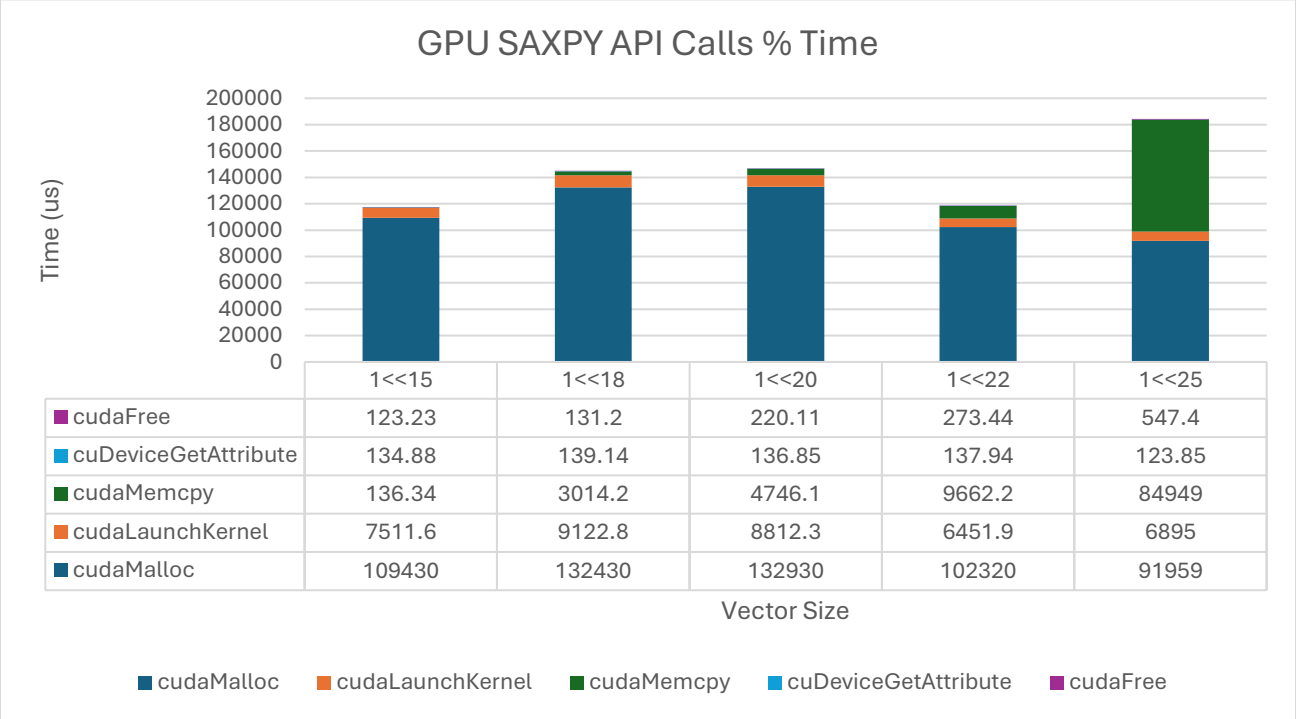
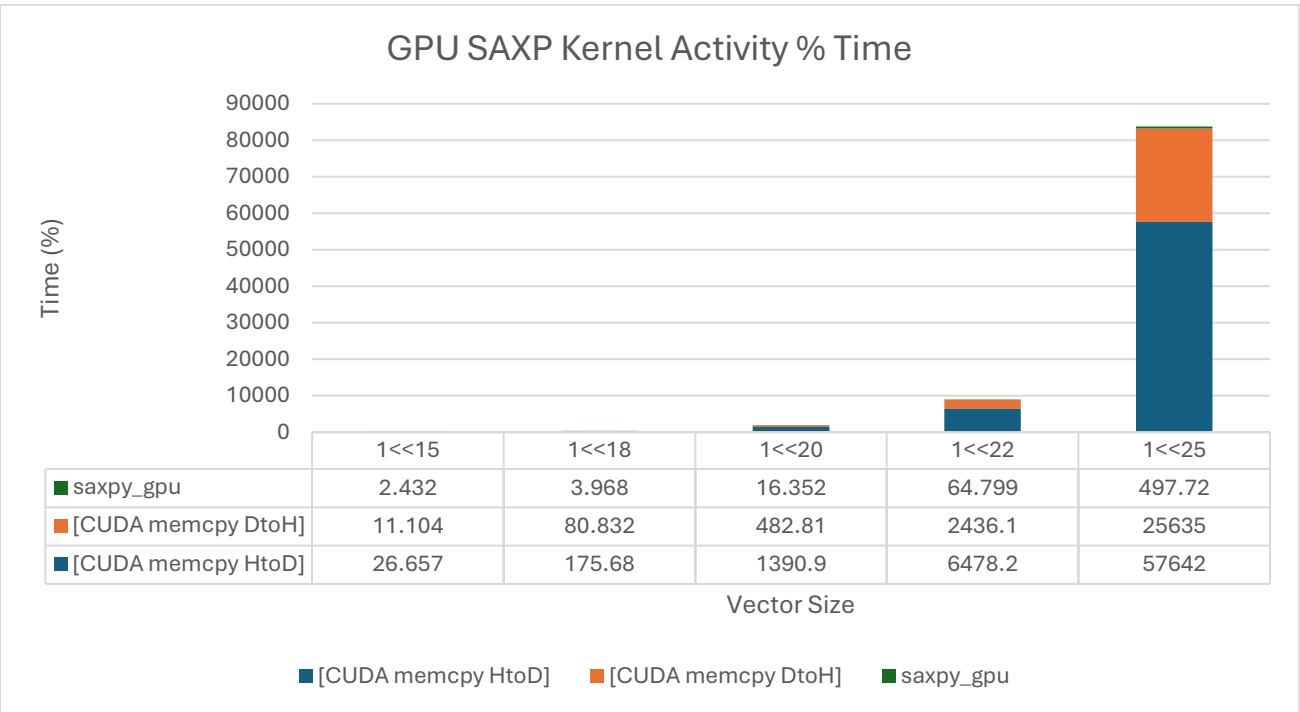
Caroline Jin
CUDA Programming Assignment 1 (Part A and B)

