ECE 60827 PA1

Te-Yu Hsin (GitHub id: EricHsin)
thsin@purdue.edu

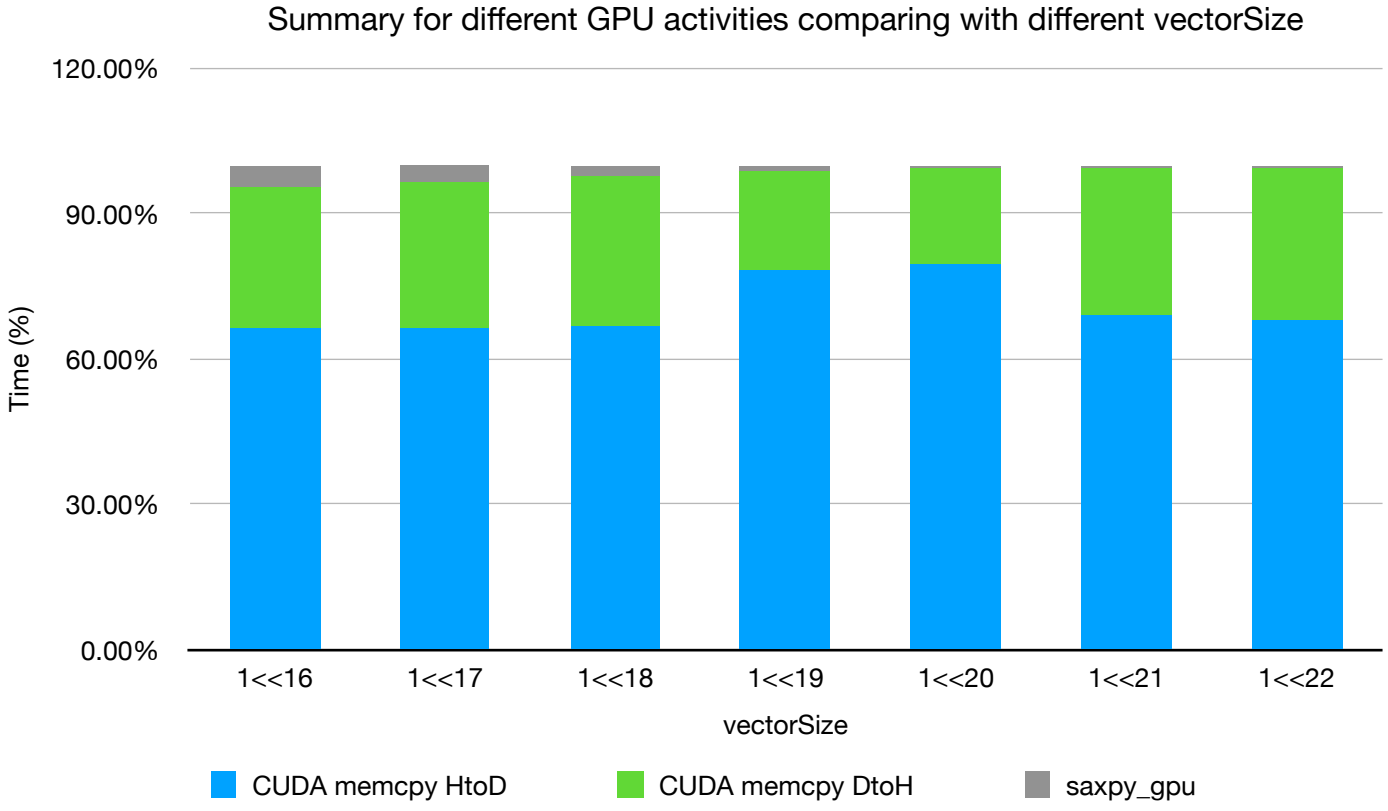
PART A: Single-precision A · X Plus Y (SAXPY)

Experiment 1. Comparing different vectorSize

GPU activities:

1. **CUDA memcpy HtoD:**
   - Around **66%** for 1<<16, 1<<17, 1<<18
   - Jumps to around **80%** for 1<<19 and 1<<20
   - Drops back to around **70%** for 1<<21 and 1<<22
2. **CUDA memcpy DtoH:**
   - Around **30%** for 1<<16, 1<<17, 1<<18
   - Drops to around **20%** for 1<<19 and 1<<20
   - Increases again to around **30%** for 1<<21 and 1<<22
3. **saxpy_gpu kernel:**
   - Decreases steadily from around **4%** at 1<<16 to around **0.64%** at 1<<22

