ECE 695 CUDA Programming Part 1 Report

Name: Wen-Bo Hung, SID: 0033893868, GitHub: PeterHung26

Size of thread block

I found that each SM can have 2048 threads, with a maximum of 16 active block per thread. Therefore, I configured each thread block to contain 128 threads, ensuring 100% occupancy of SM.

SAXPY Implementation

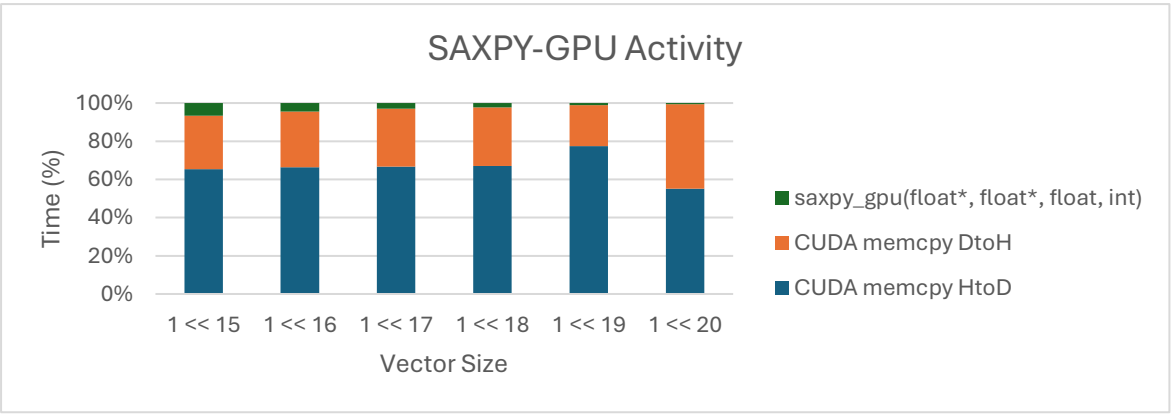
1. GPU Activity:

a. CUDA memcpy HtoD: I first generated random arrays for X and Y on the CPU and then transferred them to the GPU. This involved two memory copies from the host to the device, one for each array.

b. CUDA memcpy DtoH: After completing the SAXPY calculation, I transferred the results back to the CPU's memory and verified them on the CPU side.

c. saxpy_gpu(float*, float*, float, int): This is the kernel implementing the SAXPY operation for each thread, with each thread responsible for calculating one element of the X and Y arrays.

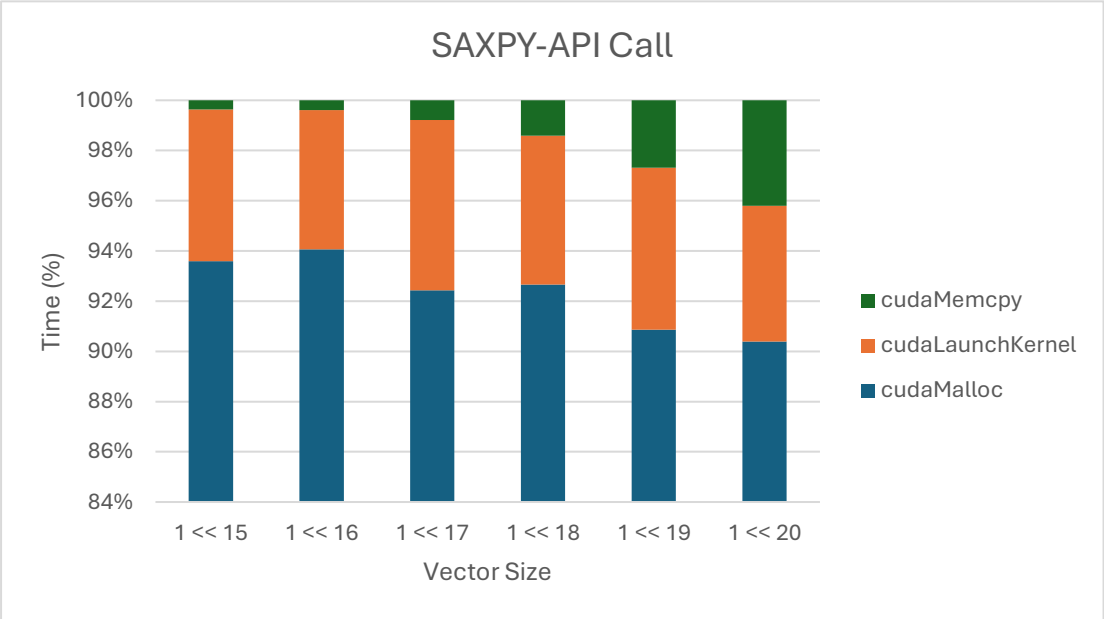
From the stacked bar chart below, we can observe that as the vector size increases, the memory copy time becomes more prominent, while the SAXPY kernel occupies a smaller portion of the GPU execution time. This indicates that the SAXPY implementation is memory-bound, with memory bandwidth serving as the bottleneck.



