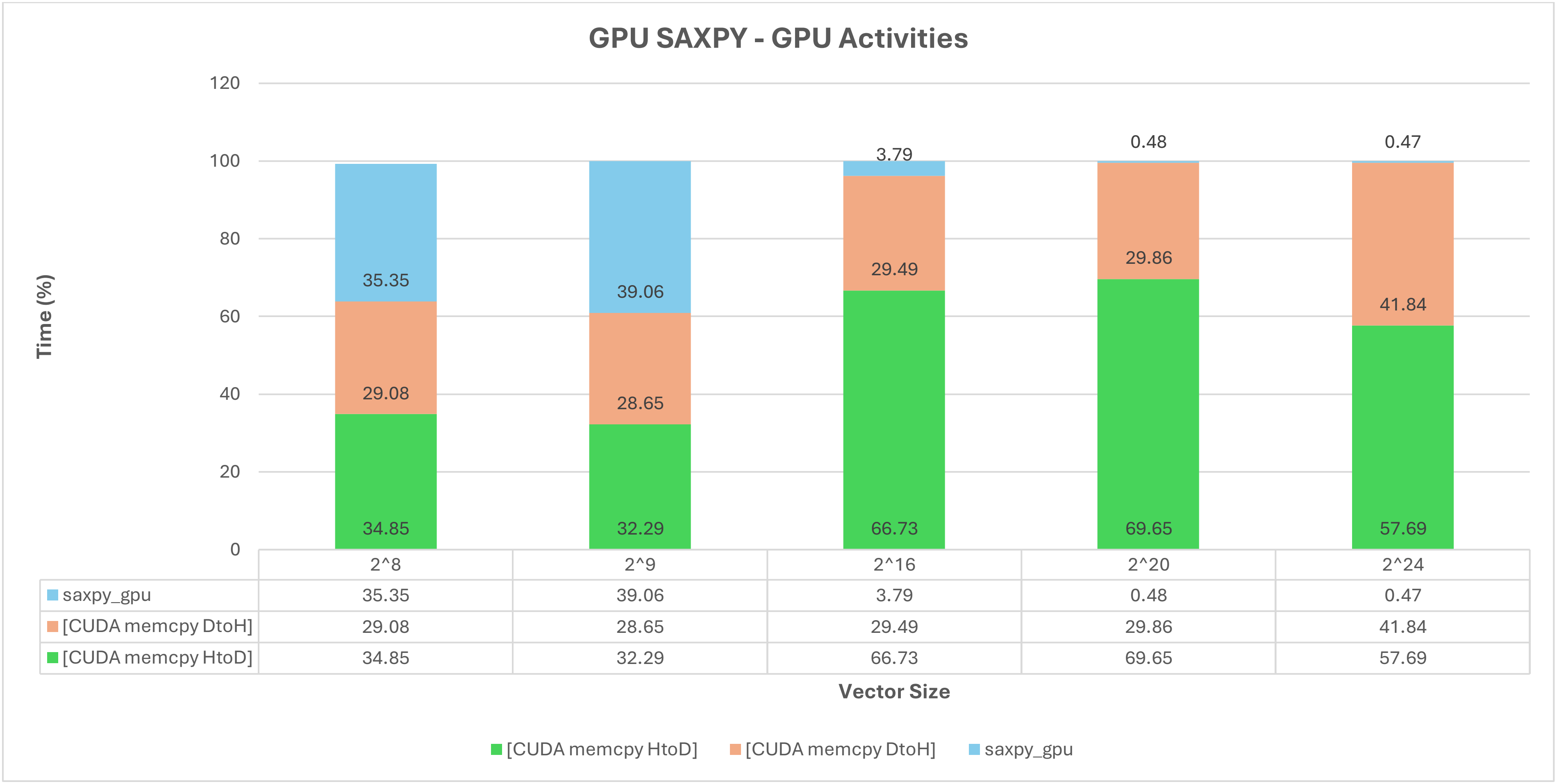
ECE 60827 – Programming Assignment 1

Section A: GPU SAXPY

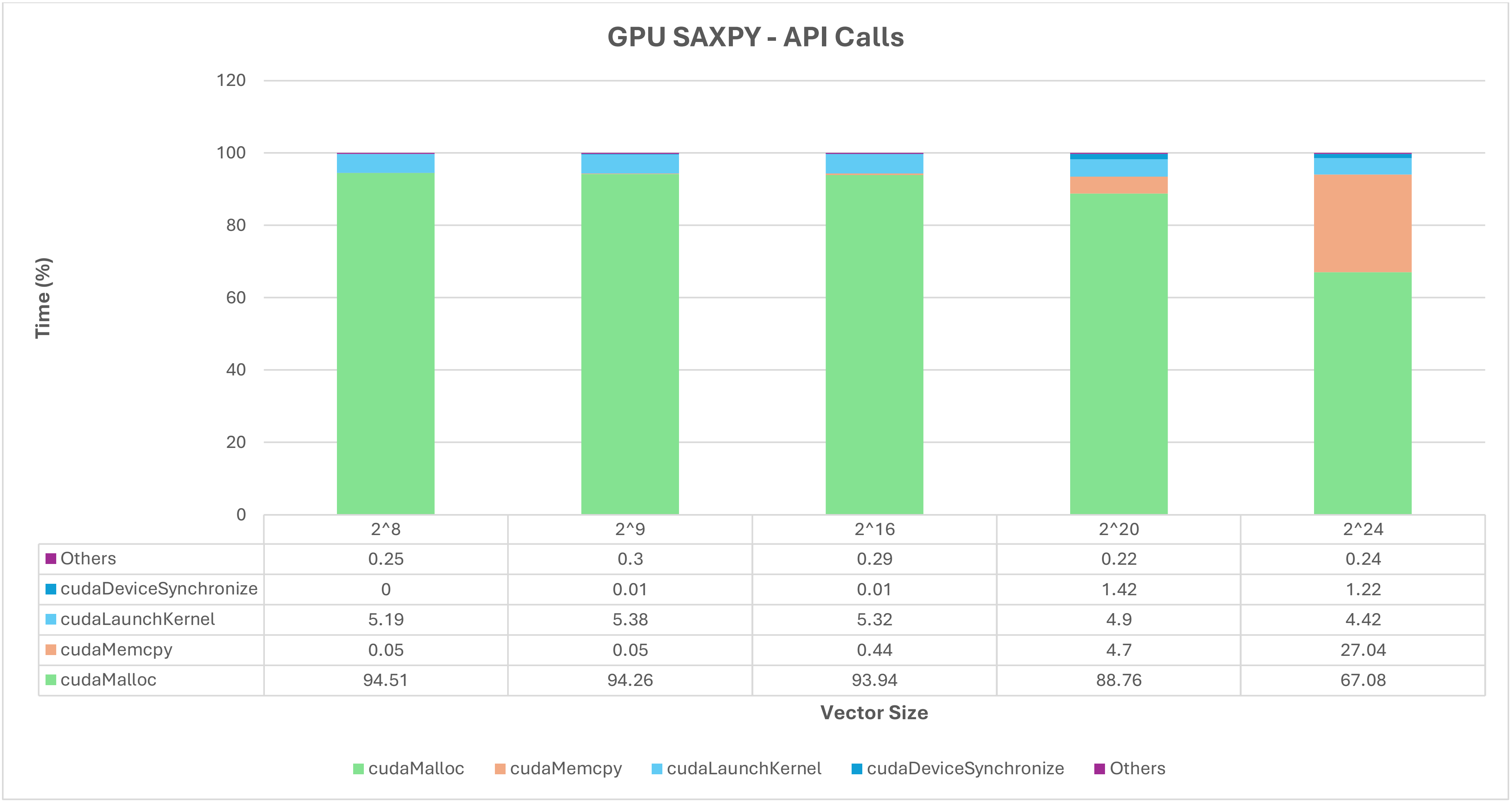
1. The vector length of the input was varied, and the following graphs were plotted from profiling.



	2^8	2^9	2^16	2^20	2^24
saxpy_gpu	35.35	39.06	3.79	0.48	0.47
[CUDA memcpy DtoH]	29.08	28.65	29.49	29.86	41.84
[CUDA memcpy HtoD]	34.85	32.29	66.73	69.65	57.69

Observations:-

1. We can observe that the percentage of execution time taken by the “saxpy_gpu” kernal decreases exponentially with increase in vector size, as the workload now becomes memory bound due to large movement of data between Device memory and host memory.