SAXPY:



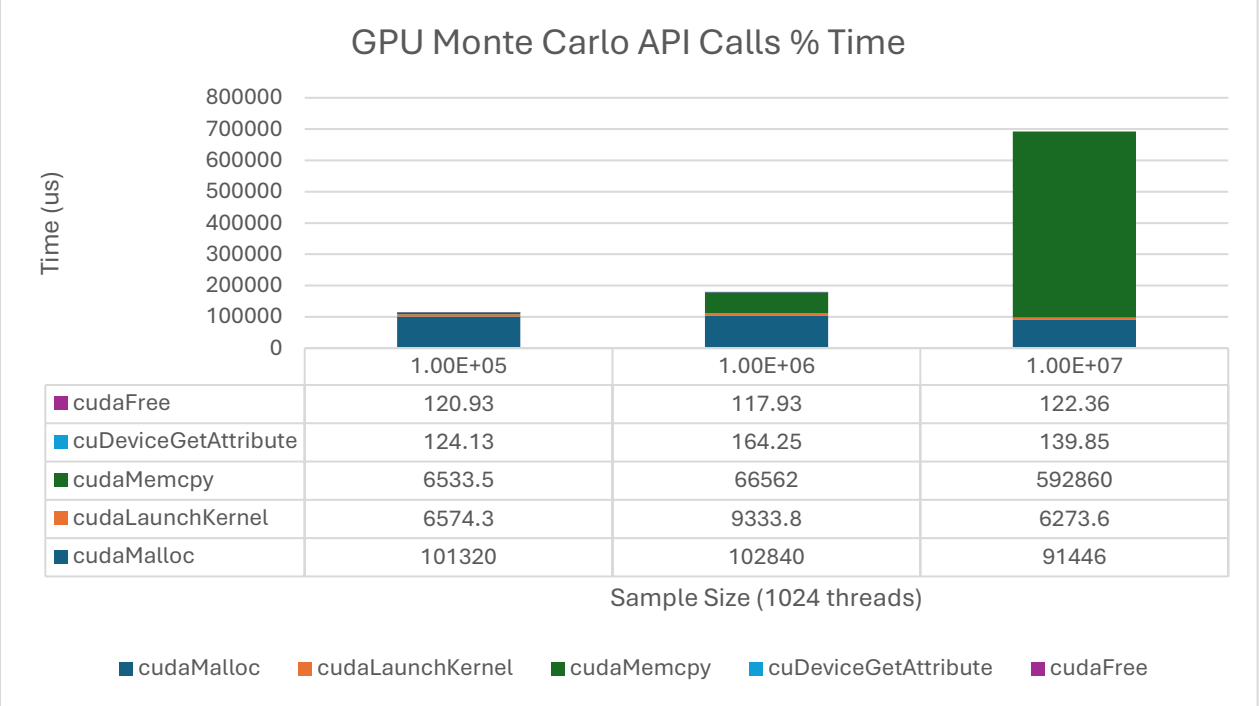
	1<<15	1<<18	1<<20	1<<22	1<<25
cudaFree	123.23	131.2	220.11	273.44	547.4
cuDeviceGetAttribute	134.88	139.14	136.85	137.94	123.85
cudaMemcpy	136.34	3014.2	4746.1	9662.2	84949
cudaLaunchKernel	7511.6	9122.8	8812.3	6451.9	6895
cudaMalloc	109430	132430	132930	102320	91959