2. API Call:

Three API calls accounted for the majority of the execution time: cudaMemcpy, cudaLaunchKernel, and cudaMalloc.

From the bar chart below, we observe that cudaMalloc accounted for over 90 percent of the execution time, while cudaLaunchKernel took up around 5 percent. As the vector size increases, the percentage of time spent on cudaMemcpy also grows. The reason why cudaLaunchKernel takes longer than cudaMemcpy could be due to the overhead associated with kernel launch, such as scheduling delays.



Monte Carlo estimation of the value of π

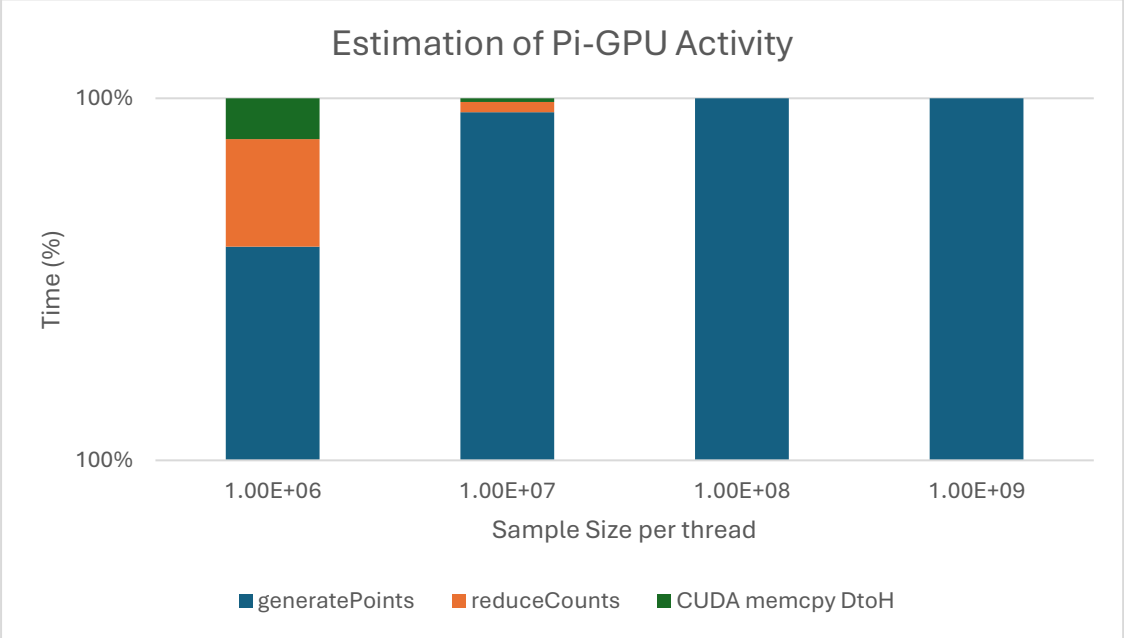
I modified three parameters—sample size, number of threads, and reduction size—and analyzed how each parameter affects the execution time.

A. Sample Size

The reduce size is 32 and generate block is 1024 in this section.

