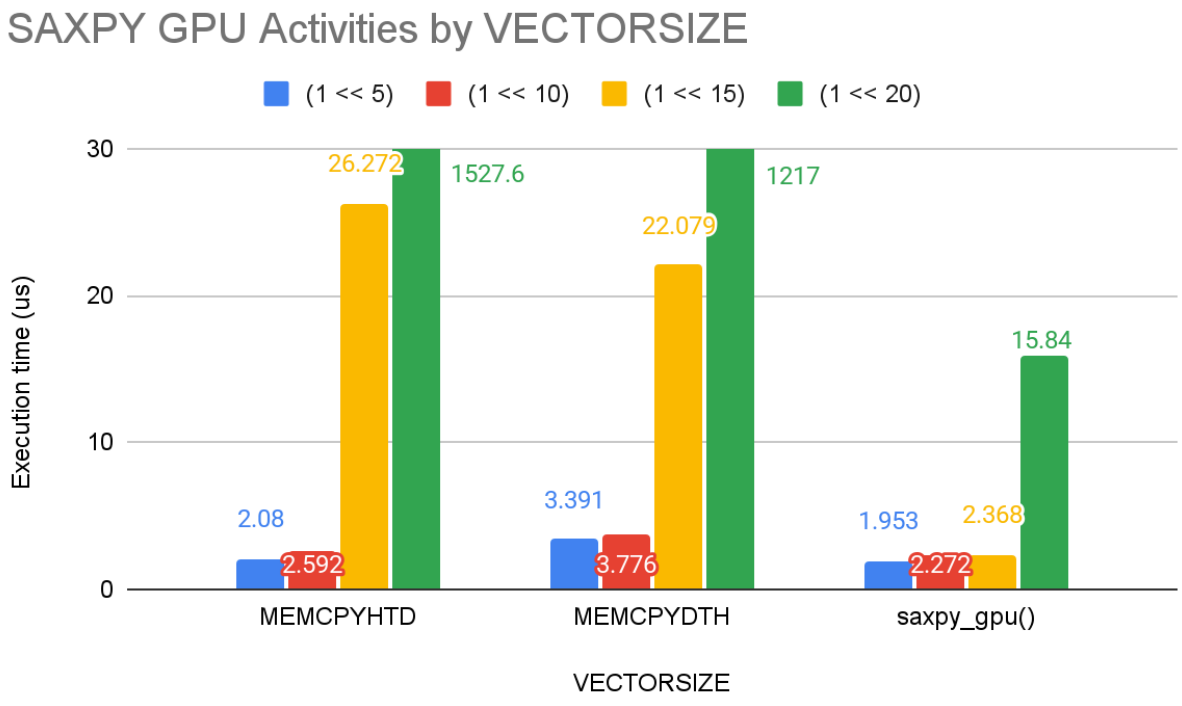
Braden Karley
ECE60827

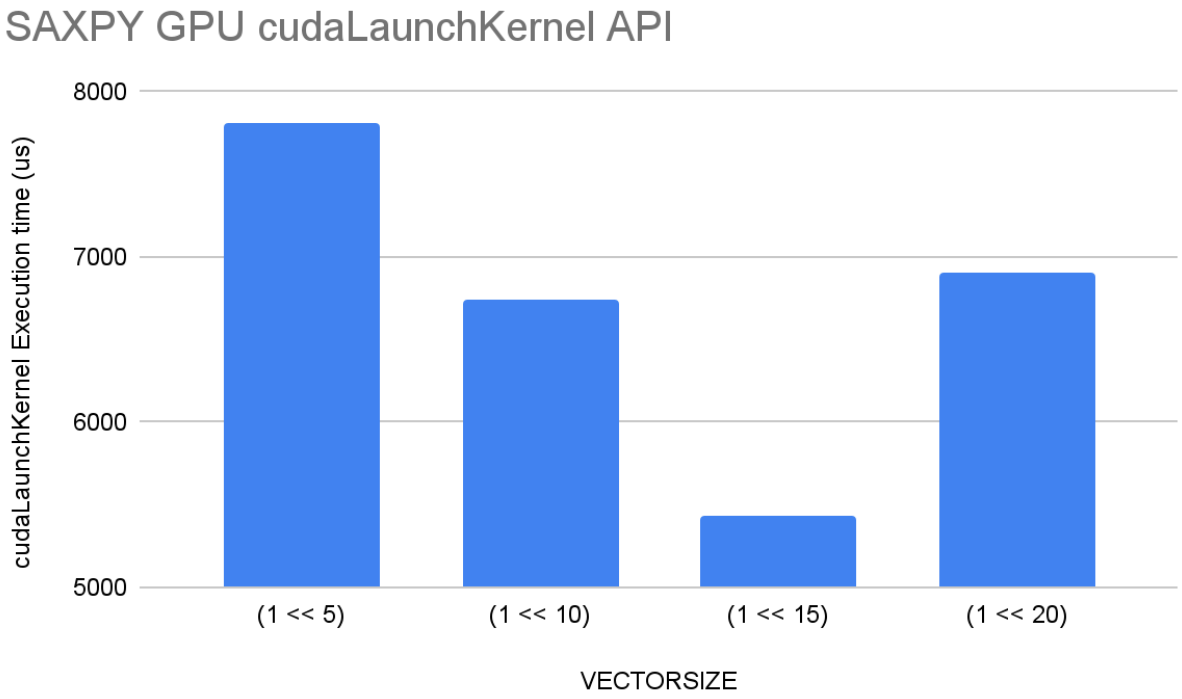
CUDA Lab 1

This lab assignment was an exercise intended to cover some of the basics of CUDA programming, as well as distinguish between algorithms suited for parallelism. The SAXPY program was hardly sped up by moving the workload to the GPU. However, the Monte Carlo simulation was significantly sped up by moving the workload to the GPU. Running on the CPU, it took nearly a minute to run all the iterations sequentially and estimate pi, but running on the GPU allowed each iteration to execute in parallel and accurately estimate pi in about 0.15 seconds.

