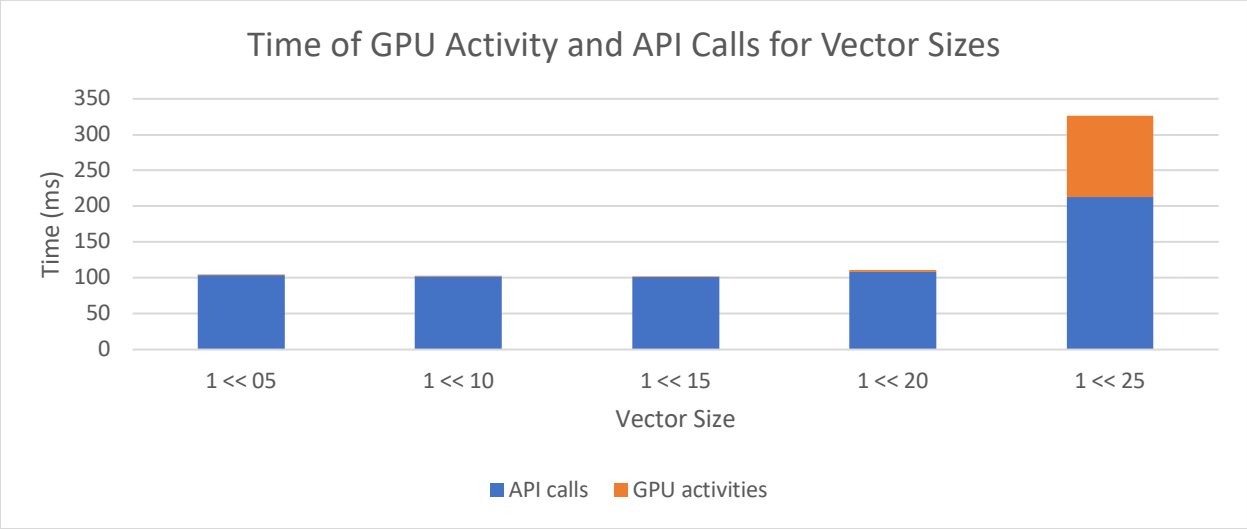
CUDA Assignment 1 – ECE60827

HoSun Choi

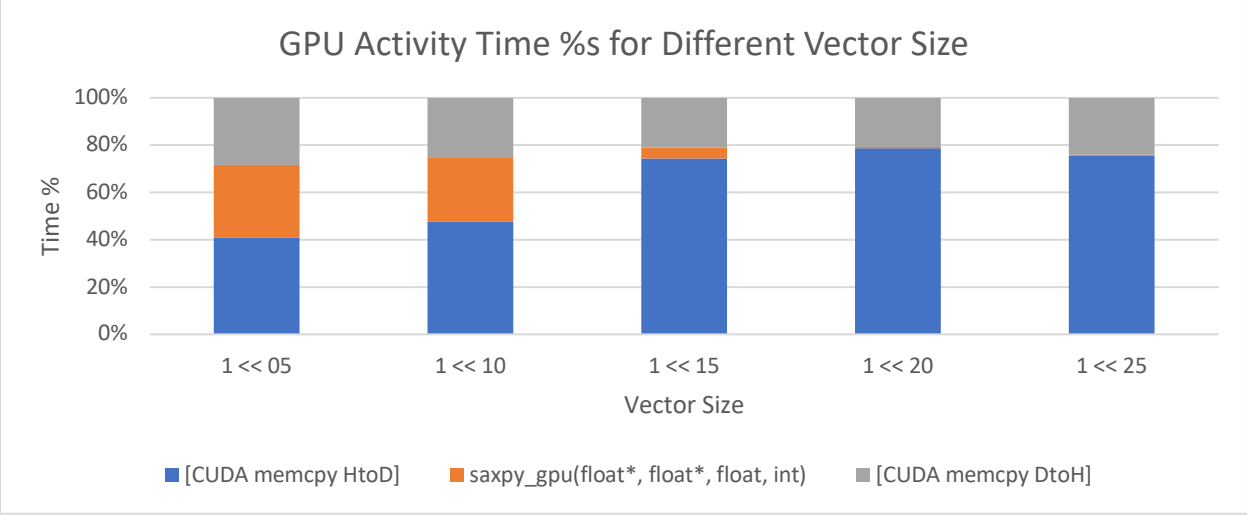
Part A: SAXPY

Vector Size

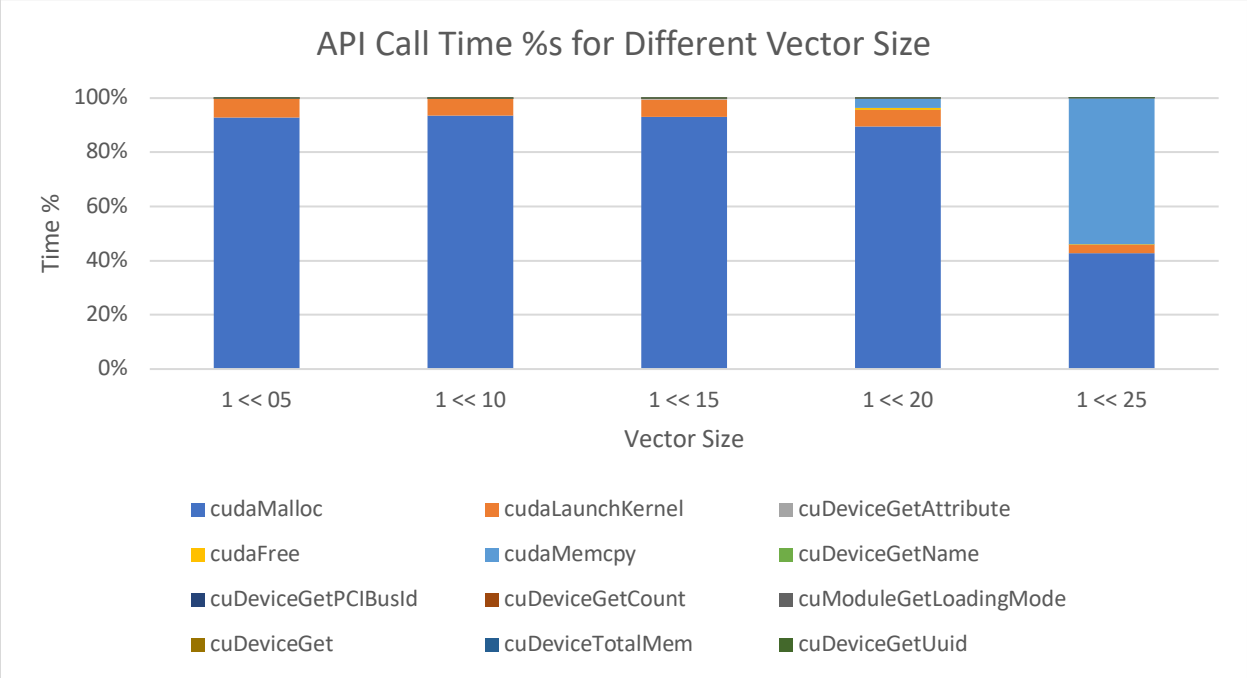
In general, API calls have a larger temporal overhead than the GPU activities for most of the vector sizes. However, once the vector size is significantly large, e.g. $vectorSize = 1 \times 10^{25}$, GPU activity time increases as well.



Within the GPU activities, at lower vector sizes, the SAXPY kernel appears to take a larger portion because copying the data to and from the CPU and GPU is much smaller. However, as the size increases, the SAXPY kernel is significantly lower due to greater memory copying.