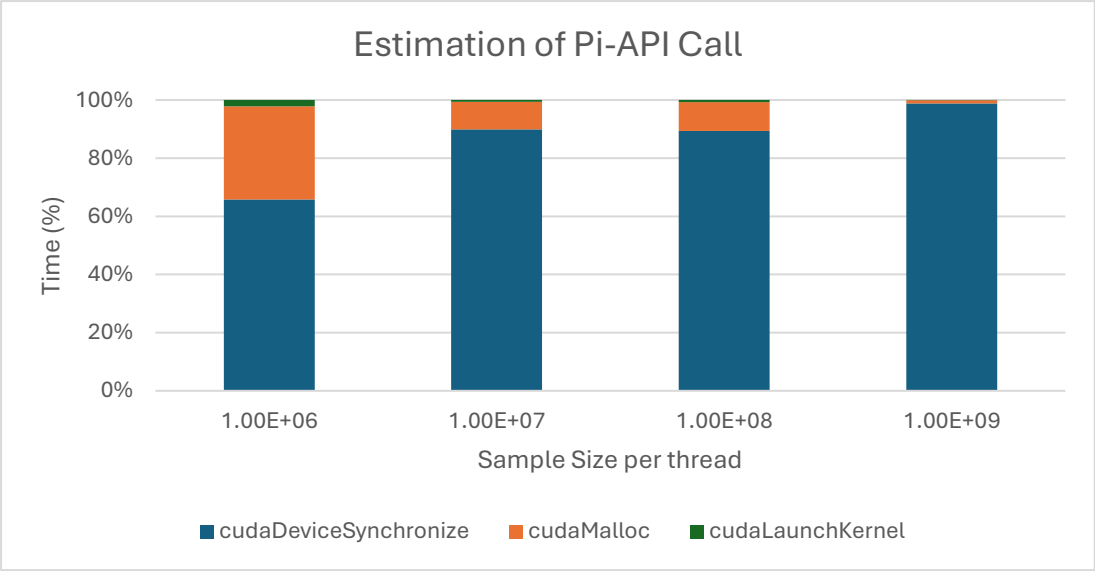
1. GPU Activity:

As shown in the chart below, the **generatePoints** kernel accounts for the majority of the execution time. According to the profiler data, the times for **reduceCount** and **memcpy** remain relatively constant as the sample size increases. However, the execution time of **generatePoints** increases linearly with the sample size.



2. API call:

CudaDeviceSynchronize had the same execution time as the **generatePoints** kernel because it was called immediately after **generatePoints**. Since kernel launches are asynchronous, control returns to the host right after invoking **generatePoints**. **CudaDeviceSynchronize** blocks execution until the kernel completes. As the sample size per thread increases, the relative overhead of **cudaMalloc** and **cudaLaunchKernel** becomes less significant.

