Julie Weller

GitHub: juliewiltbank

ECE 60827: Lab 1 Report

SAXPY Execution:

The SAXPY kernel profiling data is shown below for three different inputs of VectorSize (see Table 1). The inputs all vary dramatically ranging from 1024 to 131072.

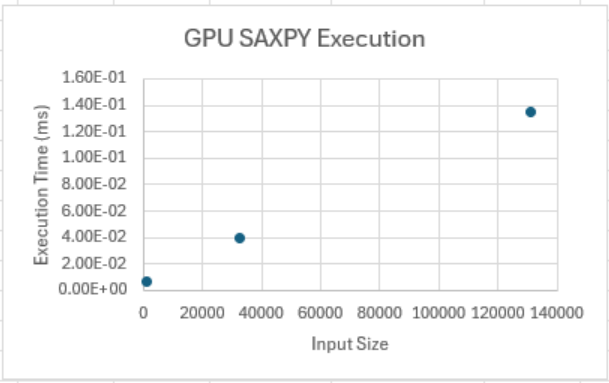
Input Size	Kernel / API Call	SAXPY Program Execution Time (ms)	Percentage of Total Time (%)
1024	cudaMemcpy HtoD	2.59E-03	38.94
1024	cudaMemcpy DtoH	2.02E-03	30.77
1024	saxpy_gpu()	2.05E-03	30.29
1024	cudaMalloc	110.43	94.21
1024	cudaLaunchKernel	6.40E+00	5.46
32768	cudaMemcpy HtoD	2.62E-02	66.21
32768	cudaMemcpy DtoH	1.11E-02	27.93
32768	saxpy_gpu()	2.34E-03	5.86
32768	cudaMalloc	92.967	93.32
32768	cudaLaunchKernel	6.0815	6.1
131072	cudaMemcpy HtoD	9.05E-02	66.99
131072	cudaMemcpy DtoH	4.12E-02	30.55
131072	saxpy_gpu()	3.33E-03	2.46
131072	cudaMalloc	103.37	92.32
131072	cudaLaunchKernel	6.7579	6.04

GPU
API

Table 1: SAXPY Raw Data from nvprof Profiling Tool

This data is further synthesized into total execution time from both Kernel and API calls for each input size (see Table 2). A notable trend is that the GPU execution time scales with input size but the API calls took largely the same time despite scaling the input size. Indeed when profiling the data, the input size for API made no difference in the time it took and varied by ~20ms randomly each program run through.

GPU		API	
Input Size	Execution Time	Input Size	Execution Time
1024	6.66E-03	1024	116.8285
32768	3.97E-02	32768	99.0485
131072	1.35E-01	131072	110.1279



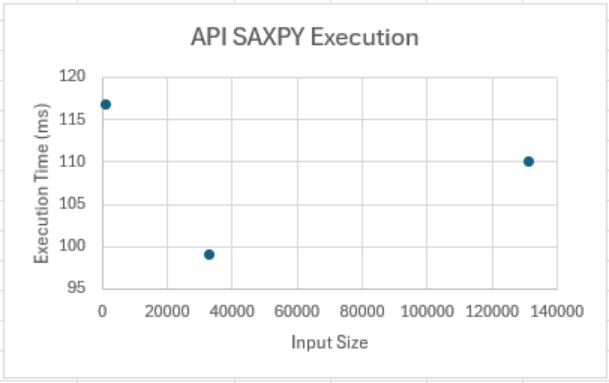
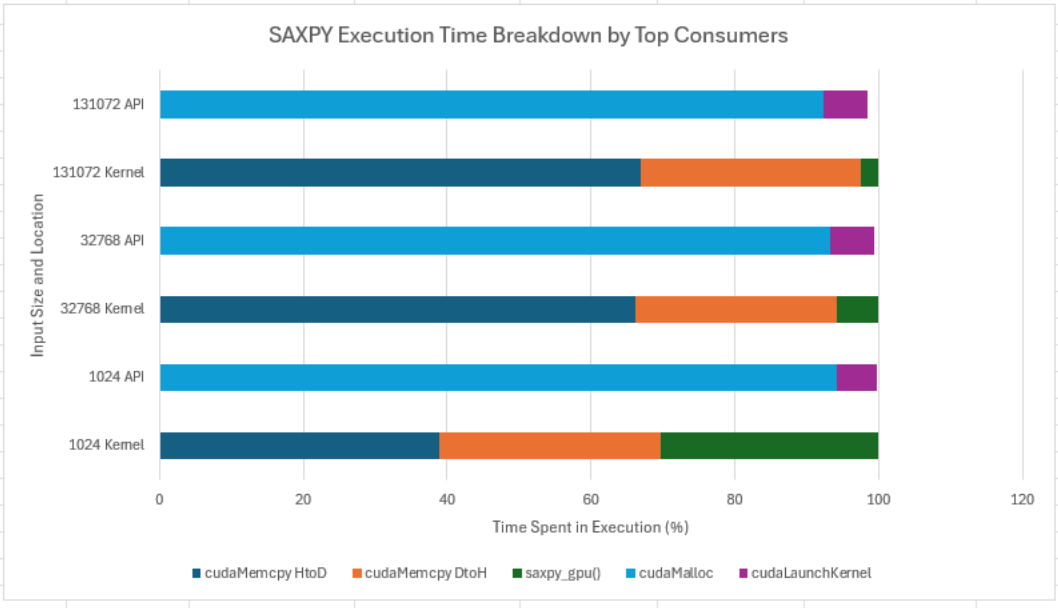


