Manoj Vishwanathan
ece60827-Lab1

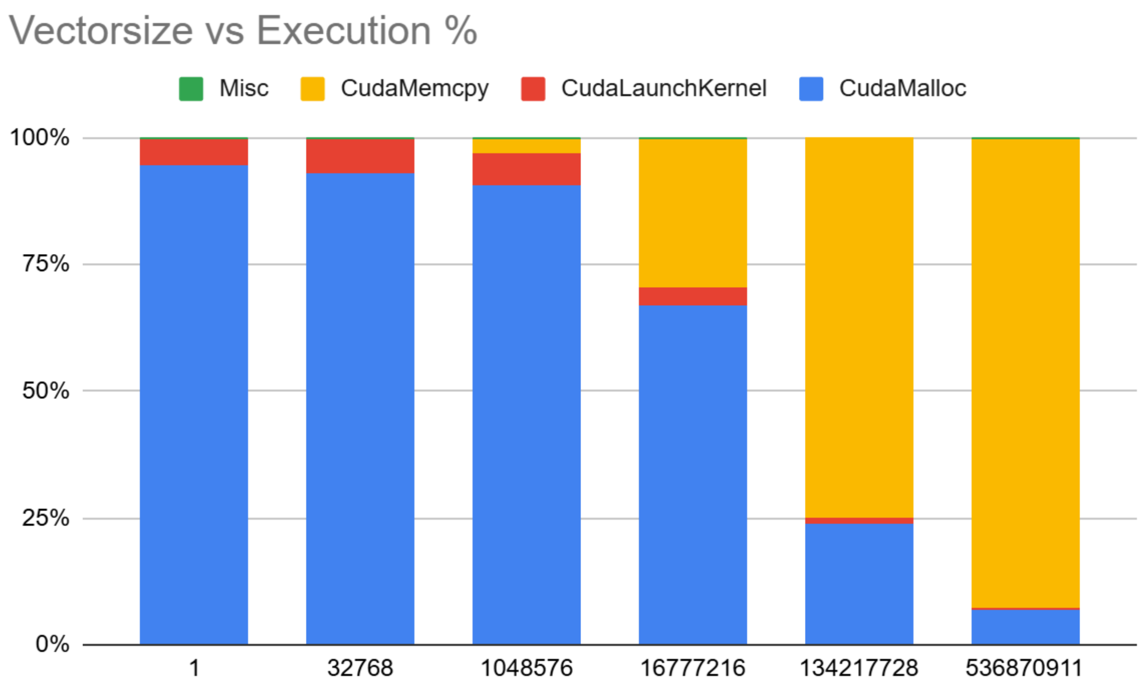
Platform information

NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB] (rev a1)
Maximum number of threads per block: 1024
Maximum number of threads per multiprocessor: 2048
Number of streaming multiprocessors: 80
HBM: 32768MiB
Memory Interface : 4096 bits
L2 Cache: 6144 KB
Register File per SM: 20490 KB

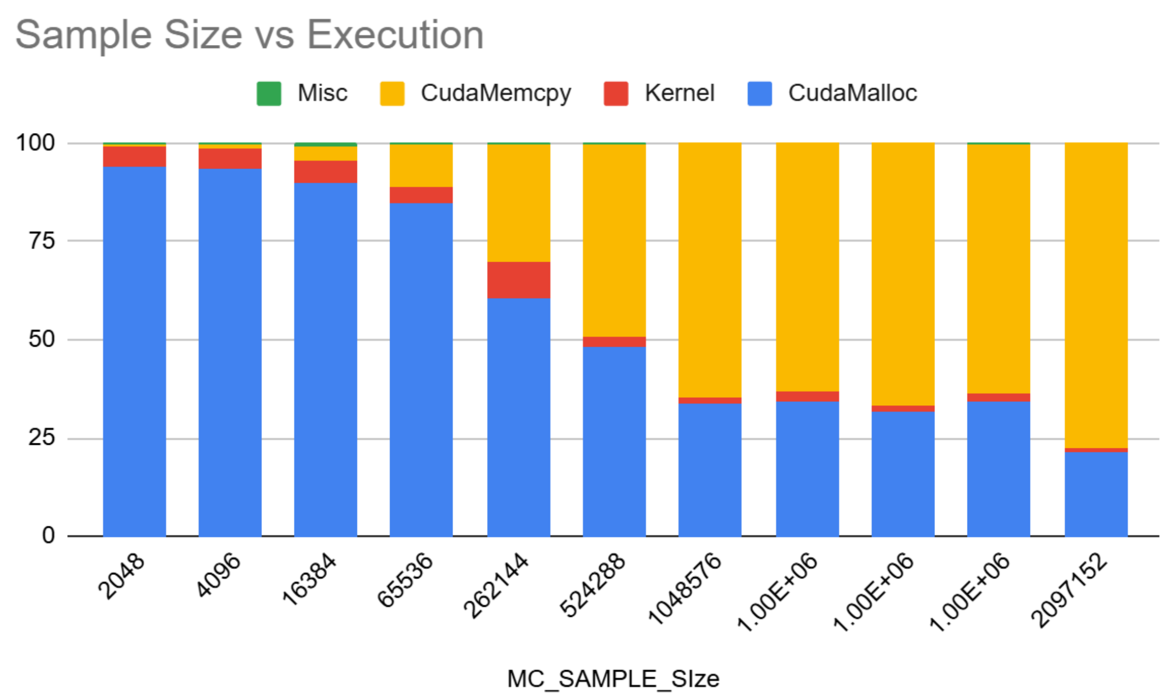
Part A:

The input in Part A is the Vector Size, ranging from 1 to approximately 500 million vectors. The primary observation is that an increase in vector size results in a surge in CudaMemcpy duration. As the vector size increases, the amount of data that needs to be copied by memcpy also increases. This increase in data volume directly translates to additional time spent on the memory copy operation. But because the kernel computation does not increase linearly with input, the memcpy becomes the dominant portion of the bar graph.

Bar Graph for Vector Size vs GPU API-Call



Part B: Monte Carlo simulation for Pi estimation
CudaMalloc is the dominant operation for small sample sizes, but memory copy increases significantly as sample size grows. While CudaMalloc remains substantial, kernel operations don't scale as much, so memcpy becomes the largest component.



Conclusion:

In summary, our findings demonstrate the impact of input size on the performance of CUDA GPU API calls. In Part A, it is observed a clear correlation between vector size and CudaMemcpy duration, with larger vector sizes leading to increased memory copy times. This behavior can be attributed to the straightforward relationship between data volume and memory copy duration. In Part B, CudaMalloc dominated for small sample sizes, but as sample size grew, memory copy emerged as the primary performance bottleneck. These results highlight the importance of considering input size when optimizing GPU code, as it can significantly influence the performance of CUDA API calls.