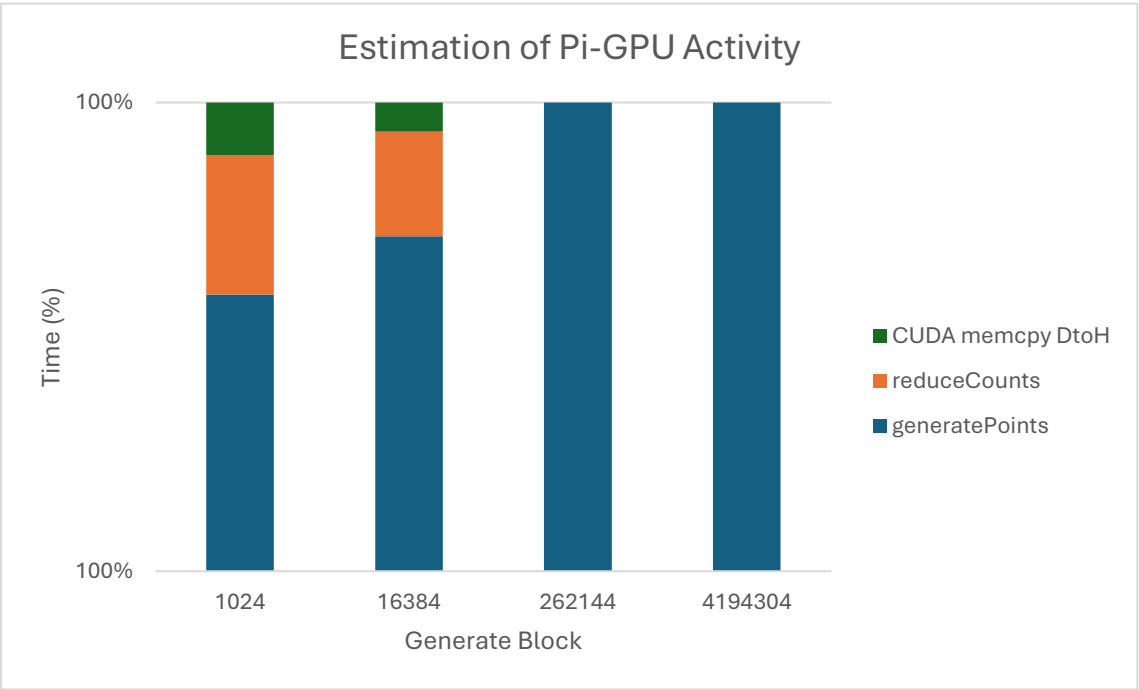


B. Generate Blocks (number of thread):

Sample size is 1e6 and reduce size is 32 in this section.

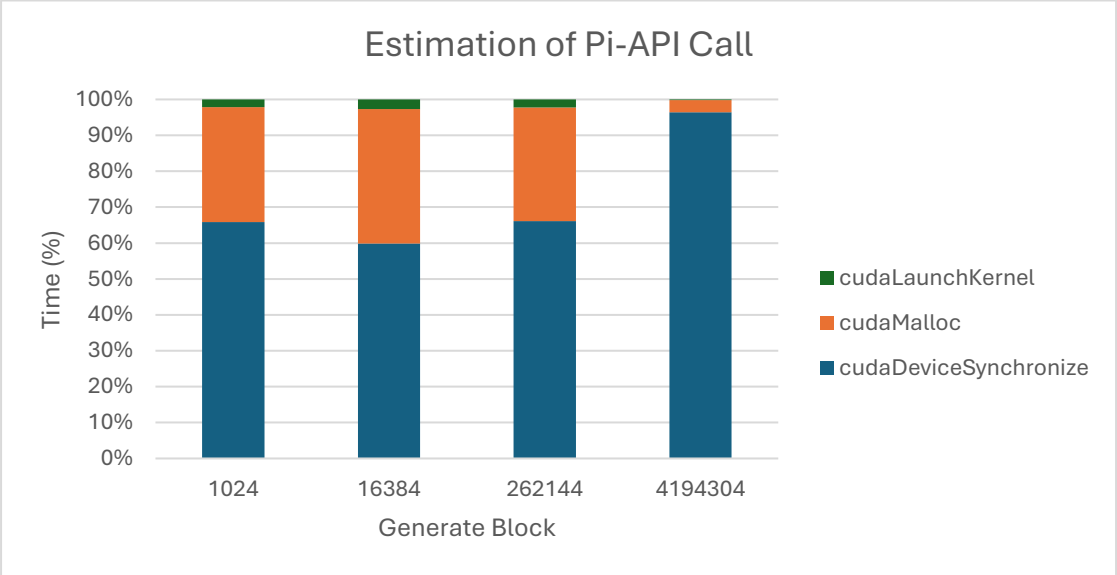
1. GPU Activity:

From the chart below, we can see that **generatePoints** accounts for the majority of the execution time, and its impact becomes even more significant as the number of generated blocks increases. Although the execution time of **reduceCounts** and **cudaMemcpy** also increases with the number of generated blocks, their contribution becomes much less significant as the number of threads grows.



2. API Call:

cudaDeviceSynchronize had almost the same execution time as **generatePoints** for the same reason mentioned in the API call analysis for sample size. Since **generatePoints** dominates the GPU execution time, **cudaDeviceSynchronize** also accounts for the largest portion of the API call execution time.

