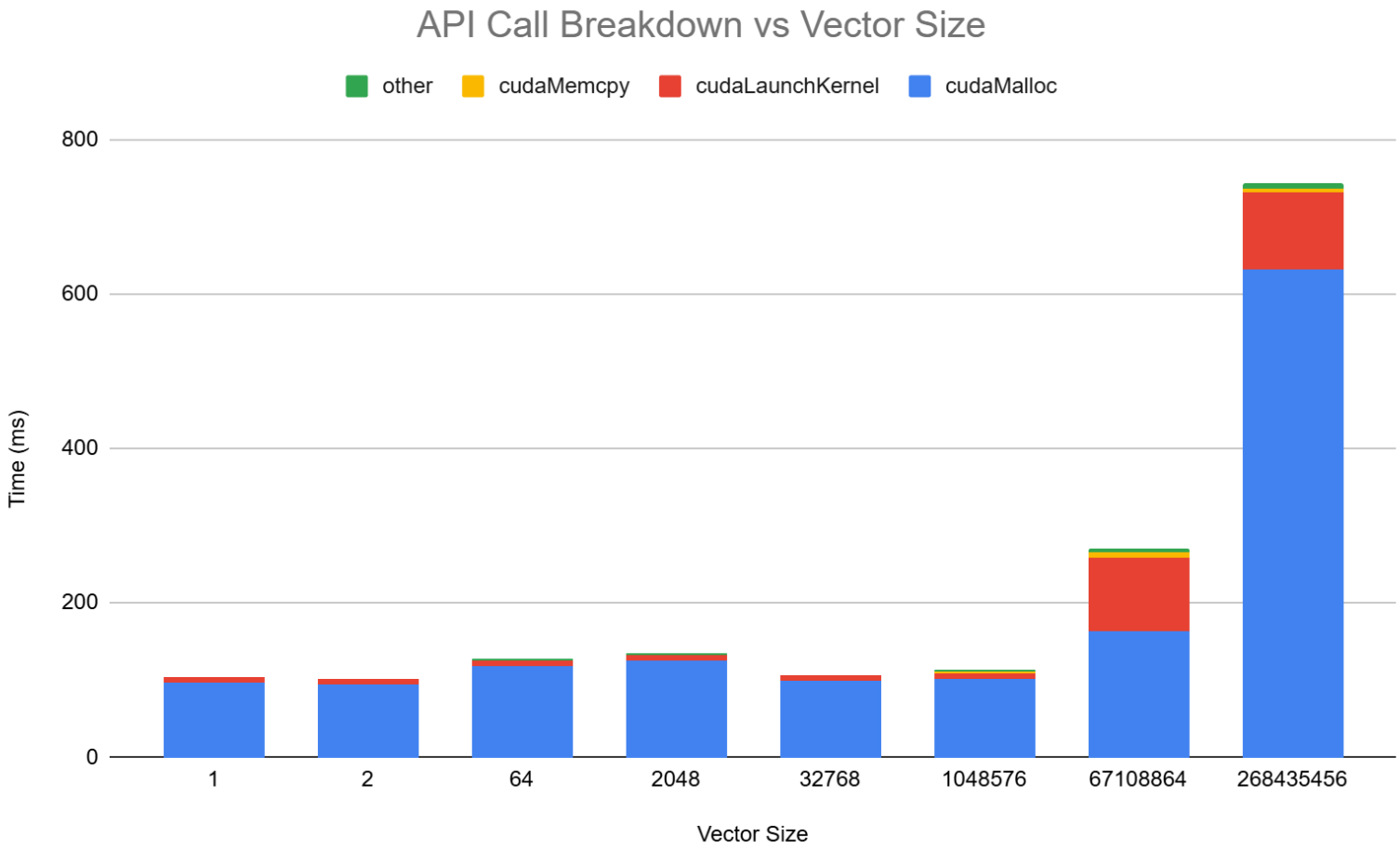
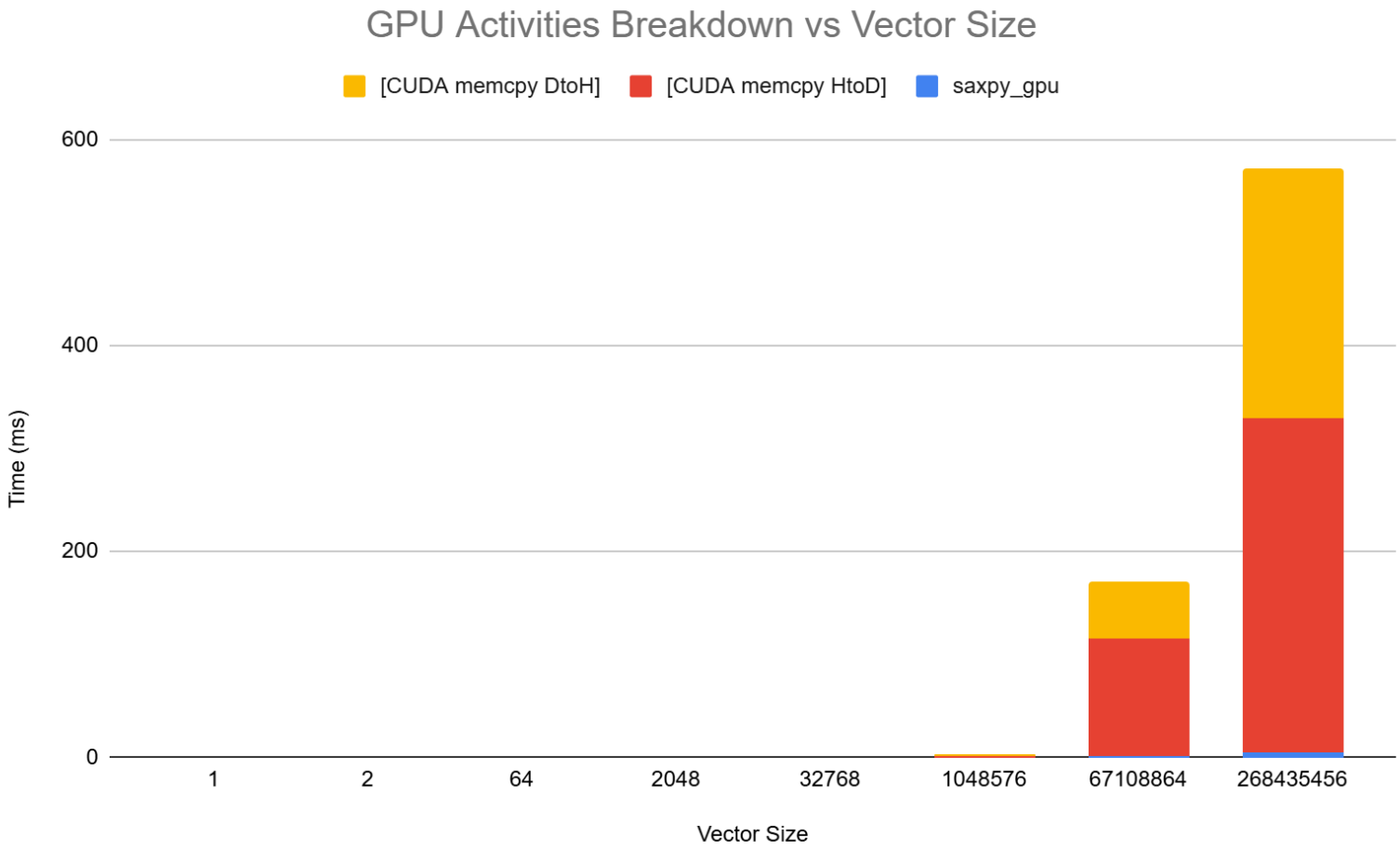
Programming Assignment 1 - Nikita Varfolomeev

Part A

	API calls (ms)				GPU activities (ms)		
vectorSize	cudaMalloc	cudaLaunchKernel	cudaMemcpy	other	saxpy_gpu	[CUDA memcpy HtoD]	[CUDA memcpy DtoH]
1	97.662	5.543	0.142	0.338	0.002	0.002	0.002
2	94.900	5.512	0.138	0.353	0.002	0.002	0.002