Table 2: SAXPY Kernel and API Execution Time for Various Input Sizes

The breakdown of top time consumers for Kernel and API for each input size is shown below in Graph 1. The biggest consumers across input size were memory operations. For the API cudaMalloc always took the most time and for the Kernel cudaMemcpy HtoD was the biggest consumer (though to a lesser extent).

	cudaMemcpy HtoD	cudaMemcpy DtoH	saxpy_gpu()	cudaMalloc	cudaLaunchKernel
1024 Kernel	38.94	30.77	30.29	0	0
1024 API	0	0	0	94.21	5.46
32768 Kernel	66.21	27.93	5.86	0	0
32768 API	0	0	0	93.32	6.1
131072 Kernel	66.99	30.55	2.46	0	0
131072 API	0	0	0	92.32	6.04



Graph 1: SAXPY Execution Time Breakdown by Top Consumers

Monte-Carlo Execution:

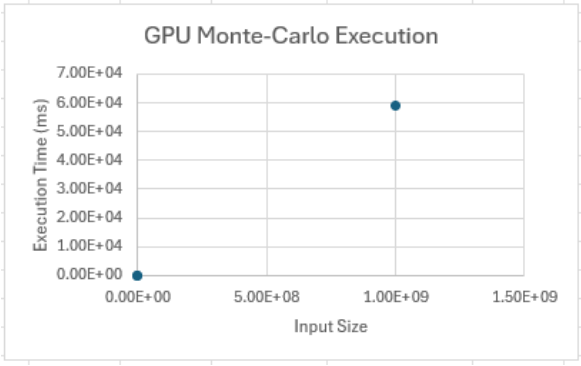
The raw data from profiling the Monte-Carlo program is shown below in Table 3. For three input sizes (sampleSize), the data ranges from 1000 to 1E9. The time execution took increased greatly between 1E6 and 1E9 but not between 1000 and 1E6.

Monte-Carlo Program				
Input Size	Kernel / API Call	Execution Time (ms)	Percentage of Total Time (%)	
1.00E+03	cudaMemcpy DtoH	2.30E-03	1.06	GPU
1.00E+03	reduceCounts()	8.45E-03	3.9	API
1.00E+03	generatePoints()	2.06E-01	95.03	
1.00E+03	cudaMalloc	104.48	93.96	
1.00E+03	cudaMemcpy	6.3282	5.69	
1.00E+06	cudaMemcpy DtoH	3.24E+00	4.88	
1.00E+06	generatePoints()	6.32E+01	95.11	
1.00E+06	cudaMalloc	108.19	59.33	
1.00E+06	cudaMemcpy	67.58	37.06	
1.00E+06	cudaLaunchKernel	6.2055	3.4	
1.00E+09	cudaMemcpy DtoH	2.67E+03	4.52	
1.00E+09	generatePoints()	5.63E+04	95.46	
1.00E+09	cudaMalloc	109.34	0.19	
1.00E+09	cudaMemcpy	5.89E+04	99.79	

Table 3: Monte-Carlo Raw Data from nvprof Profiling Tool

This gap in size can be shown clearly in Table 4. The smaller two input sizes merged into one data point near the origin, dwarfed by the larger input size and it's much longer execution time. Interestingly, this program shows that varying sampleSize in the code affects both API and Kernel unlike the SAXPY program which largely only affected Kernel calls.