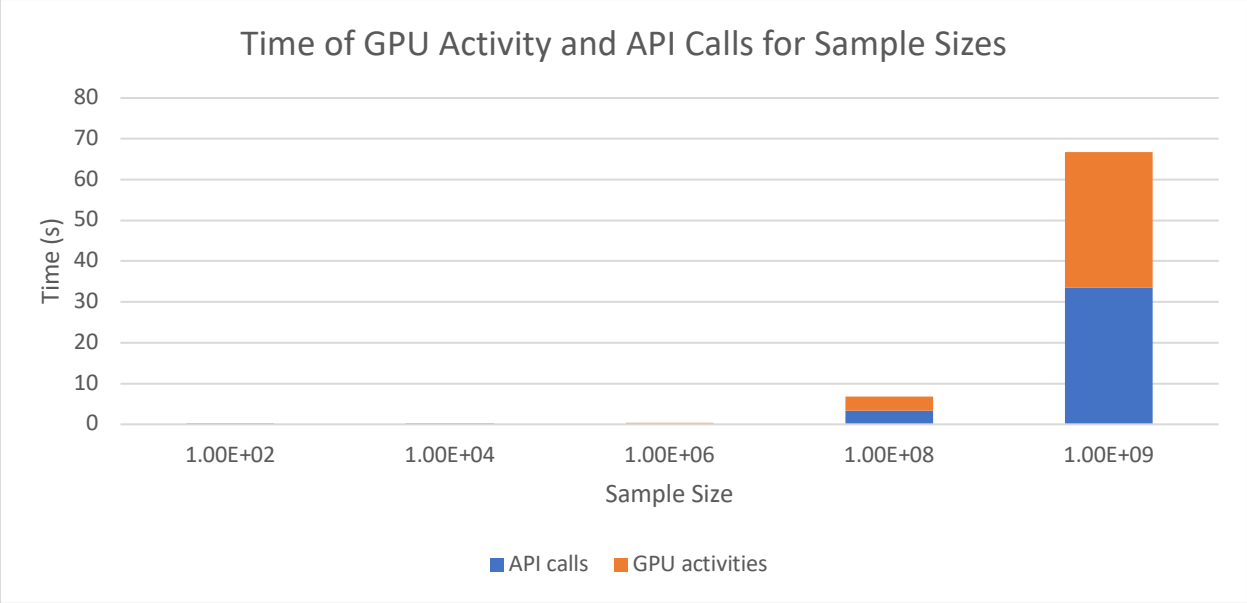
When it comes to the API calls, lower vector sizes have similar time percentages, but once vector sizes become greater, the memory copy API call increases in proportion due to large data copying of slow memory.