64	118.177	6.235	1.314	1.541	0.002	0.002	0.002
2048	124.940	6.888	1.509	1.754	0.002	0.003	0.002
32768	99.328	6.323	0.166	0.442	0.002	0.026	0.011
1048576	101.921	5.581	2.584	2.945	0.015	1.528	0.603
67108864	163.917	95.162	5.770	6.416	1.003	114.840	55.055
268435456	630.650	100.735	5.662	7.139	3.998	324.650	242.910





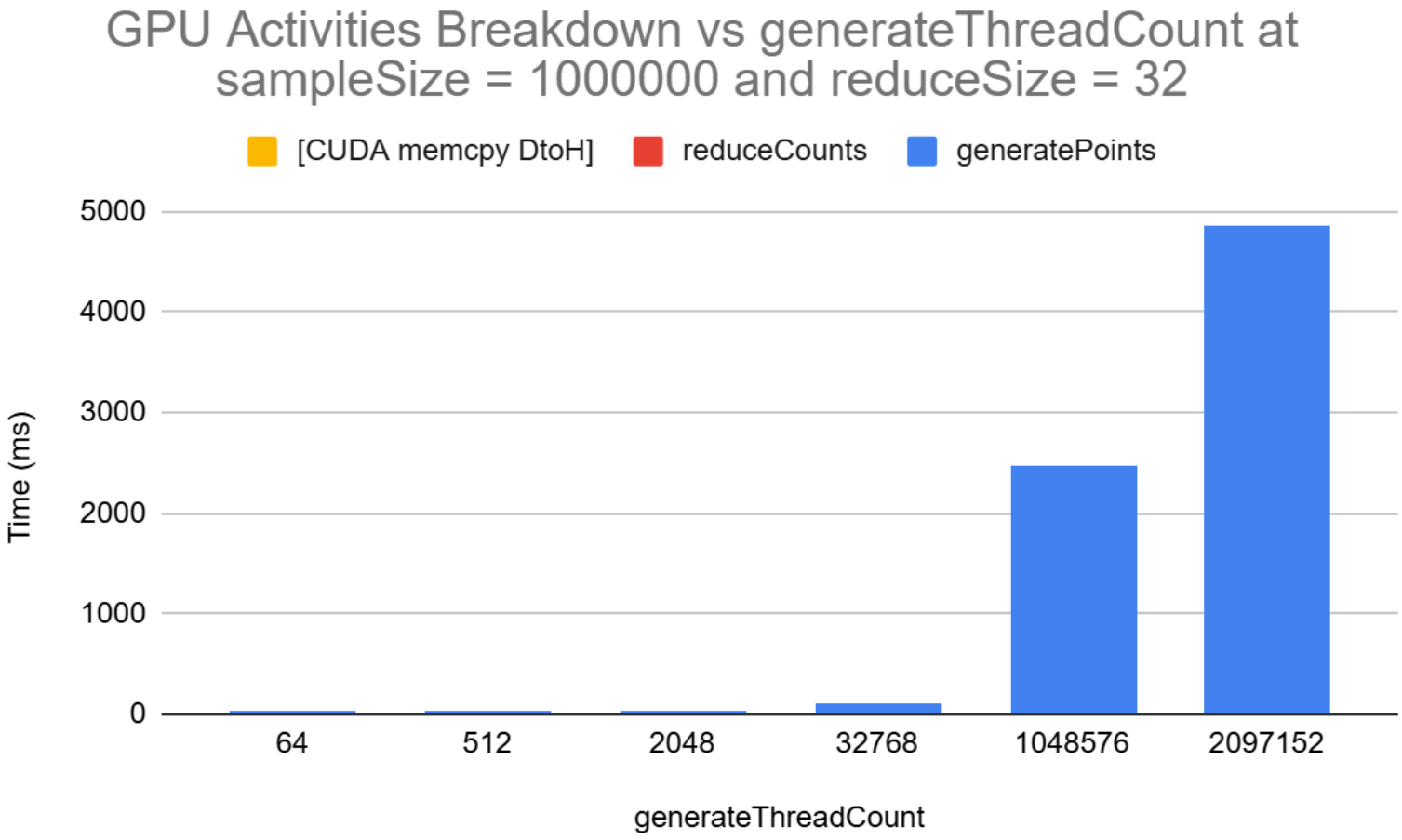
The execution time generally increased as vector size increased. The increase rate was less than linear.
As vector size increased from 1 to 1048576, the execution time remained steady.
When the vector size was increased from 1048576 to 67108864 (64 times), the execution time roughly doubled.
An increase from 67108864 to 268435456 (4 times) led to an increase in execution time of rougly 2.5 times.