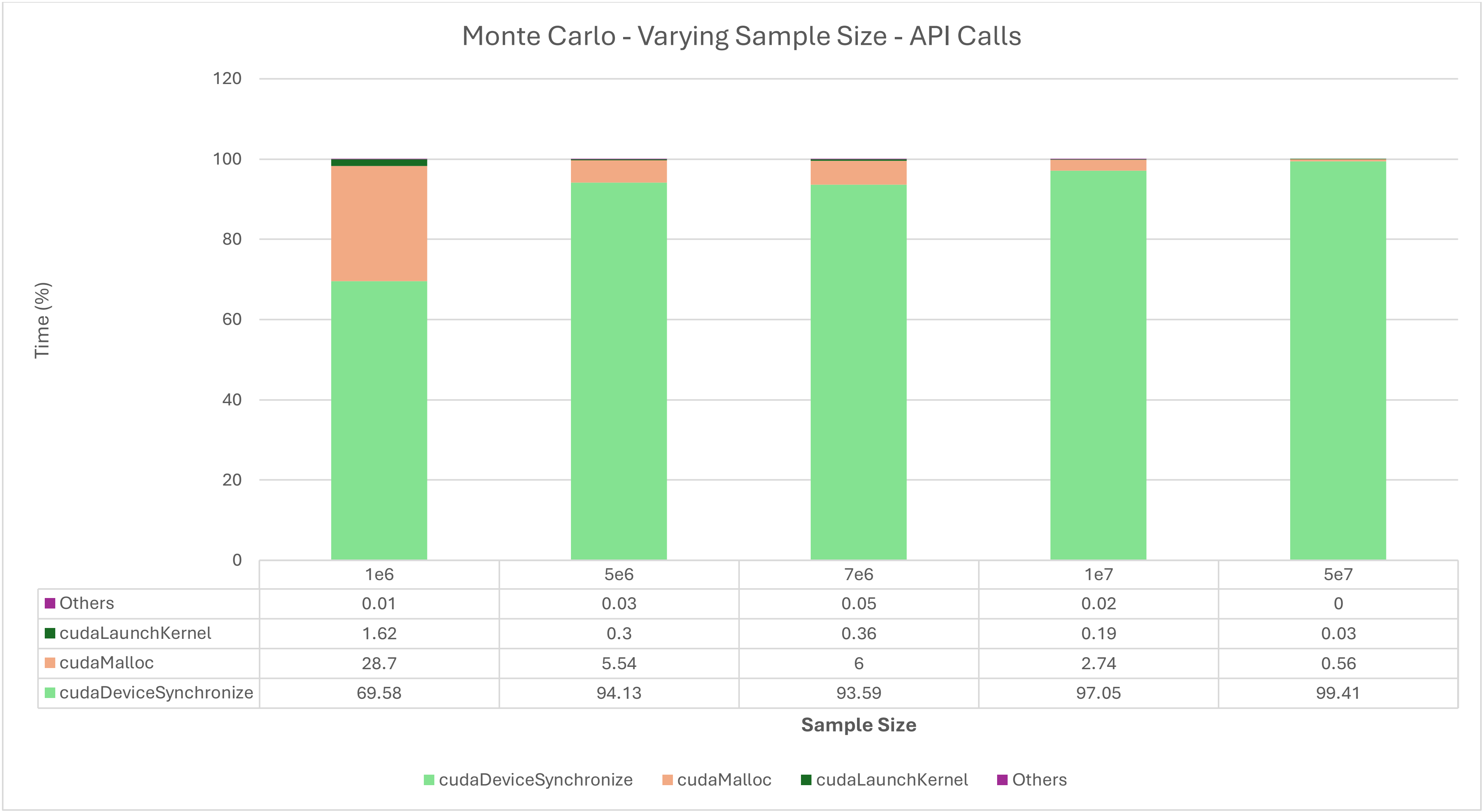
	2^8	2^9	2^16	2^20	2^24
🟪 Others	0.25	0.3	0.29	0.22	0.24
🟦 cudaDeviceSynchronize	0	0.01	0.01	1.42	1.22
🟦 cudaLaunchKernel	5.19	5.38	5.32	4.9	4.42
🟧 cudaMemcpy	0.05	0.05	0.44	4.7	27.04
🟩 cudaMalloc	94.51	94.26	93.94	88.76	67.08

Observations:-

We can again observe that the workload becomes heavily memory bound with increase in vector size as the execution time percentage of “cudaMemcpy” becomes significant in the last case. In addition, we can also observe that “cudaMalloc” takes up the major chunk of execution time across vector sizes. It was also observed that the execution time increased with increase in vector size.

