Lab1 Report

Name: Yunzhao Liu, Github: Louis-99

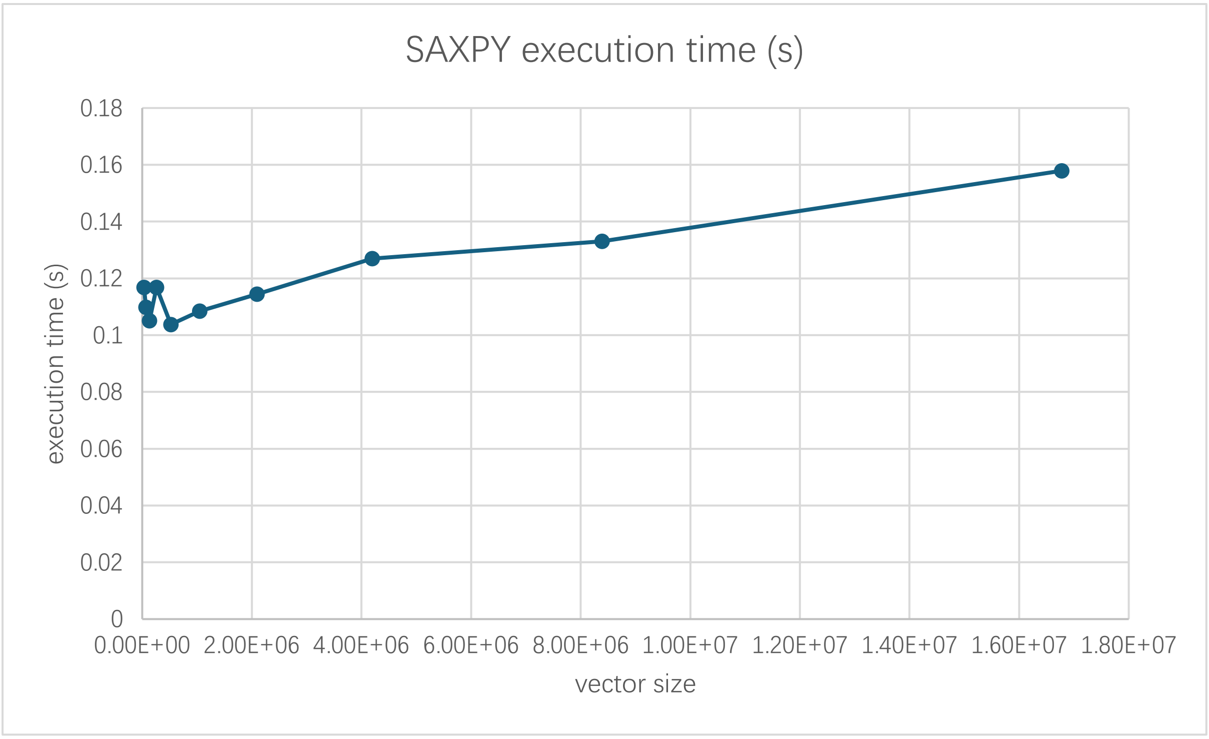
PART A: SINGLE-PRECISION A · X PLUS Y (SAXPY)

Execution Time

I set the vector size to 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, and 16777216

And measure the execution time. The relation between vector size and execution time is shown in the graph below:

X-axis is the vector size, Y-axis is the execution time



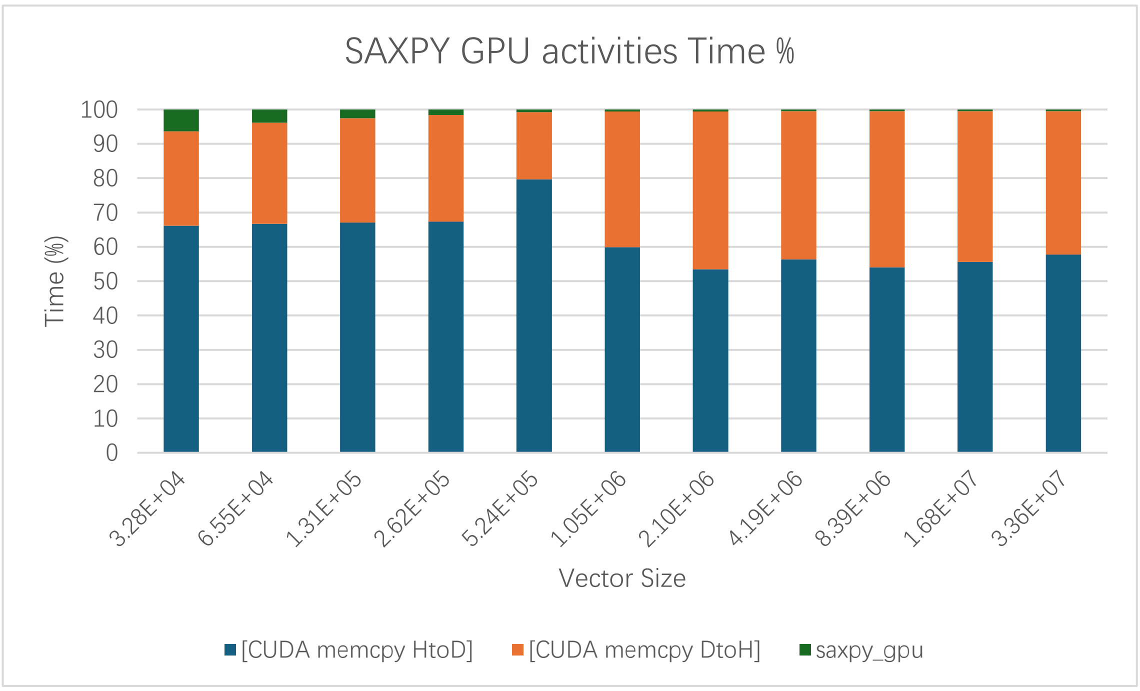
When the vector size is smaller than 2.1e6, the execution time is about the same. When the vector size is larger than 2.1e6 the execution time is a linear function of vector size.

The reason for this is that when the vector size is small enough, all threads can be parallelized and run in parallel, but when the vector size is large, there are more threads and threads block, not all threads can run in parallel because limitation of the GPU.

Breakup of Execution Time

I set the vector size to 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, and 16777216

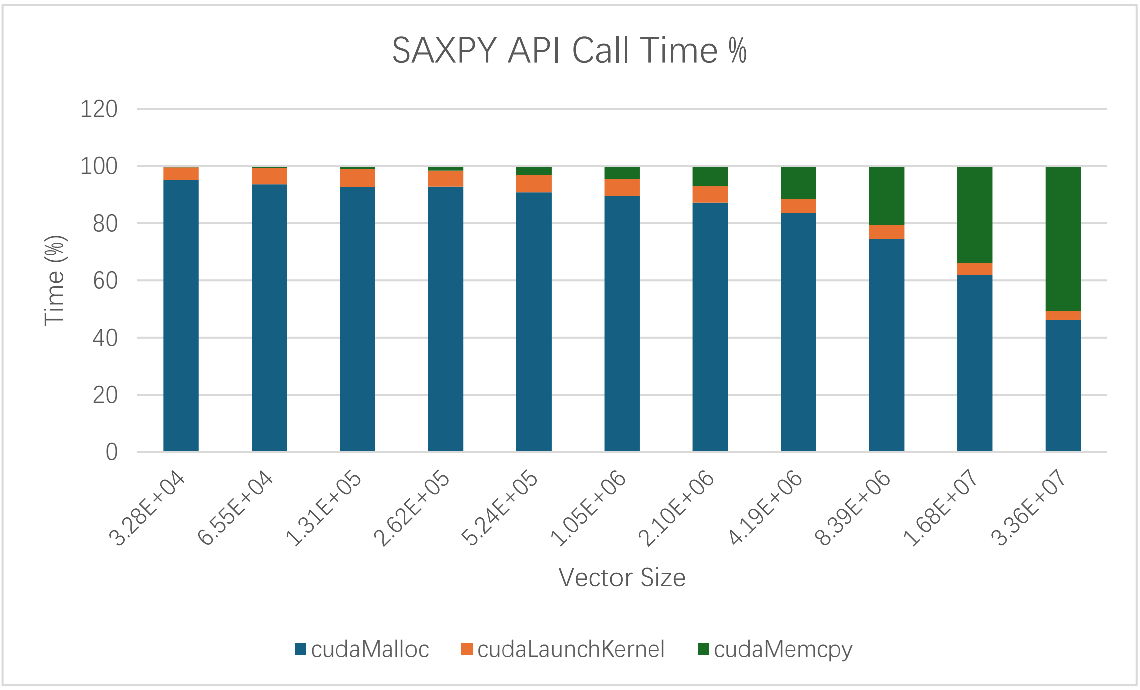
I got GPU activities time percentage breakup:



Memory copies between host and device takes up most of GPU activities time regardless of vector size. As the vector size increases, the time percentage of SAXPY kernel decreases.

I think because SAXPY is memory-bound so the memcpy is the majority in the GPU activities time.

I got API call time percentage breakup:



API call of cudaMalloc and cudaMemcpy takes up most of API Call time. As the vector size increases, the time percentage of cudaMalloc decreases and the time percentage of cudaMemcpy increases.

The reason why cudaMemcpy time percentage as the vector size increases is likely because the amount of data needs to copy increases. This API call time also includes the time to wait for the kernel to finish.

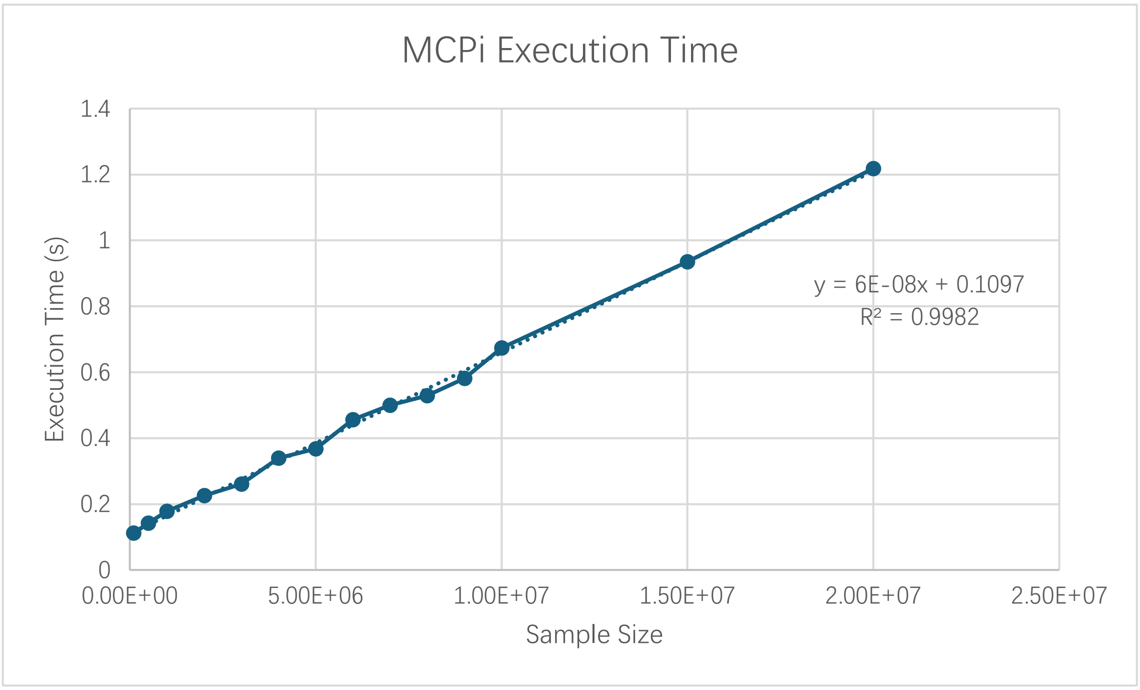
PART B: MONTE CARLO ESTIMATION OF THE VALUE OF π

Execution Time

The Impact of the sample size of each thread

I set the number of threads to sample point to 1024 and sweep the sample size:

X-axis is the sample size, Y-axis is the execution time

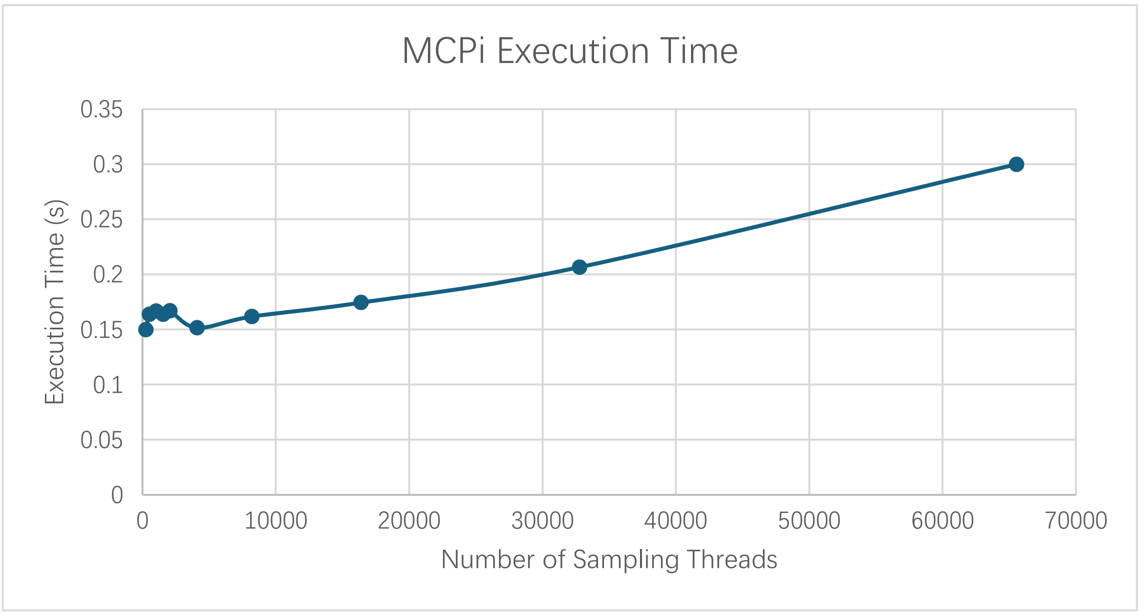


The relationship between execution time of MCPi and the sample size is a linear relationship. This is because the workload in each thread linearly increase as the sample size increases.

The Impact of the Number of Sampling Threads

I set the number of sample points per thread to 1e6 and sweep the number of sampling threads.

X-axis is the number of sampling threads, Y-axis is the execution time



When the number of sampling threads is smaller than 8000, the execution time is about the same.

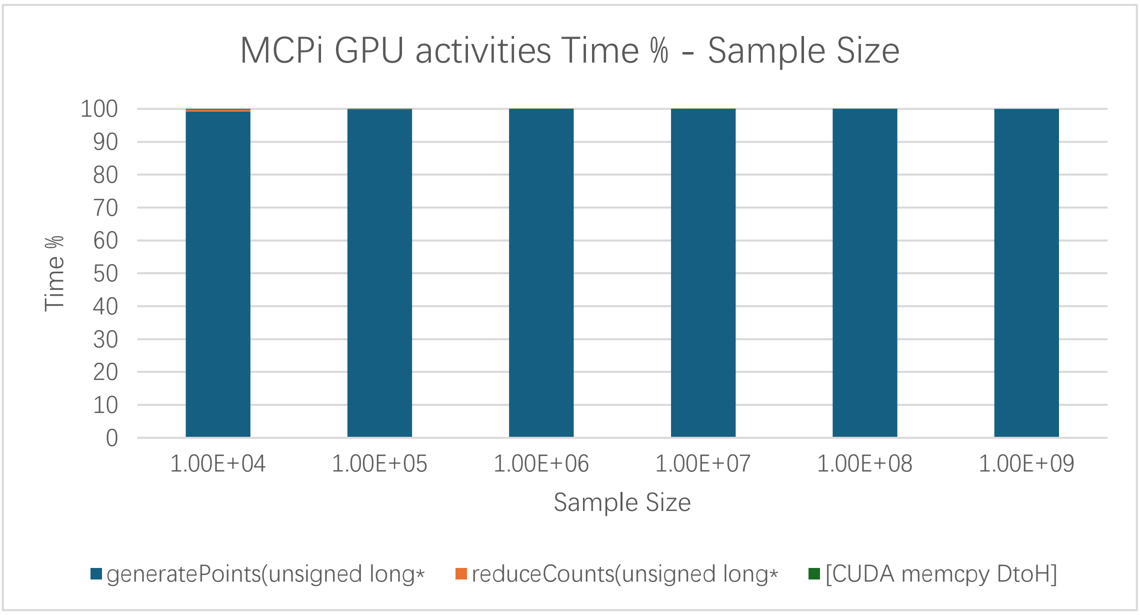
When the number of sampling threads is larger than 8000, execution increases linearly as the number of sampling threads increases.