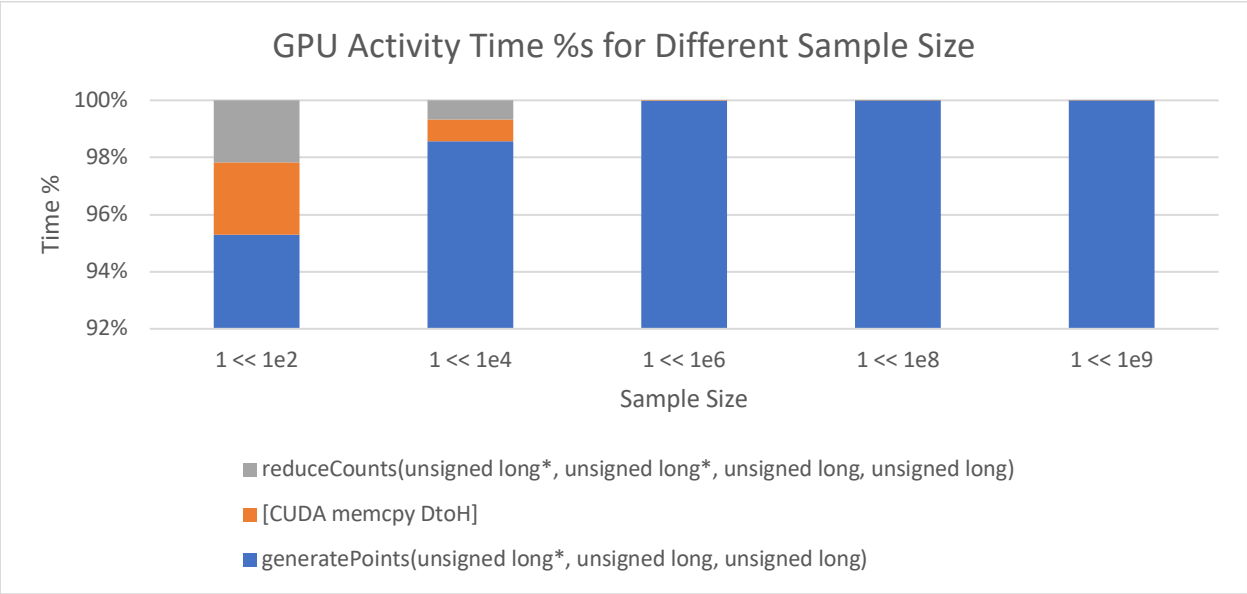
Part B: Monte Carlo Estimation of Pi

Sample Size

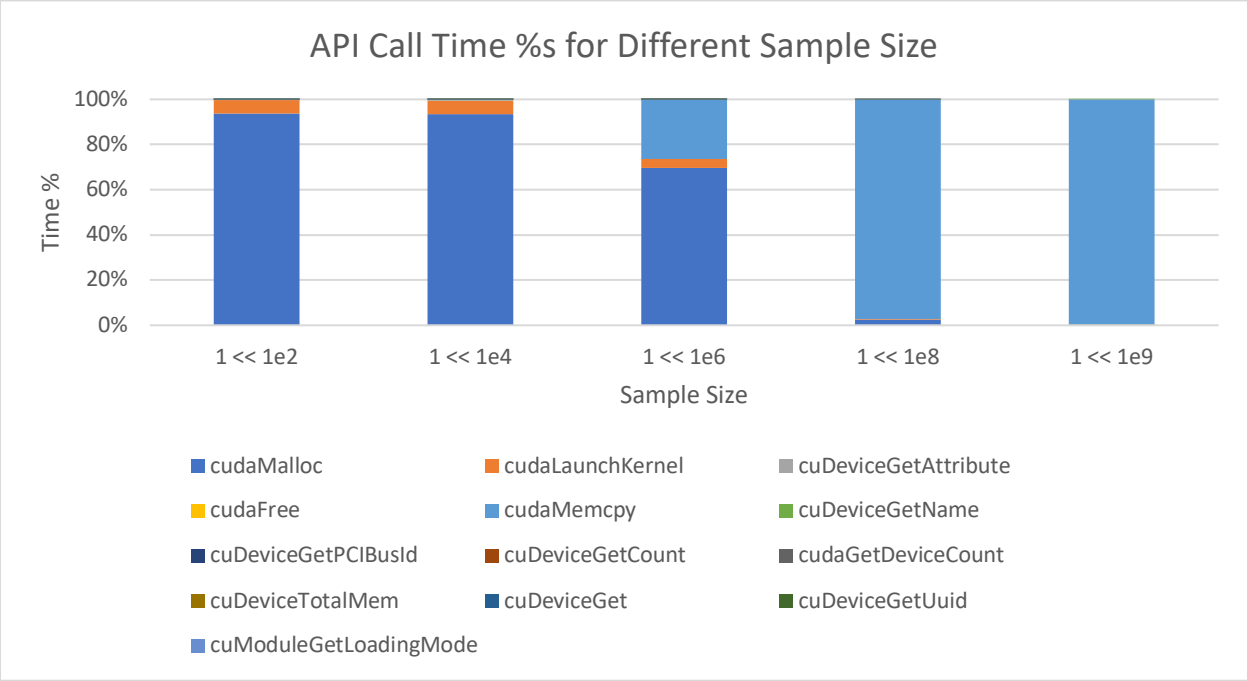
In the Monte Carlo estimation of Pi, increasing the sample sizes increases the overall duration of the program. This is as expected because each thread has to generate more data. The increase is linear despite appearing exponential, which is due to the exponential increase of sample sizes.



With regard to the sample size, at lower number of samples, the activity of generatePoints function takes a smaller proportion. However, as the number of samples increase, it takes up a larger proportion of execution, which is as expected since the loop in the function runs longer.