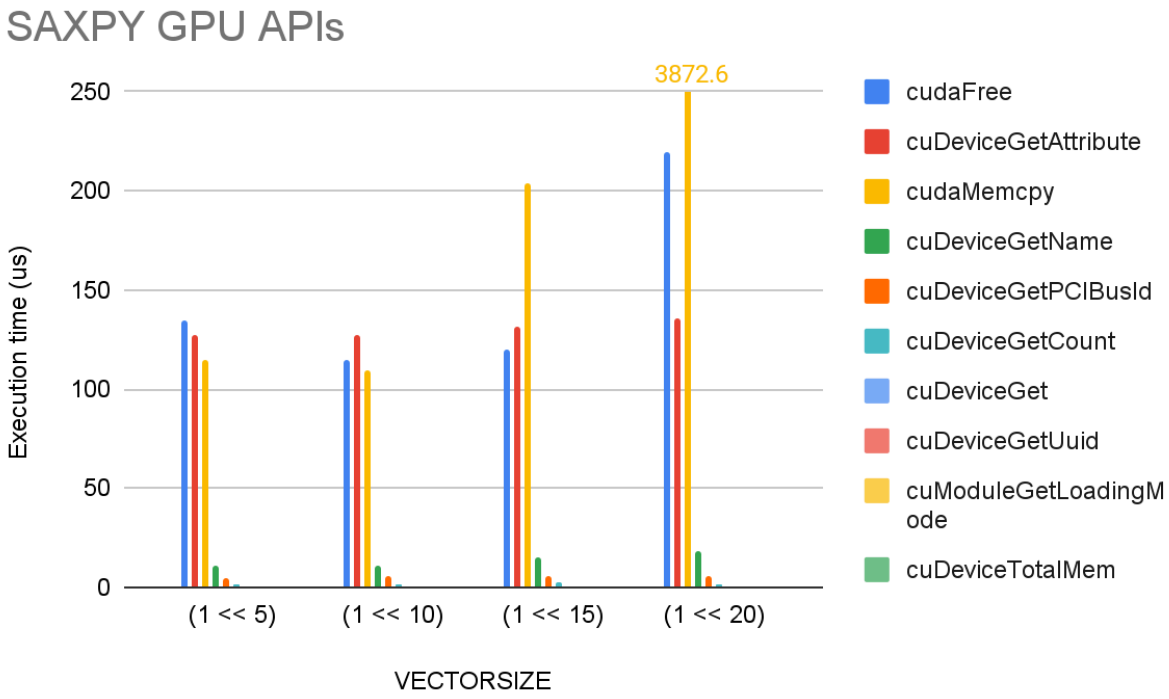
By altering some variables, we can see the effect on the GPU execution times by using the 'nvprof' command. The figure below varies the size of the vectors involved in the SAXPY program, and clearly there are significant delays the larger the memory. The gap between the execution times of 1 << 15 and 1 << 20 is the most significant, particularly when data is copied from the host to the device, or vice versa. It seems that the change in vector size did not have a noticeable effect on the actual saxpy_gpu() kernel until 1 << 20 was used.