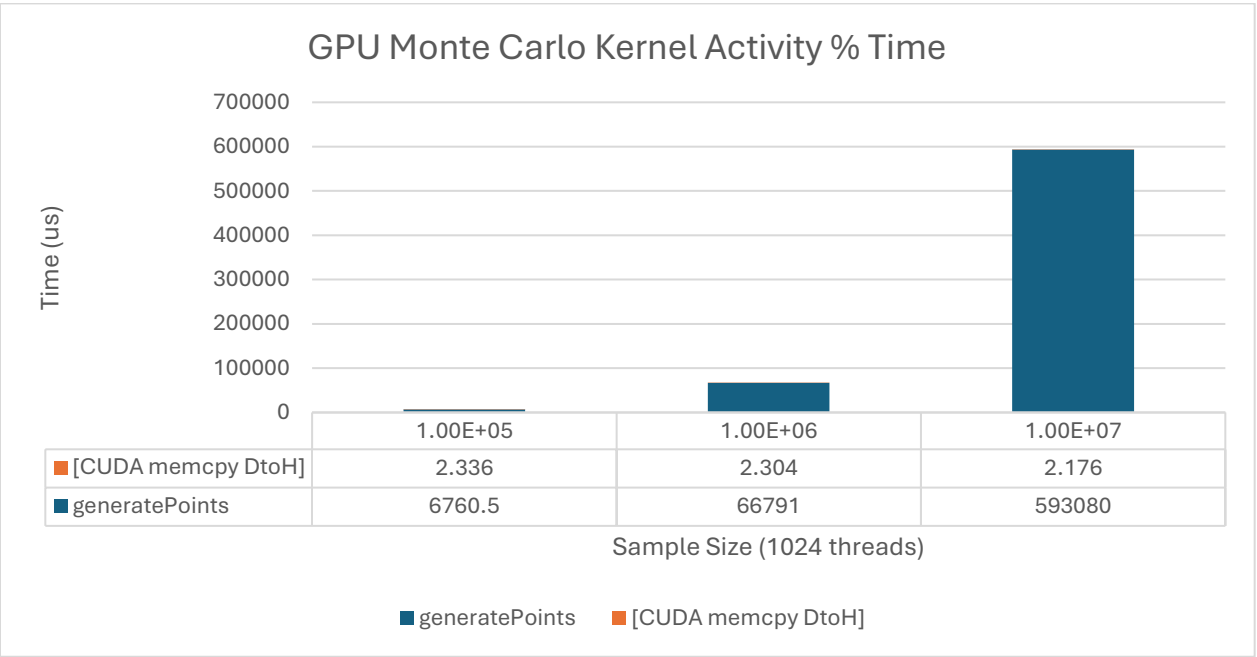


	1<<15	1<<18	1<<20	1<<22	1<<25
saxpy_gpu	2.432	3.968	16.352	64.799	497.72
[CUDA memcpy DtoH]	11.104	80.832	482.81	2436.1	25635
[CUDA memcpy HtoD]	26.657	175.68	1390.9	6478.2	57642

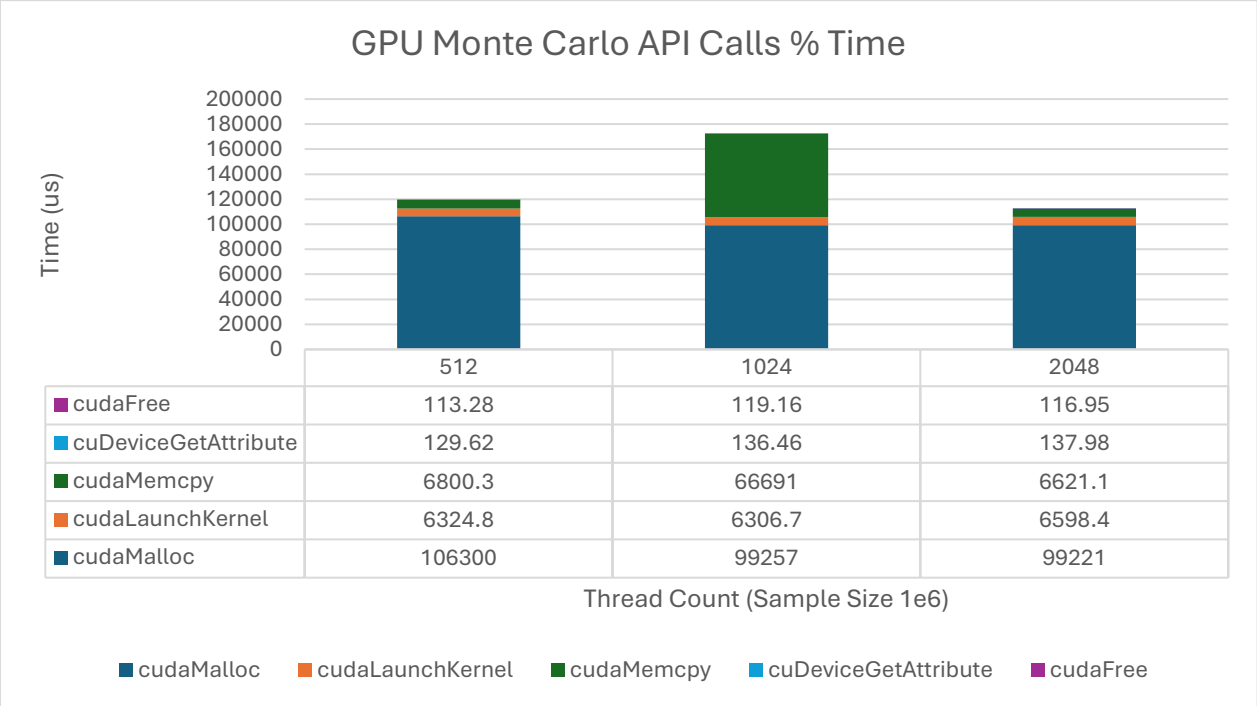
By varying the vector size for the GPU SAXPY code, there is a clear observation that an increased vector size greatly multiplies the time spent in the kernel. Additionally, a larger vector means the host code spends a greater amount of time copying data from the host to device (and vice versa) via cudaMemcpy.