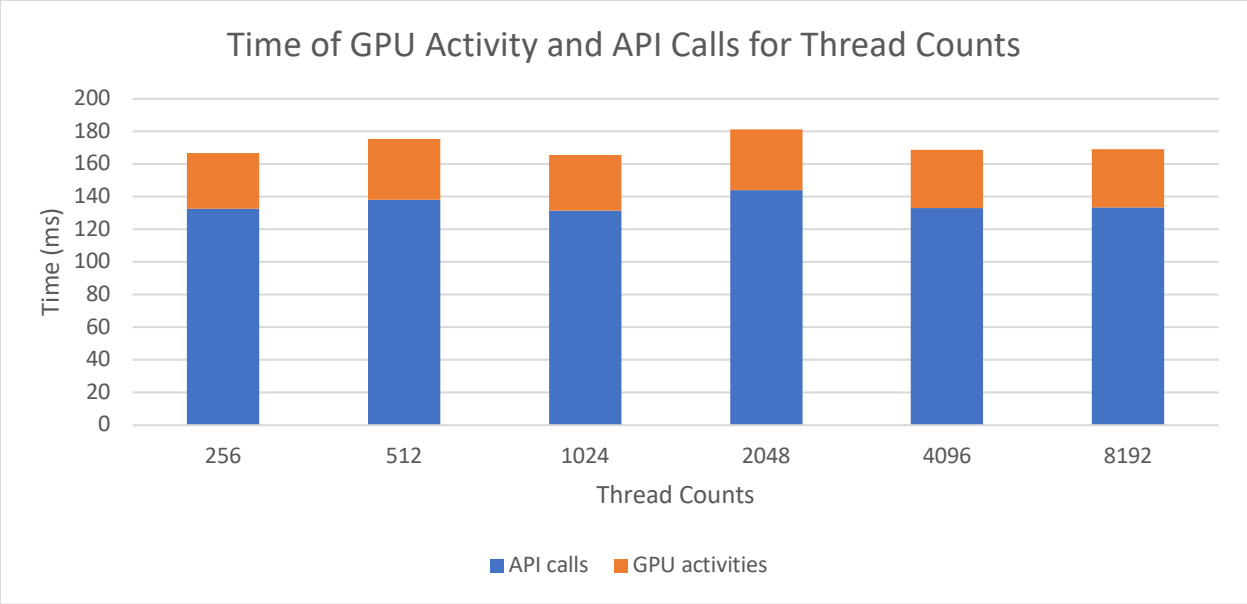


For API Calls, lower sample sizes are dominated by memory allocation of CUDA. However, as the number of samples increase, copying memory begins to dominate. Despite looking at the code for possible sources of memory copying, it is kind of perplexing as increasing the sample size for this code should not increase the proportion of cudaMemcpy since the increase of time should only be from the generatePoints function.