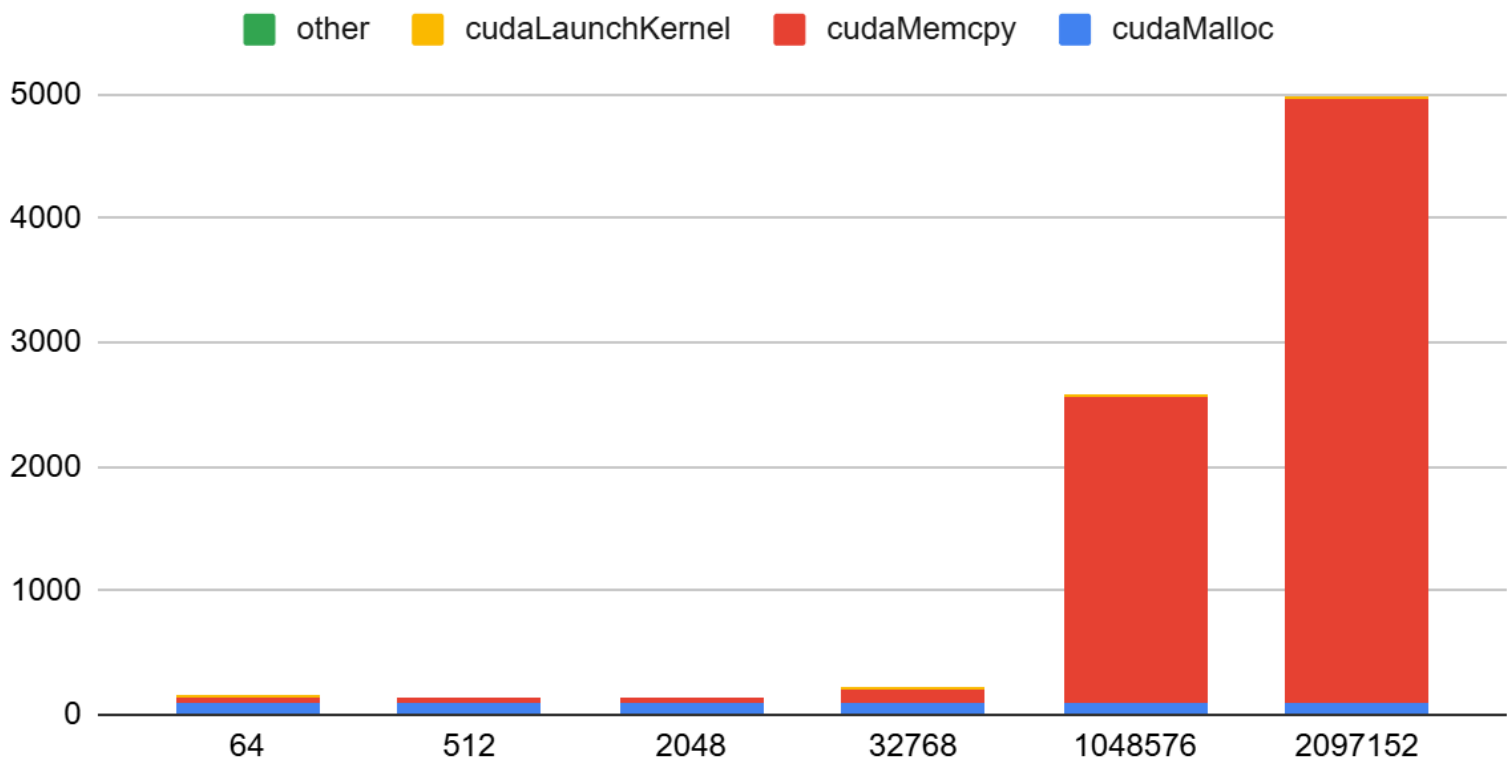
Additionally, the percent of time spent on cudaLaunchKernel increases as vector size increases. It gets progressively more difficult for the GPU to efficiently allocate threads and blocks.

Also, as vector size increased, the distribution of GPU activities changed. With smaller sizes, GPU spent about a third of the time in each of "CUDA memcpy HtoD", "CUDA memcpy DtoH", and "saxpy_gpu". However, as sizes increased, time spent executing saxpy_gpu remained the same, while "CUDA memcpy HtoD" and "CUDA memcpy DtoH" increased.