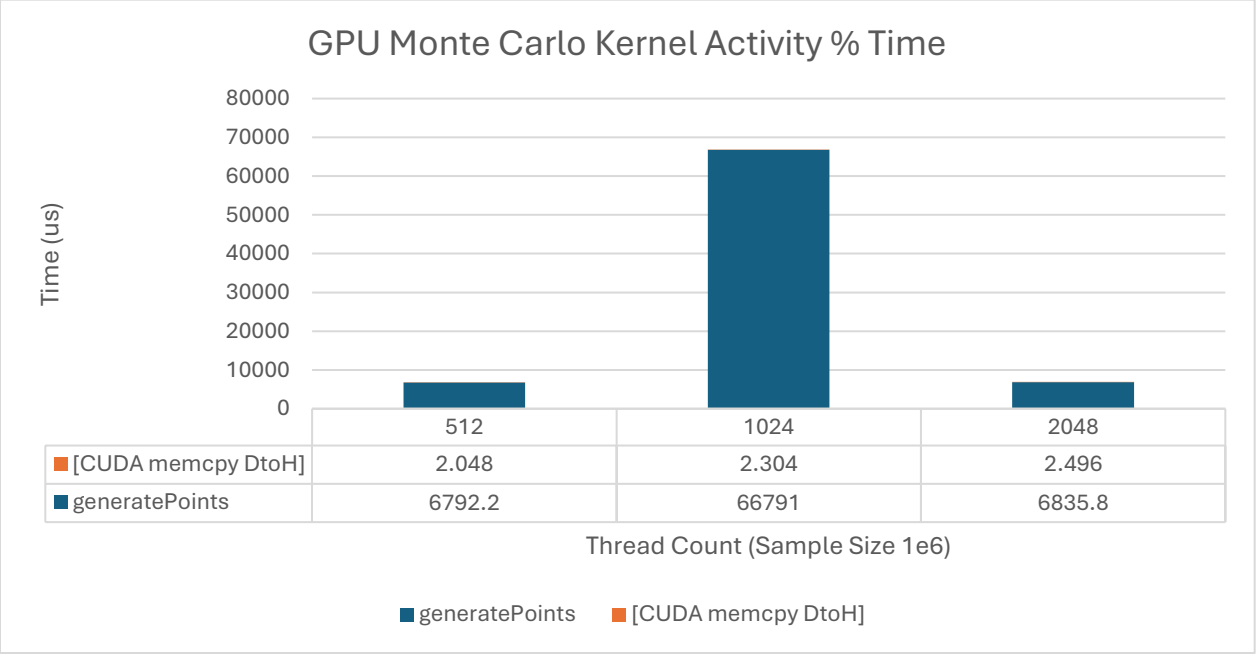
Monte Carlo Pi Approximation:





By varying sample size (the amount of points each thread will generate), the pattern shows exponentially greater time spent in the kernel. In the host code, there was a larger amount of time spent in cudaMemcpy for larger sample size. In the code, data is only transferred one way from device to host. In this transaction, with a greater sample size, the number of hits would be a lot larger per thread, leading to a longer delay.





When varying the thread count, there is a spike in time spent in the kernel and cudaMemcpy at 1024 threads compared to the lower and higher number of threads. Based on the number of blocks and how the memory/scheduling is handled in the GPU possibly the higher number of thread count per block is not actually as efficient all the time.