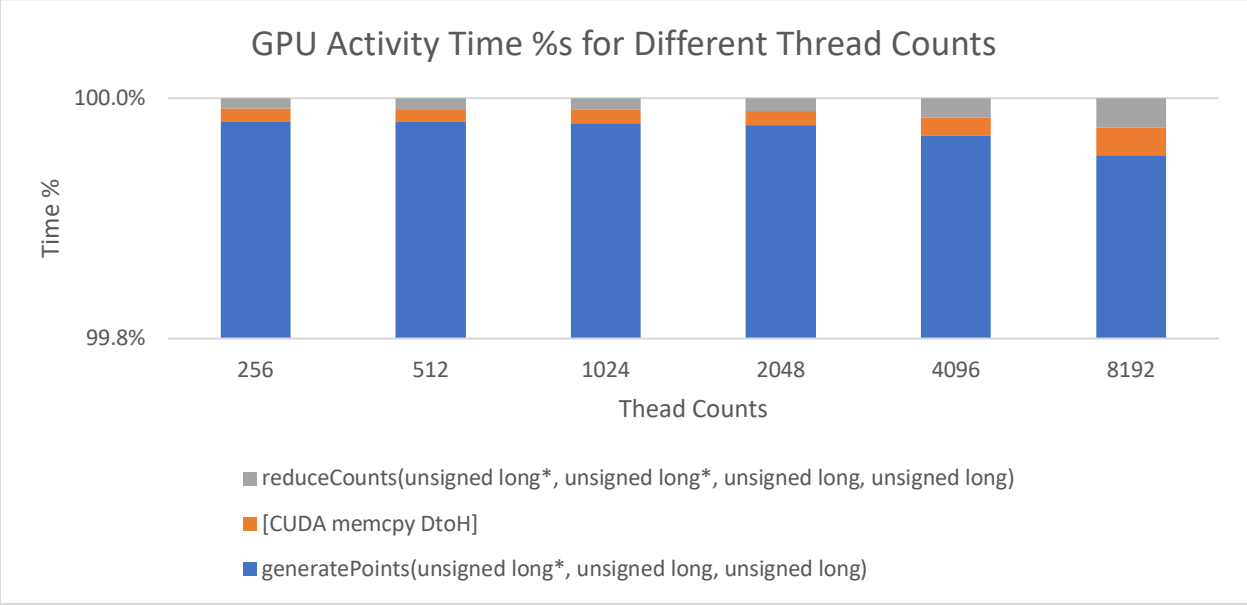


Thread Count

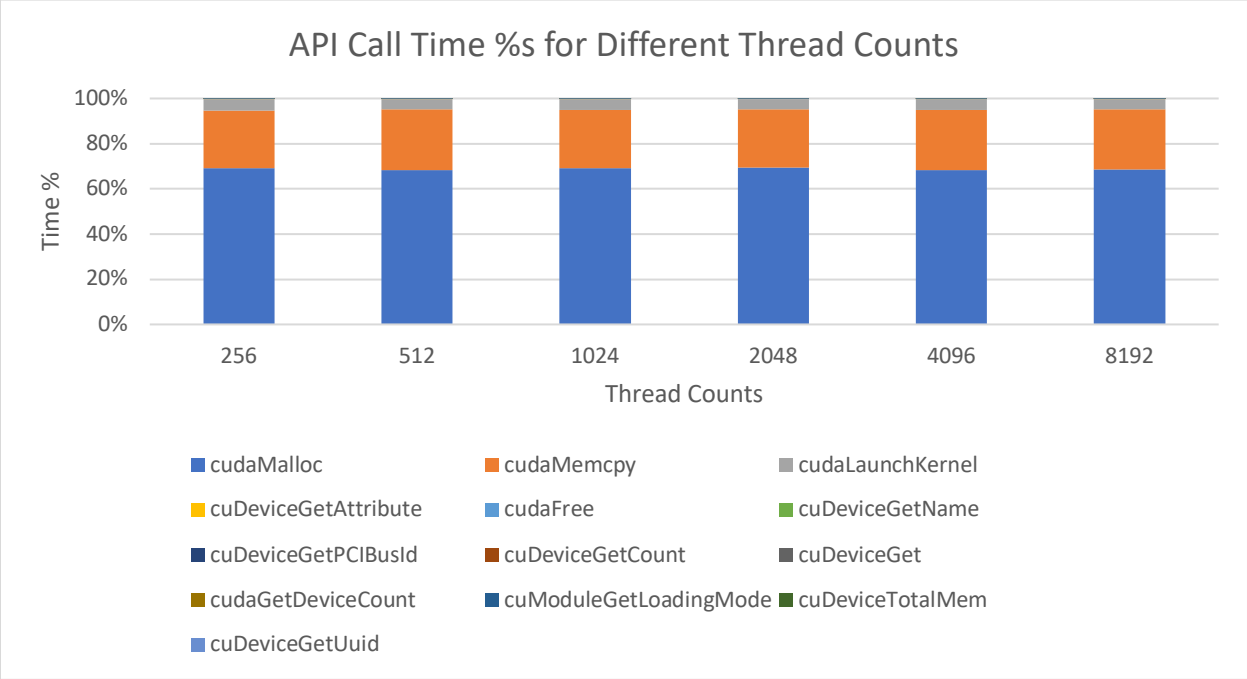
As the number of threads increases, the duration of the Monte Carlo simulation remains relatively constant. I did expect that increasing the number of threads would increase the duration because the number of reduced partial sums should increase, and that would in turn increase the duration for memory copying. However, it seems that up to 8192 threads, the difference is negligible.



On the side of GPU activity, however, the number of threads does increase the proportion of data copying from the GPU to CPU. There also seems to be an increase in reduceCounts function, which may be due to the fixed block size allocation of a single block of 256 threads.