Part B

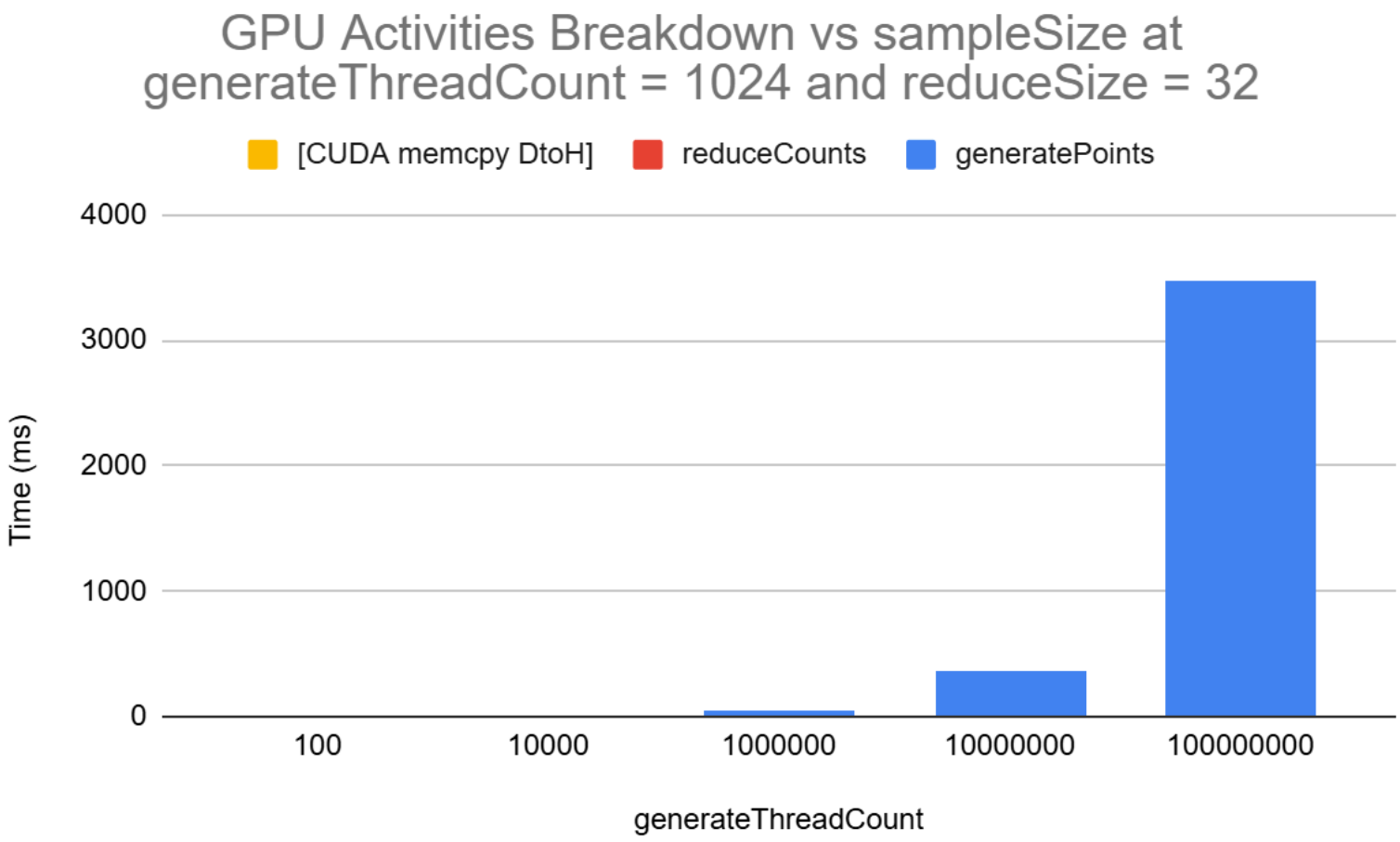
sampleSize = 1000000, reduceSize = 32								
	GPU activities (ms)			API calls (ms)				
generateThreadCount	generatePoints	reduceCounts	[CUDA memcpy DtoH]	cudaMalloc	cudaMemcpy	cudaLaunchKernel	other	total
64	39.008	0.004	0.000	104.500	38.610	6.915	0.309	150.334
512	39.636	0.004	0.000	100.070	39.385	6.120	1.305	146.880
2048	39.992	0.005	0.000	102.080	39.794	6.652	0.253	148.779
32768	109.110	0.005	0.000	103.840	108.900	5.232	0.010	217.870