The data shows that for vector sizes in the 1<<19 to 1<<20 range, the host-to-device memory transfer becomes significantly less efficient, maybe due to alignment, page, or buffer boundary issues, while the thread kernels remain very efficient. For even larger sizes, the HtoD efficiency recovers somewhat, possibly because the allocation aligns more



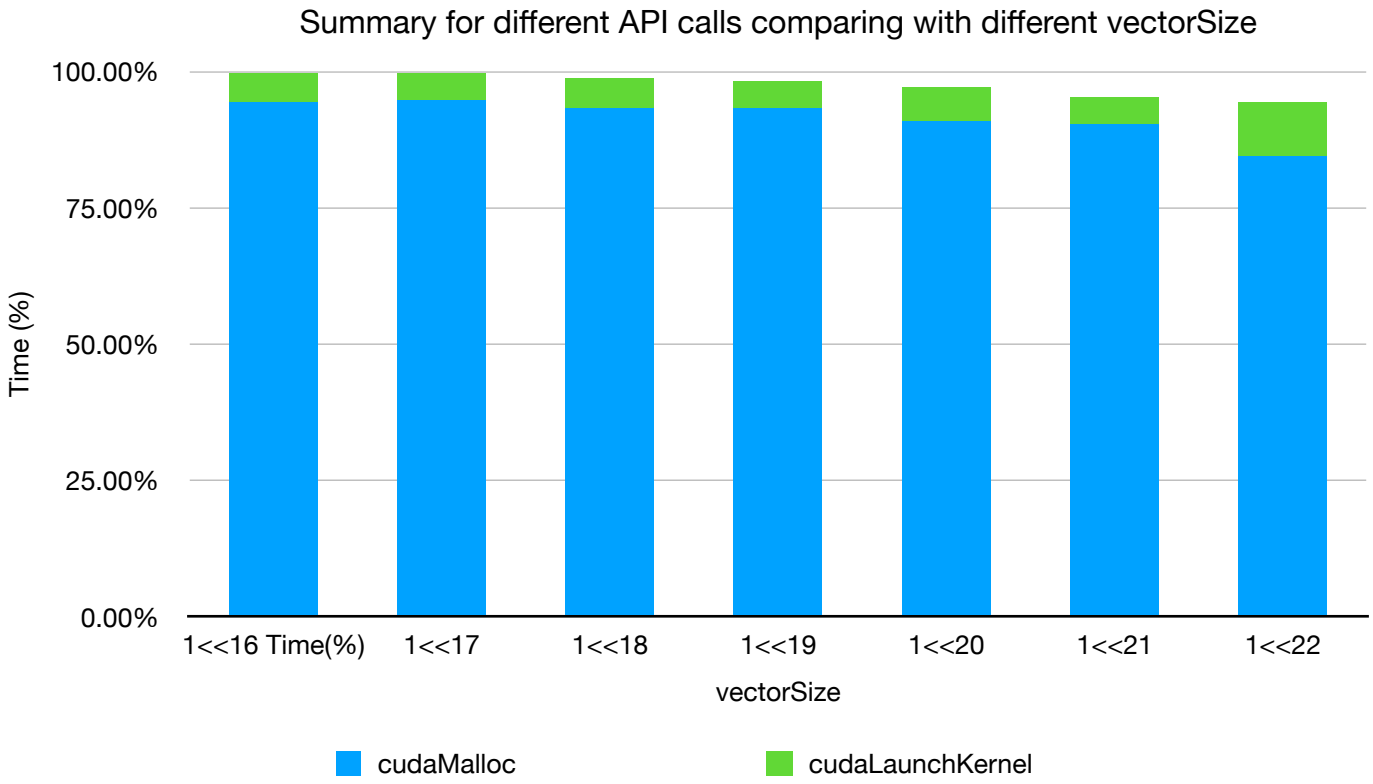
favorably with the GPU's internal architecture.