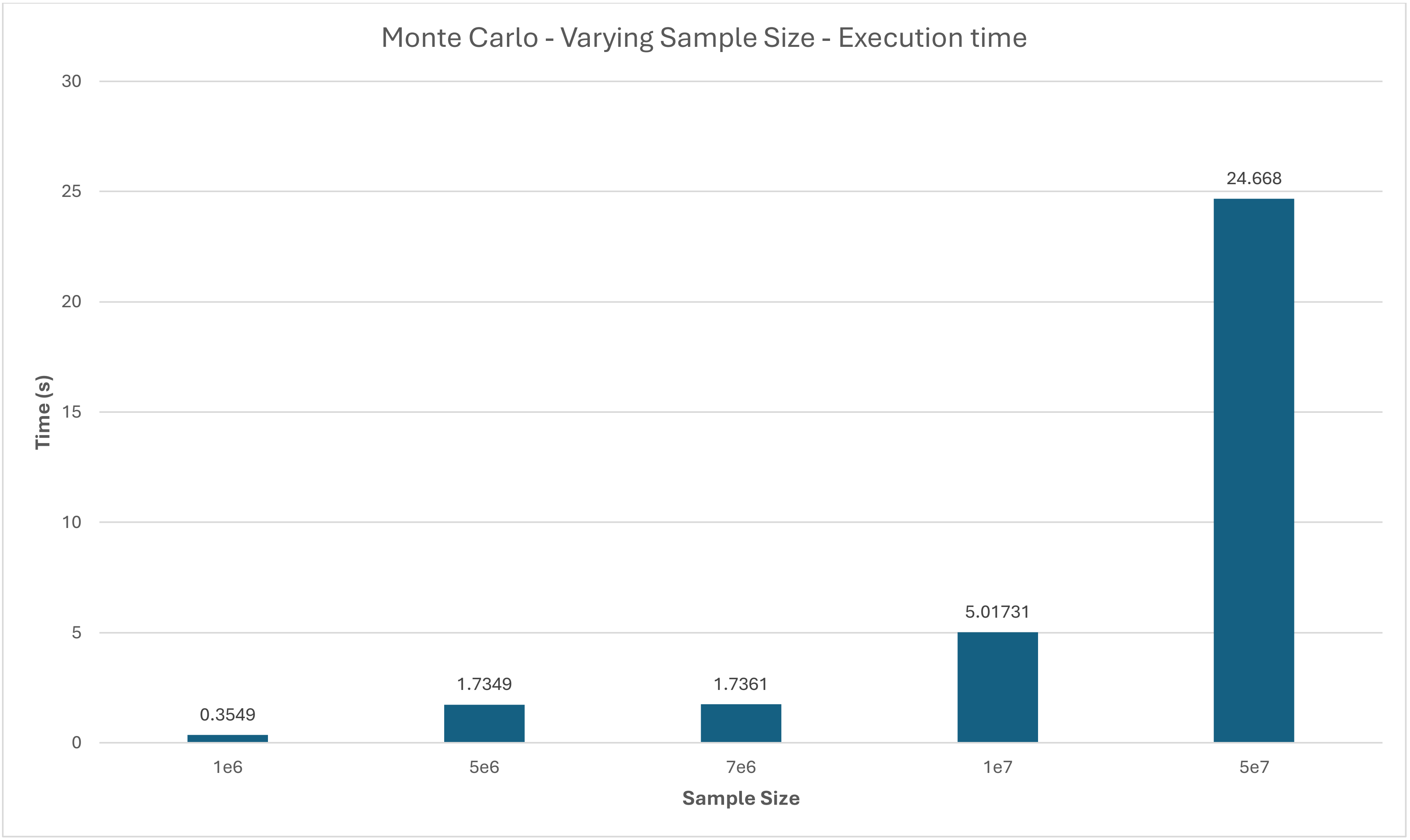
Section B : Monte-Carlo estimation of Pi

1. The following graphs were plotted by varying the “sampleSize” while keeping the “reduceSize = 32” and the “generateThreadCount = 1024” constant.