This is because when the number of threads is large enough, GPU cannot run all threads in parallel.

Breakup of Execution Time

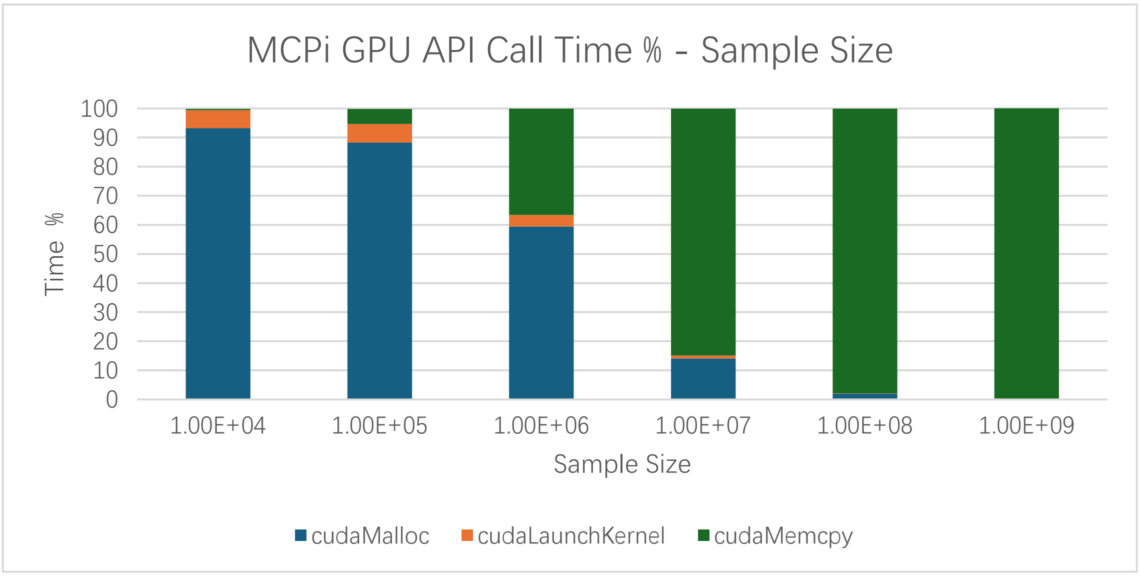
The Impact of the sample size of each thread

I set the number of threads to sample point to 1024 and sweep the sample size. I got GPU activities time percentage breakup:



More than 99% of the GPU activities time is the generate points kernel. This is because even though the number of sampling points is 10k, the generating points kernel is significantly more computation intensive than other operations on GPU.