1048576	2471.960	0.021	0.021	99.197	2472.020	6.481	0.382	2578.080
2097152	4859.250	0.046	0.041	103.110	4859.460	11.675	0.505	4974.750

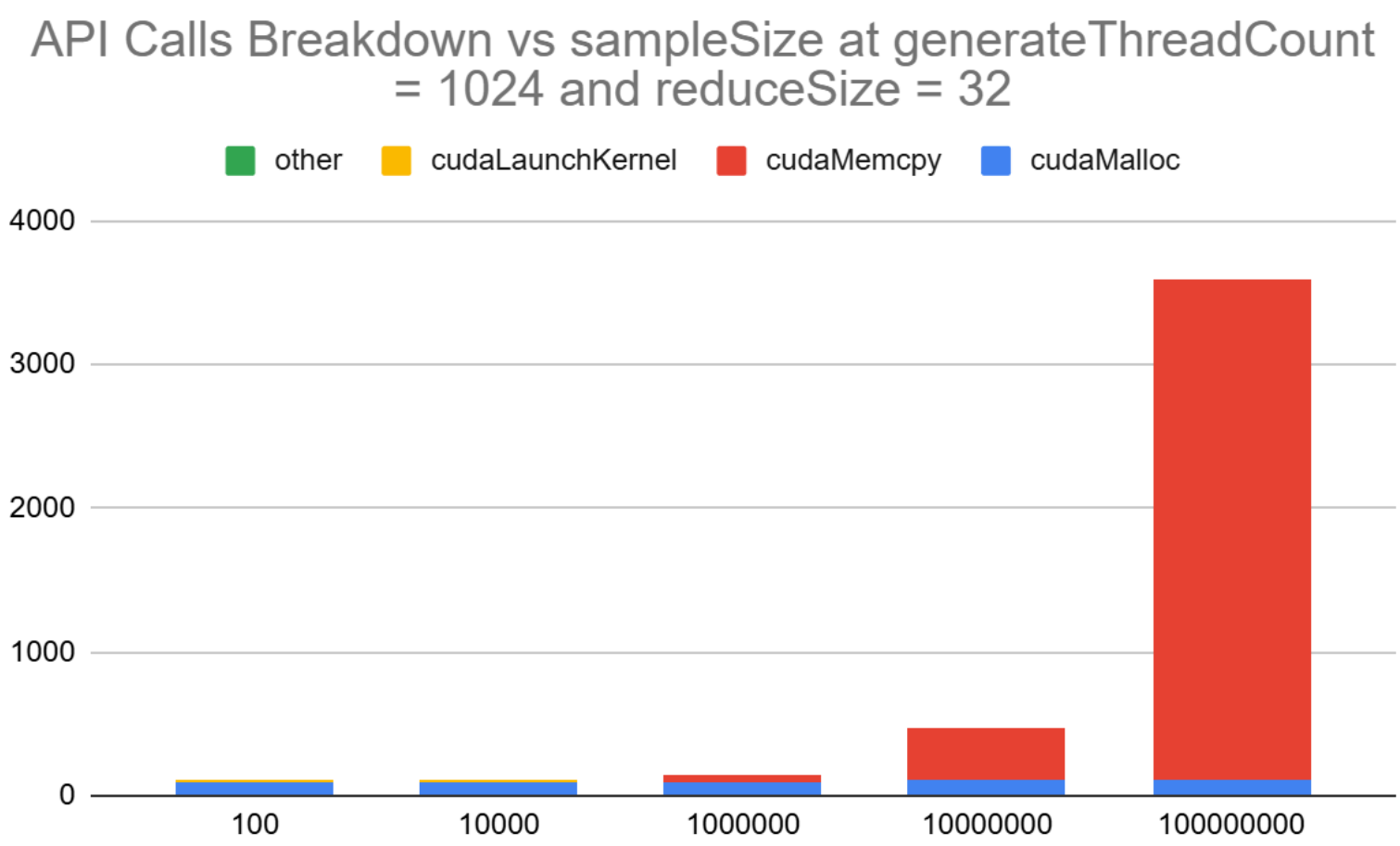




As the number of threads generating numbers grows, total execution time grows. The amount of time gpu spends in generatePoints scales with the number of threads as expected. More threads lead to more total sample points. Without growing the reduceSize, the amount of time API calls spend in cudaMemcpy scales with the number of total sample points.