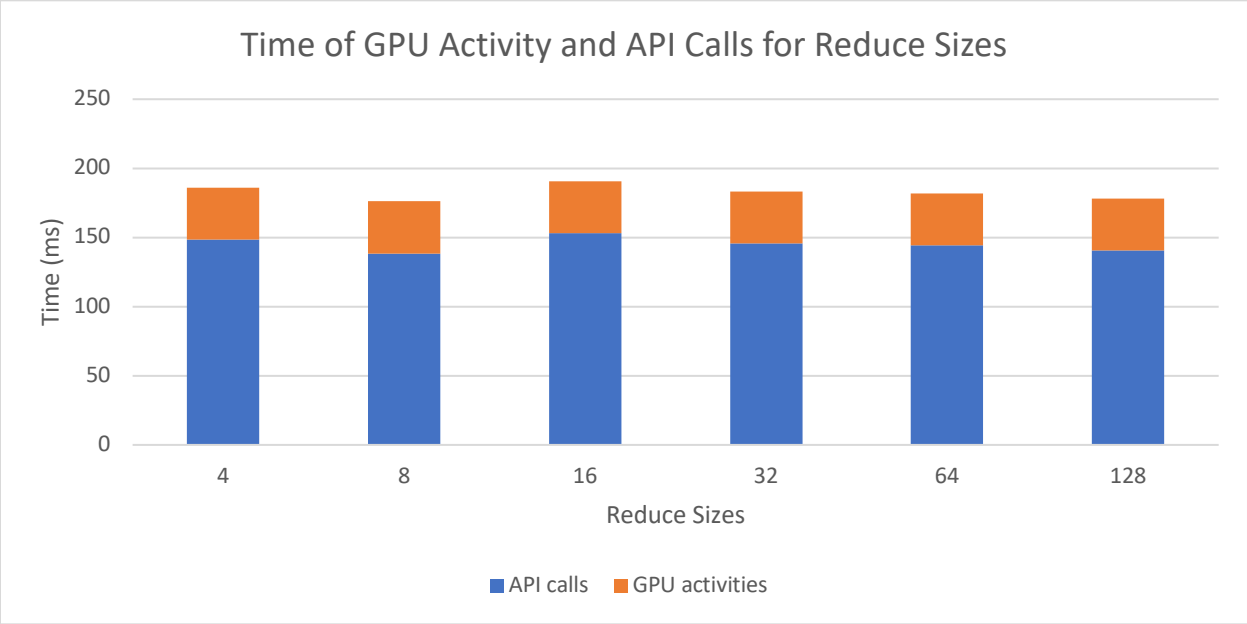


On the side of API calls, however, the time percentage remains relatively same, even though I expected memory copying to increase.

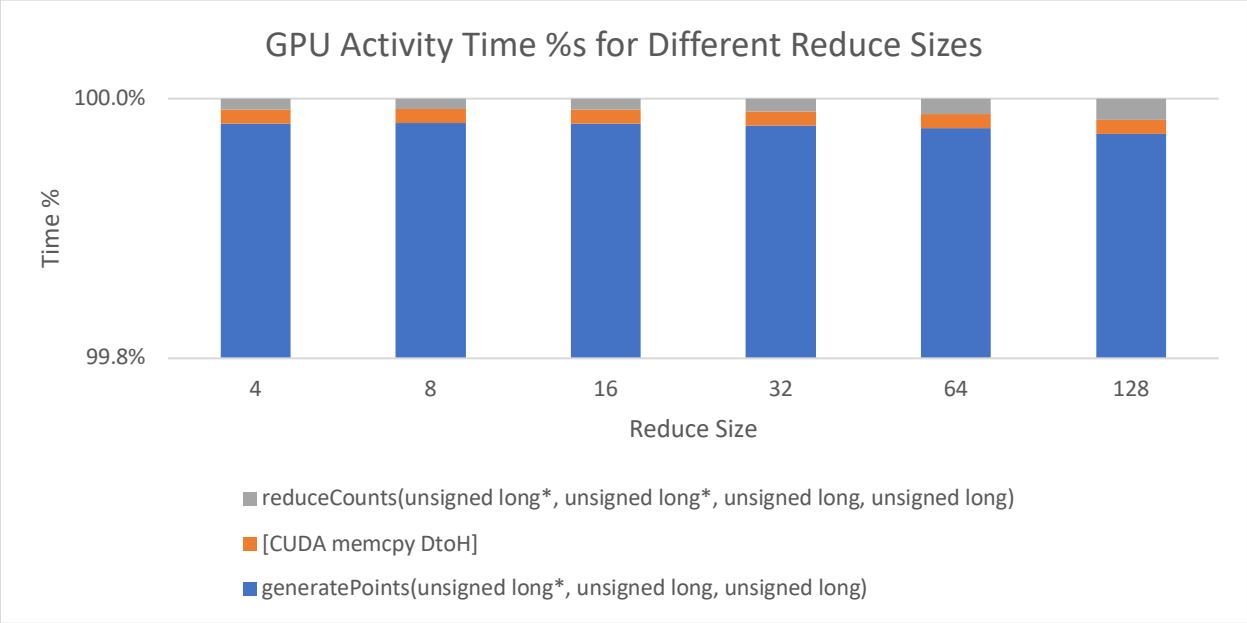


Reduce Size

Increasing the reduce size values does not really change the duration of the Monte Carlo simulation. Increasing the reduce size would mean that each thread has more data to combine, but with greater reduce size, there would be less data to copy from GPU to CPU, offsetting the increased time of data combining.



In GPU activity, increasing reduce size does seem to increase the proportion of reduceCounts function time as it has more partial sums to combine.



On the API calls side, the time percentages remain relatively constant despite changing the reduce size values.