API calls:

cudaMalloc decreases steadily from around **95%** at 1<<16 to around **84%** at 1<<22.

cudaLuanchKernel is relatively low (around **5%**) for most sizes. However, at the largest vector size (1<<22), the percentage spikes to **9.35%.**

For most tested vector sizes, **cudaMalloc** is the dominant API call, indicating that memory allocation is one of the major performance bottlenecks in GPU computing. In addition, as the vector size increases, the relative cost of memory allocation decreases slightly, while kernel launch overhead can increase—especially at the highest tested sizes, where kernel launch overhead jumps to about **9.35%.**



PART B: Monte Carlo estimation of the value of π

Experiment 1. Comparing different SampleSize

GPU activities:

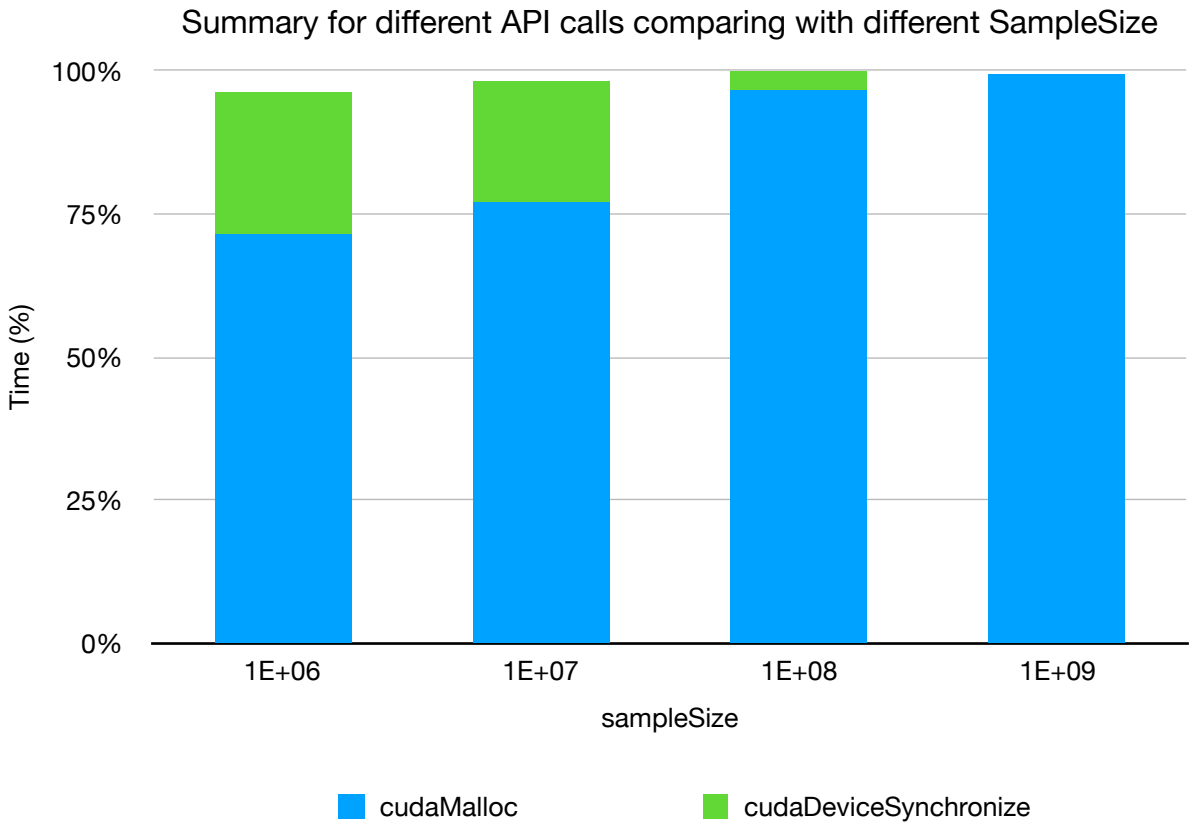
GeneratePoints function dominates execution time (around 99%), even if I keep changing the number of SampleSize, this indicates that the majority of execution time is spent inside the generatePoints kernel, suggesting it is the primary bottleneck.