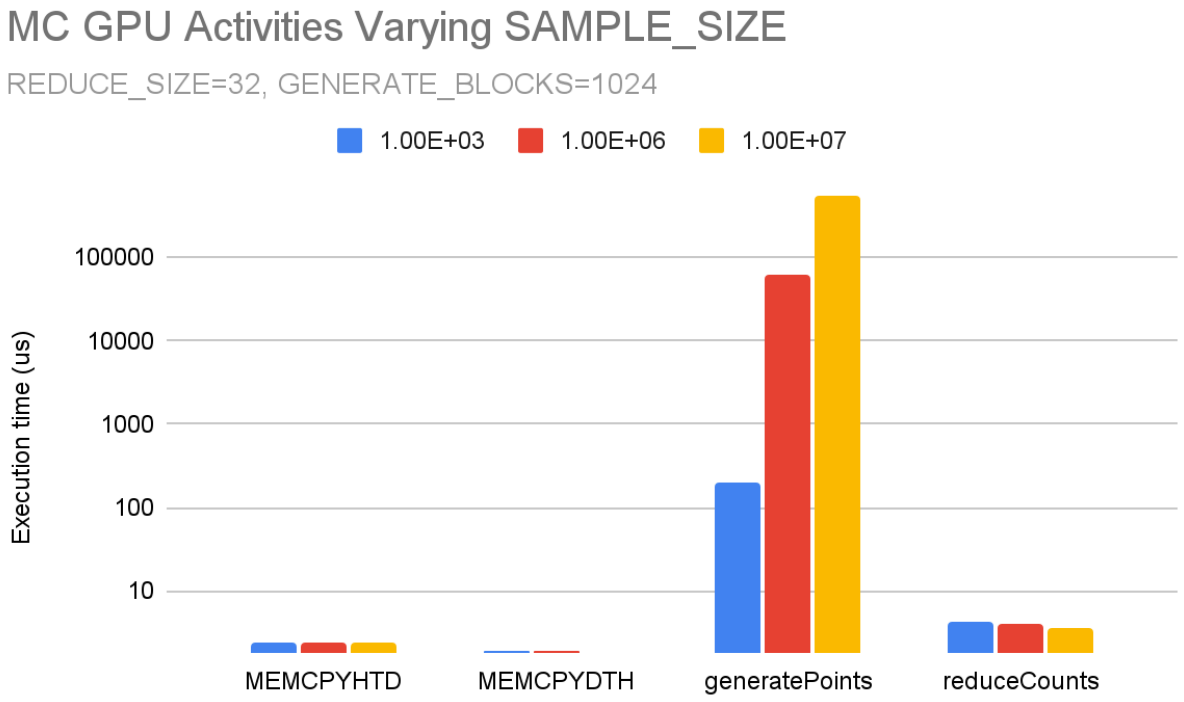
An effect was also observed on the API execution times in the program. While the cudaLaunchKernel API varied greatly between 5000 and 8000 microseconds, it is not clear how exactly vector size affects it. Several other APIs, however, do appear to have a direct correlation.