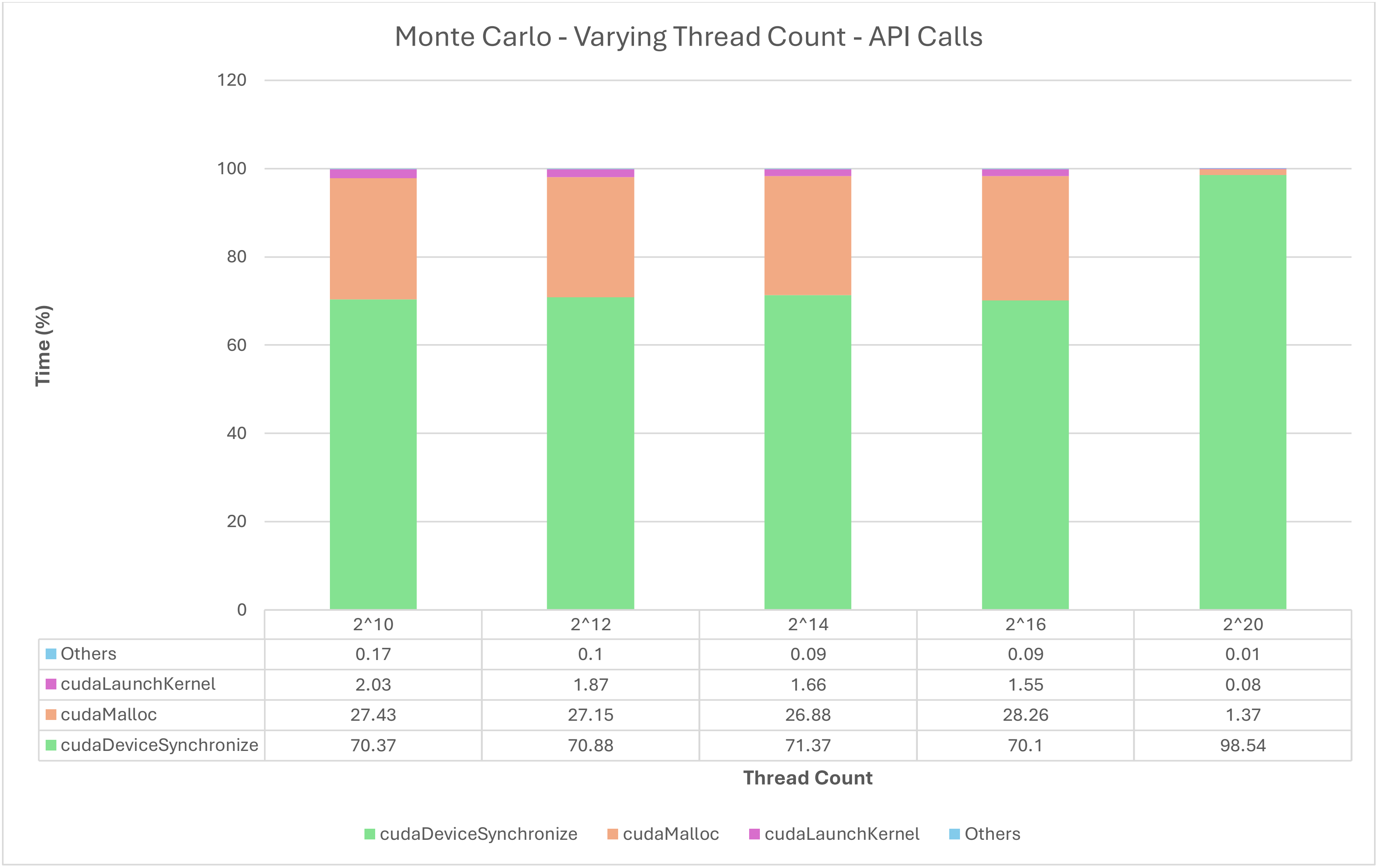
	1e6	5e6	7e6	1e7	5e7
🟪 Others	0.01	0.03	0.05	0.02	0
🟩 cudaLaunchKernel	1.62	0.3	0.36	0.19	0.03
🟧 cudaMalloc	28.7	5.54	6	2.74	0.56
🟩 cudaDeviceSynchronize	69.58	94.13	93.59	97.05	99.41

Observation : We can observe that “cudaDeviceSynchronize” takes up the major chunk of the execution time. This shows the impact of synchronization across threads and motivates the programmer to look for another alternative to avoid it. We can further observe that the “cudaMalloc” takes up the next big chunk. This indicates the fact that the GPUs as only efficient with the cost of memory transaction is amortized over a larger computation. In this case, the computation load is less.