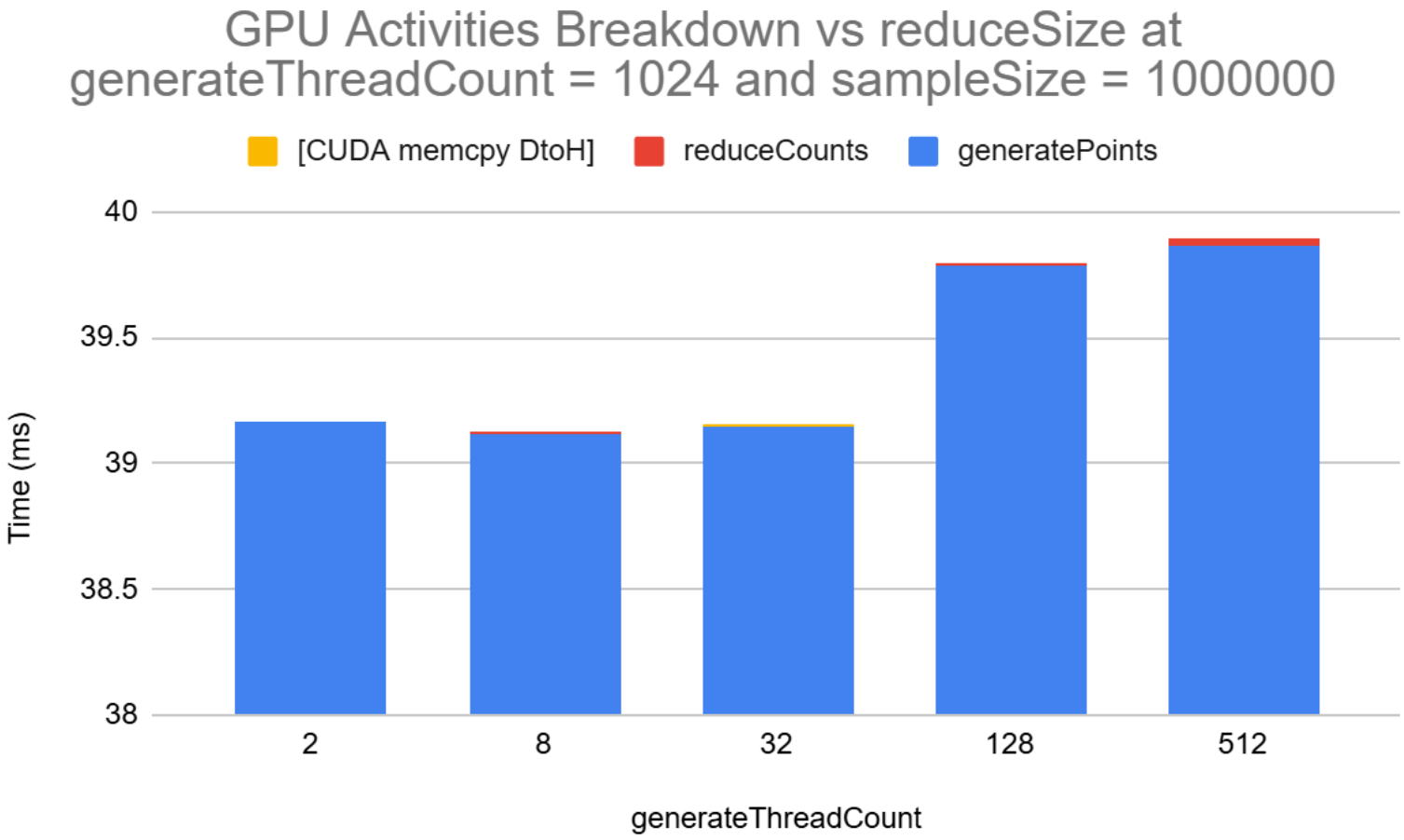
generateThreadCount = 1024, reduceSize = 32								
	GPU activities (ms)			API calls (ms)				
sampleSize	generatePoints	reduceCounts	[CUDA memcpy DtoH]	cudaMalloc	cudaMemcpy	cudaLaunchKernel	other	total
100	0.160	0.005	0.002	97.598	0.071	6.560	0.227	104.456
10000	0.555	0.005	0.002	99.593	0.359	6.389	0.248	106.589
1000000	39.072	0.005	0.002	101.210	38.842	6.799	0.279	147.130
10000000	361.620	0.004	0.002	103.630	361.370	6.180	0.291	471.471
100000000	3475.600	0.004	0.002	109.560	3475.510	6.462	0.258	3591.790

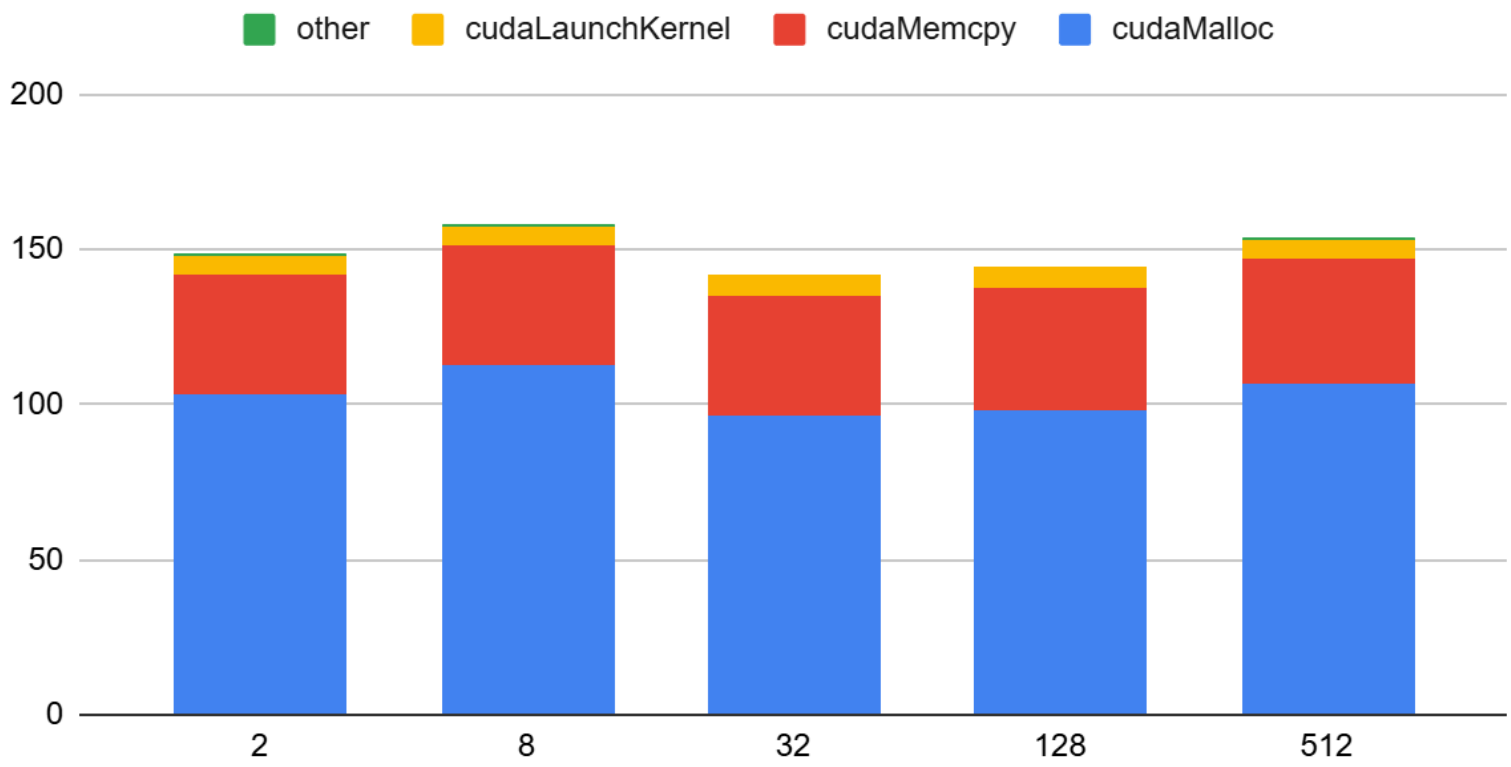




Increasing sampleSize increases the total number of sample points, leading to greater execution times.The results are very similar to the previous test of varying the number of generating threads.

generateThreadCount = 1024, sampleSize = 1000000								
	GPU activities (ms)			API calls (ms)				
reduceSize	generatePoints	reduceCounts	[CUDA memcpy DtoH]	cudaMalloc	cudaMemcpy	cudaLaunchKernel	other	total
2	39.166	0.002	0.002	103.020	38.941	6.177	0.815	148.953
8	39.121	0.003	0.002	112.340	38.861	6.474	0.307	157.982
32	39.146	0.005	0.002	96.185	38.934	6.760	0.253	142.132
128	39.784	0.010	0.002	98.338	39.533	6.740	0.306	144.917
512	39.862	0.032	0.002	107.060	39.716	6.612	0.222	153.610





The total number of points generated stays constant in this test. The reduceSize changes. The amount of time the GPU spends in generatePoints stays constant. The amount of time the GPU spends in reducePoints increases with the increase in reduceSize. There does not seem to be an impact on the total execution time or the distribution of the API calls. Increasing the total number of samples has a much stronger impact on it.