ReduceCounts and CUDA memcpy DtoH contribute negligible time (0.01%), indicating that reduceCounts is not a significant time-consuming operation. Lastly, very little time is spent on data transfers back to the host, meaning most of the computation remains within the GPU.

API calls:

cudaMalloc time increases dramatically with sample size, since each thread is going to malloc more space due to the increase of sample size, this suggests that memory allocation will become an increasing bottleneck as the sample size grows.

cudaDeviceSynchronize time decreases as sample size grows, this indicates that although for small sample sizes, synchronization overhead is more noticeable, as the workload increases, the relative impact of **cudaDeviceSynchronize** becomes negligible.



Experiment 2. Comparing different number of GeneratingThreads

This experiment explores the impact of **GenerateThreads** ranging from 1024 to 8192, with a fixed **ReduceSize = 32** and provides percentages of execution time spent on **GPU activities** and **API calls**.

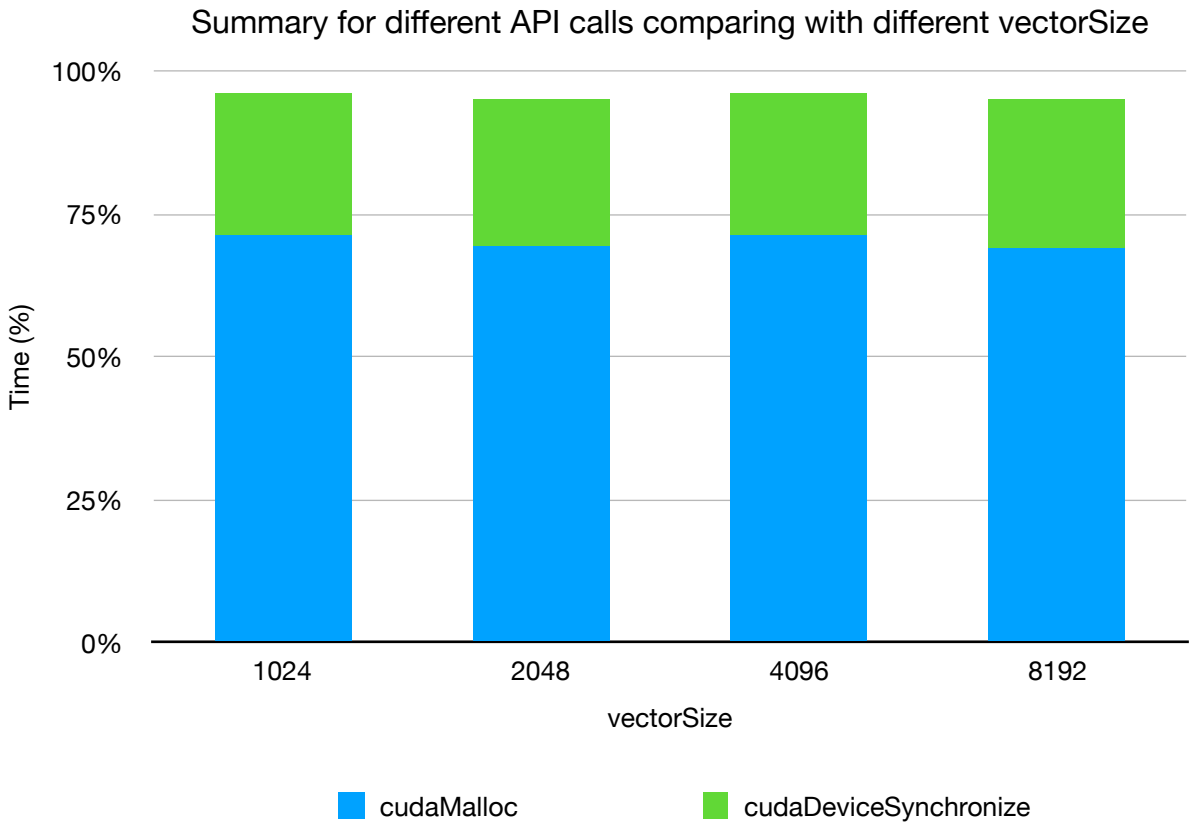
GPU Activities:

generatePoints still dominates execution time (99.98%) across all thread counts. This indicates that the majority of execution time is spent inside the **generatePoints** kernel. The number of threads does not impact the proportion of time spent on this kernel, implying that its workload is evenly distributed across different thread configurations. While **reduceCounts** and **CUDA memcpy DtoH** remain negligible (~0.01%).