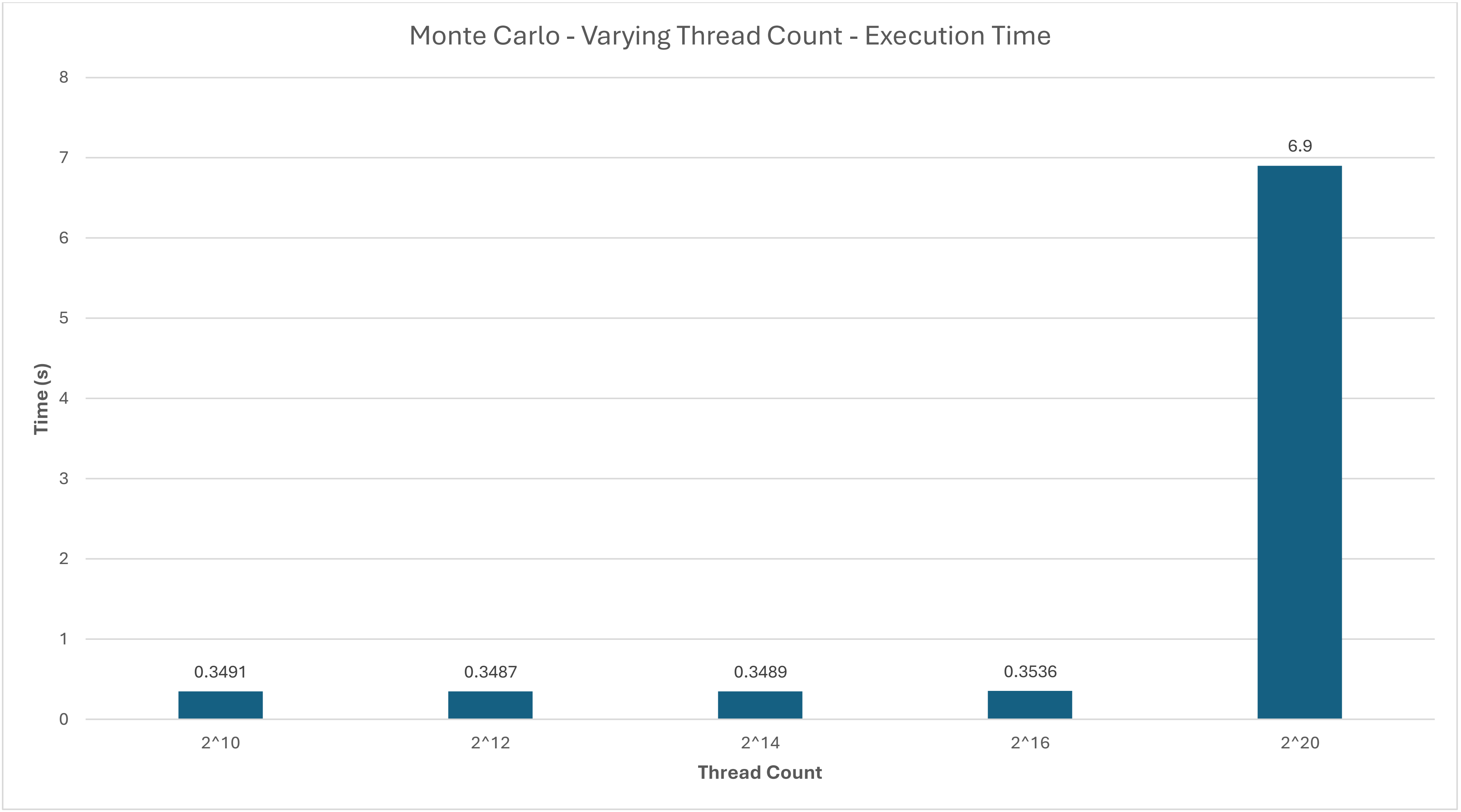
Observation : The graph shows exponential increase in execution time with increase in sample size.

2. The following graphs were plotted by varying the “generateThreadCount” while keeping the “reduceSize = 32” and the “sampleSize = 1e6” constant.



