhead.



Monte Carlo execution time breakup is significantly less memory bound as a result of each GPU thread generating its own random points to sample rather than requiring points to be sent to it

from the CPU. It then only needs to transfer the psums back to the CPU. We do see this become a significant time consumer for very large sample sizes. Given the total execution time of the program compared to the time spend executing on the CPU, the CPU summation is fairly minor (~6 seconds) compared to the GPU execution time (~26 seconds) for very large sample sizes. However, the large amount of time used for Memcpy points to a major reason to use reduce threads rather than a full CPU summation– you will be sending much less data back to the CPU.