API calls

cudaMalloc consistently takes around 70% of execution time, regardless of whether GenerateThreads = 1024 or GenerateThreads = 8192.

cudaDeviceSynchronize percentage fluctuates around 25%. This indicates that kernel synchronization time remains relatively stable. The slight variations may be due to thread scheduling inefficiencies or the way GPU execution units handle different workloads.

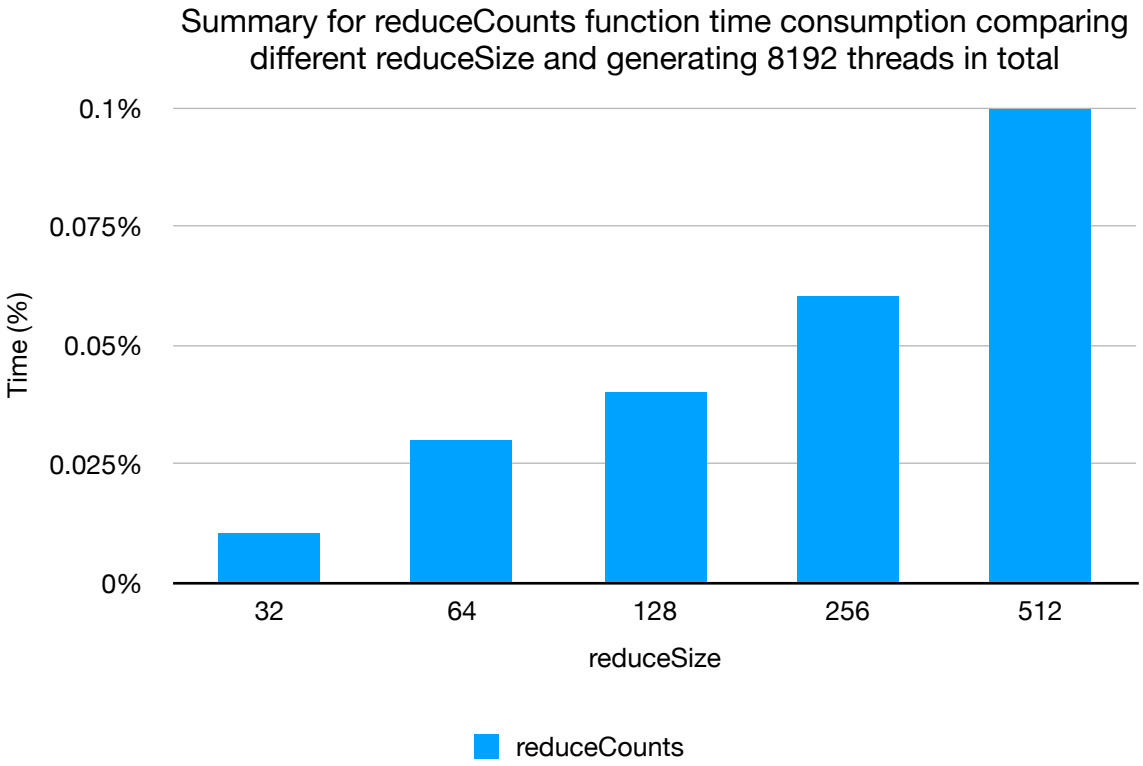


Experiment 3. Comparing different number of reduceSize

This experiment explores the impact of **ReduceSize** ranging from 32 to 512, while keeping **GenerateThreads** = 8192 and provides percentages of execution time spent on **GPU activities** and **API calls**.

GPU Activities:

Same as previous experiments, **generatePoints** consistently dominates execution time (99.9%). However, **reduceCounts** execution time increases as **ReduceSize** increases. This makes sense since a larger reduce size means that more data is being processed in this step and more threads are working on summing up results, but even at the highest **ReduceSize**, the impact of **reduceCounts** remains negligible.



API calls

For API calls, both **cudaMalloc** and **cudaDeviceSynchronize** time remains stable, **cudaMalloc** is high (~70%) but fluctuates slightly and **cudaDeviceSynchronize** is around 25%.