GPU		API	
Input Size	Execution Time	Input Size	Execution Time
1.00E+03	2.16E-01	1.00E+03	110.8082
1.00E+06	6.64E+01	1.00E+06	181.9755
1.00E+09	5.89E+04	1.00E+09	59035.54



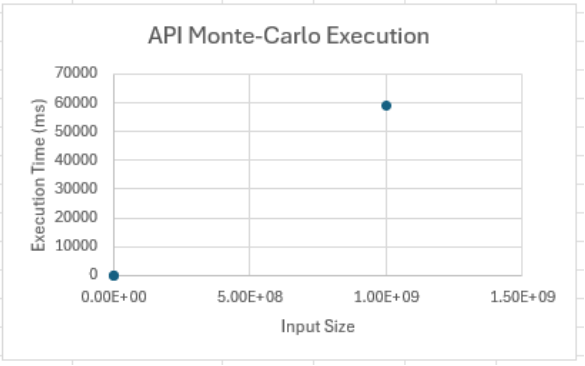
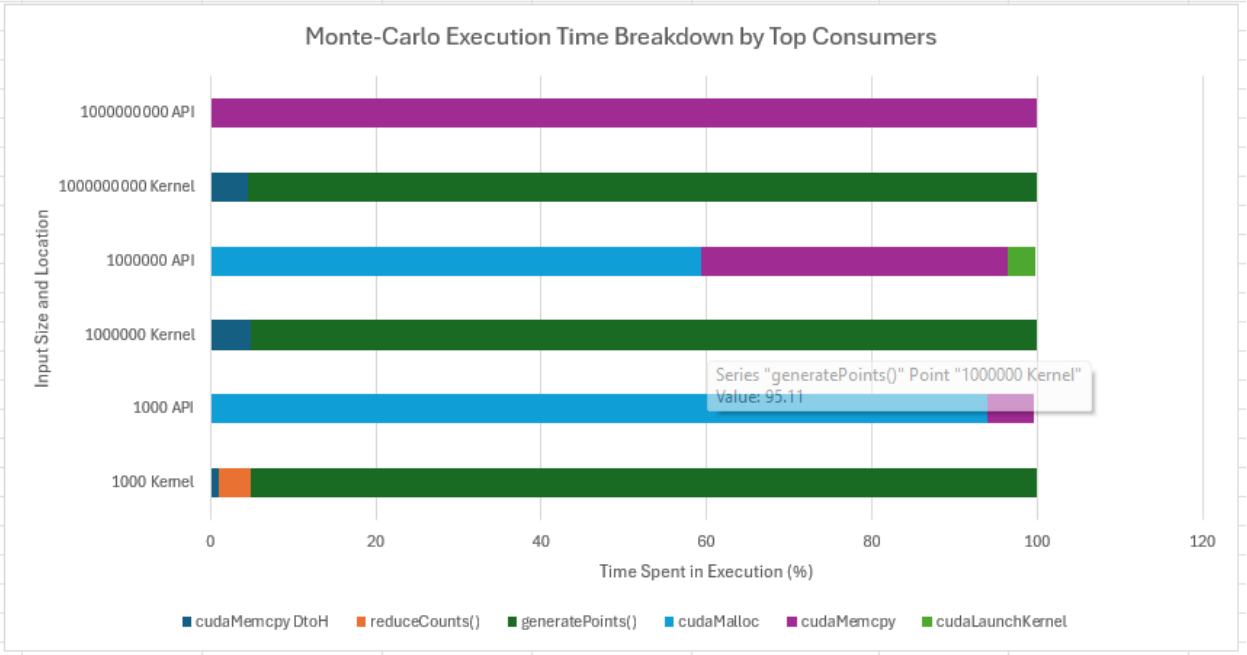


Table 4: Monte-Carlo Kernel and API Execution Time for Various Input Sizes

The breakdown of top time consumers for Kernel and API for each input size in the Monte-Carlo program is shown below in Graph 2. The biggest consumers across input size were not always memory operations unlike SAXPY. For the API cudaMalloc started out as the biggest consumer of time but then steadily decreases with input size and cudaMemcpy took over. For the Kernel generatePoints() was the biggest consumer every time.

	cudaMemcpy DtoH	reduceCounts()	generatePoints()	cudaMalloc	cudaMemcpy	cudaLaunchKernel
1000 Kernel	1.06	3.9	95.03	0	0	0
1000 API	0	0	0	93.96	5.69	0
1000000 Kernel	4.88	0	95.11	0	0	0
1000000 API	0	0	0	59.33	37.06	3.4
1000000000 Kernel	4.52	0	95.46	0	0	0
1000000000 API	0	0	0	0.19	99.79	0



Graph 2: Monte-Carlo Execution Time Breakdown by Top Consumers