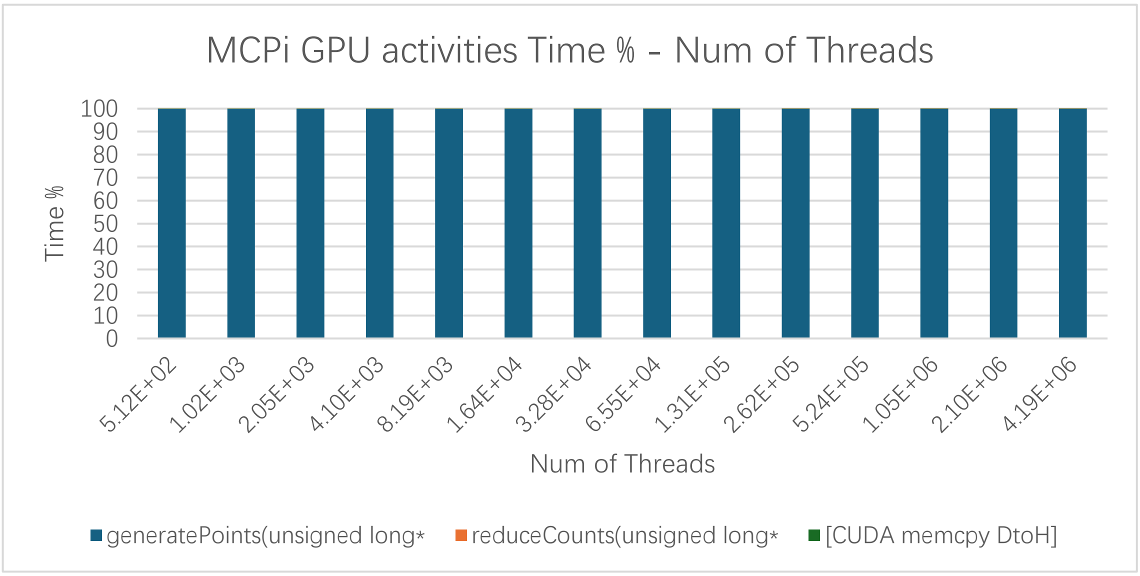
I got API Call time percentage breakup:



When sample size increases, the time percentage of cudaMalloc decreases and the time percentage of cudaMemcpy increases. The reason of the increase of cudaMemcpy time percentage is that the cudaMemcpy need to wait the CUDA kernel finished to begin to copy data and the kernel execution time increases as the sample size increases.

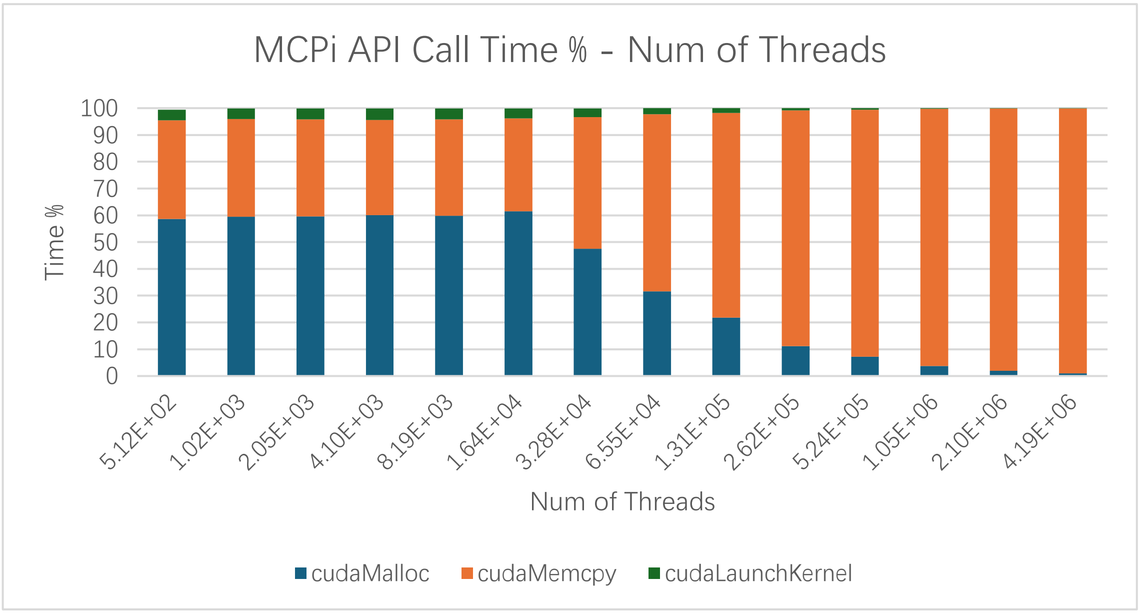
The Impact of the Number of Sampling Threads

I set the number of sample points per thread to 1e6 and sweep the number of sampling threads. I got GPU activities time percentage breakup:



More than 99% of the GPU activities time is the points generating kernel. The points generating kernels are always the kernel with the most intensive workload.

I got API call activities time percentage breakup:



The API Call time percentage of cudaMemcpy increases when the number of threads is larger than 16k. This is because the run time of generatePoints increases as the number of threads increases when the number of threads is above 16k and cudaMemcpy API Call waits for the kernel to finish before copying the data.