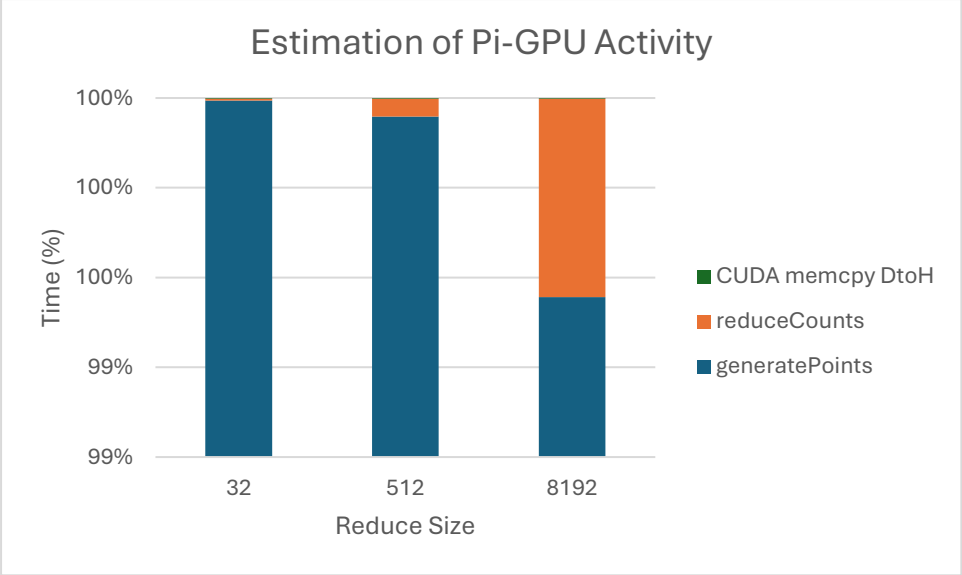


C. Reduce Size:

Generate block is 16384 and sample size is 1e6 in this section.

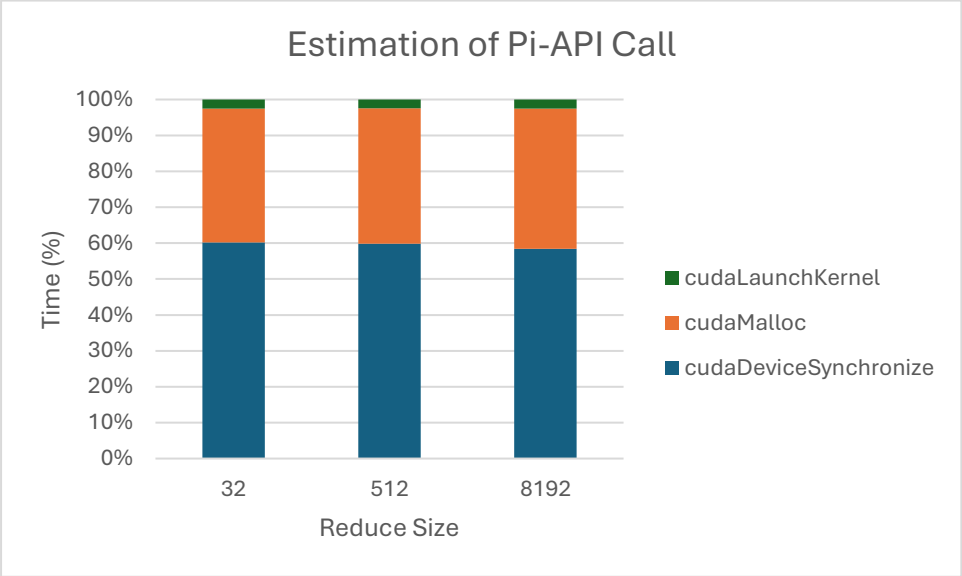
1. GPU Activity:

From the bar chart below, we can see that execution time increases as the reduce size increases. However, compared to the execution time of **generatePoints**, this increase is negligible. Therefore, the impact of reduce size on overall execution time is minimal.



2. API Call:

Reduce Size has little effect on the execution time of API call.



Summary:

As shown in the graphs above, the Monte Carlo estimation of π is a compute-bound application. The sample size and the number of generate blocks significantly impact the execution time of the **generatePoints** kernel, whereas the reduce size has minimal effect on the overall GPU execution time.