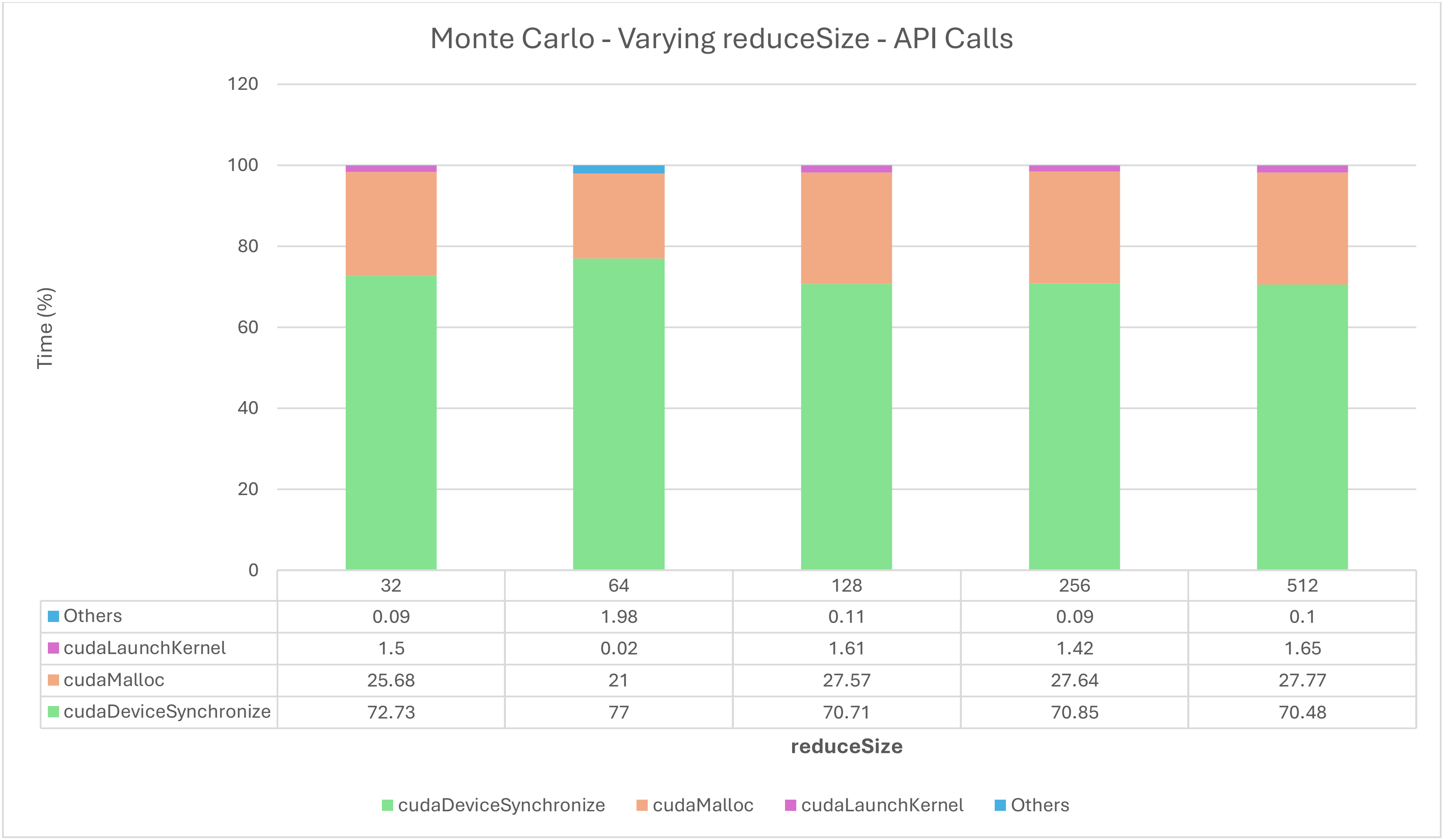
	2^10	2^12	2^14	2^16	2^20
🟦 Others	0.17	0.1	0.09	0.09	0.01
🟪 cudaLaunchKernel	2.03	1.87	1.66	1.55	0.08
🟧 cudaMalloc	27.43	27.15	26.88	28.26	1.37
🟩 cudaDeviceSynchronize	70.37	70.88	71.37	70.1	98.54

Observation : We can observe that the execution time split remains almost constant except the last case where “cudaDeviceSynchronization” becomes predominating. This is expected since increase in “generateThreadCount” is not likely to impact the API calls in any way as they only increase the workload with the same proportion for all the API calls. They contribute to the increase in accuracy of Pi estimate by increasing the number of sample points.