As the vector size increased, the APIs related to memory increased, particularly when the size was greater than 1 << 15. The API cudaFree increased at 1 << 20, but the API cudaMemcpy increased at 1 << 15, and grew exponentially at 1 << 20 to nearly 20x what it was at 1 << 15. The other APIs were relatively insignificant and showed very little change because they had little to do with the size of the vectors.



In the Monte Carlo simulations, there are very clear effects as the variables change. In the three graphs below, the GPU activities and API execution times show distinct correlations. As the sample size increases, the generatePoints kernel increases in execution time, while the reduceCount kernel decreases, oddly enough.

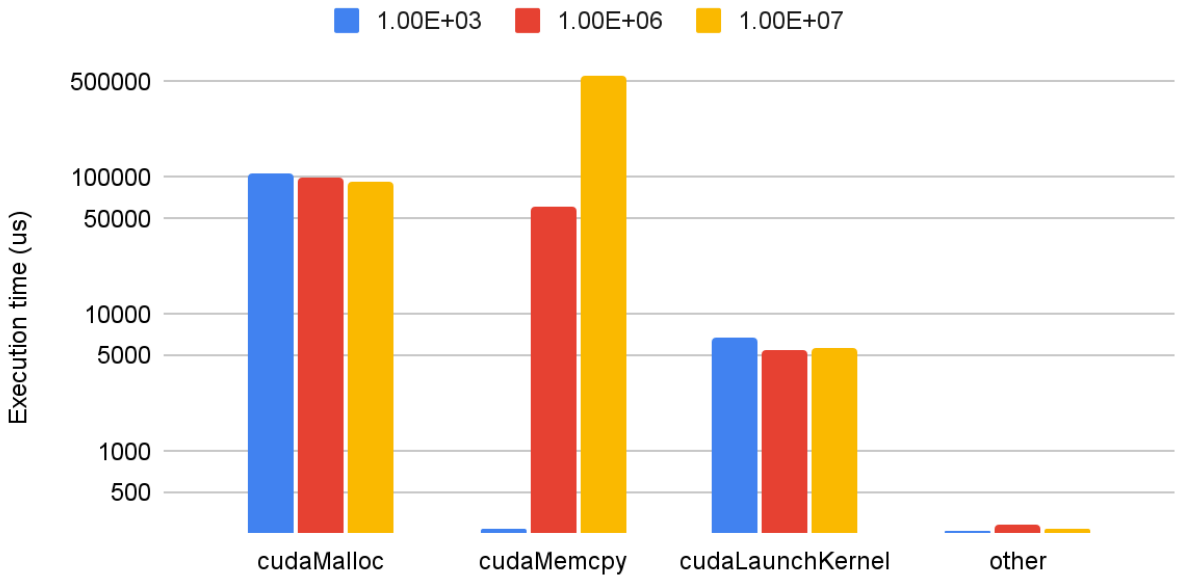


Similarly to the SAXPY program, a very clear effect is shown in the memory APIs. Wherever data is copied to/from the device, the execution time increases greatly with the sample size. This seems like an obvious observation, as it should take more time to move more data, and the nvprof metrics prove it.

To maintain accurate estimations of pi, the GENERATE_BLOCKS variable was constrained to be more than or equal to the square of the REDUCE_SIZE variable. Graphing was simply easier if the ratio of the two were shown rather than showing them independently. Strangely, there appeared to be little effect on the time it took to copy data from the device back to the host or on the generatePoints kernel. However, execution times increased for copying data from the host to the device and for the reducePoints kernel. This is likely due to the fact that the reducePoints kernel was reducing the size of pSums and moved a smaller amount of data back to the host. It would take longer to reduce the number of values in each thread the more data in the blocks being processed.

MC GPU APIs Varying SAMPLE_SIZE

REDUCE_SIZE=32, GENERATE_BLOCKS=1024



MC GPU Activities Varying REDUCE_SIZE/GENERATE_BLOCKS Ratio

SAMPLE_SIZE=1e6