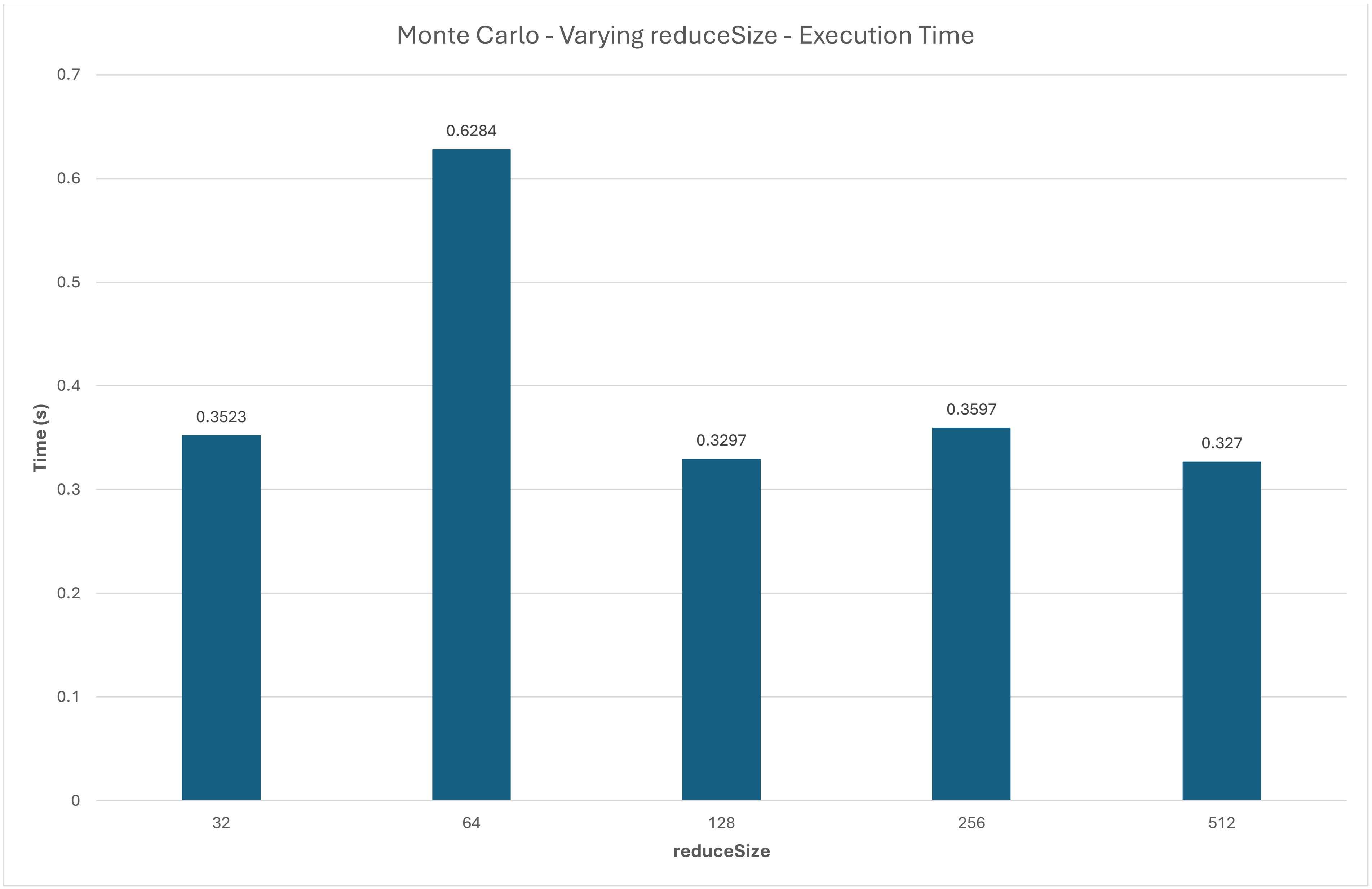
Observation : As expected, increase in “generateThreadCount” leads to increase in execution time. But the reason for the overshoot in the last case is yet to be studied in detail, as it requires a much deeper understanding of the correlation between the GPU Device architecture and programming structure.

2. The following graphs were plotted by varying the “reduceSize” while keeping the “generateThreadCount = 1024” and the “sampleSize = 1e6” constant.



	32	64	128	256	512
🟦 Others	0.09	1.98	0.11	0.09	0.1
🟪 cudaLaunchKernel	1.5	0.02	1.61	1.42	1.65
🟧 cudaMalloc	25.68	21	27.57	27.64	27.77
🟩 cudaDeviceSynchronize	72.73	77	70.71	70.85	70.48

Observation : The split in execution time across various API calls remain almost constant except for the 2nd case where we can see an increase in “cudaDeviceSynchronization” call. Again, a deeper understanding of GPU Device architecture is required to reason out this outlier.



Observation : Again the execution time remain almost constant across various reduce sizes, expect for ‘64’ where an overshoot is observed.