ECE 60827
Riya Gautam (gautam28@purdue.edu)
Programming Assignment 1

a) **SaxPy using GPU**

Descriptions:

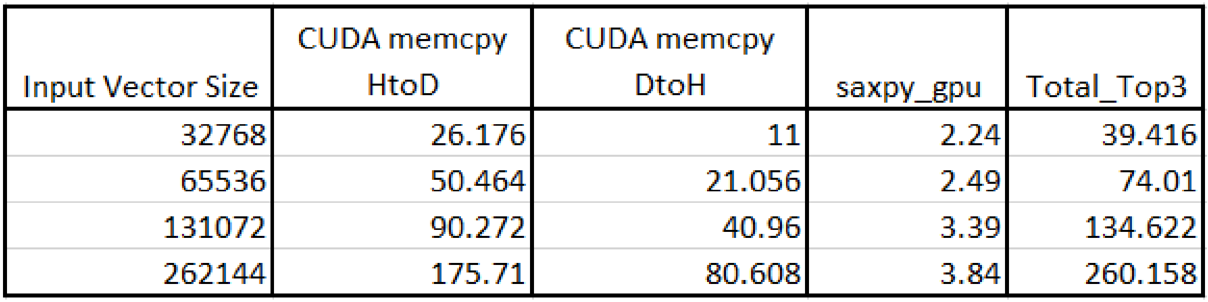
1)saxpy_gpu kernel:

Takes x, y, scale and size as arguments and performs the SAXPY operation (y[i] = scale*x[i] + y[i]. Here, each thread will process one element of the vectors based on its unique global index.

2)runGpuSaxpy

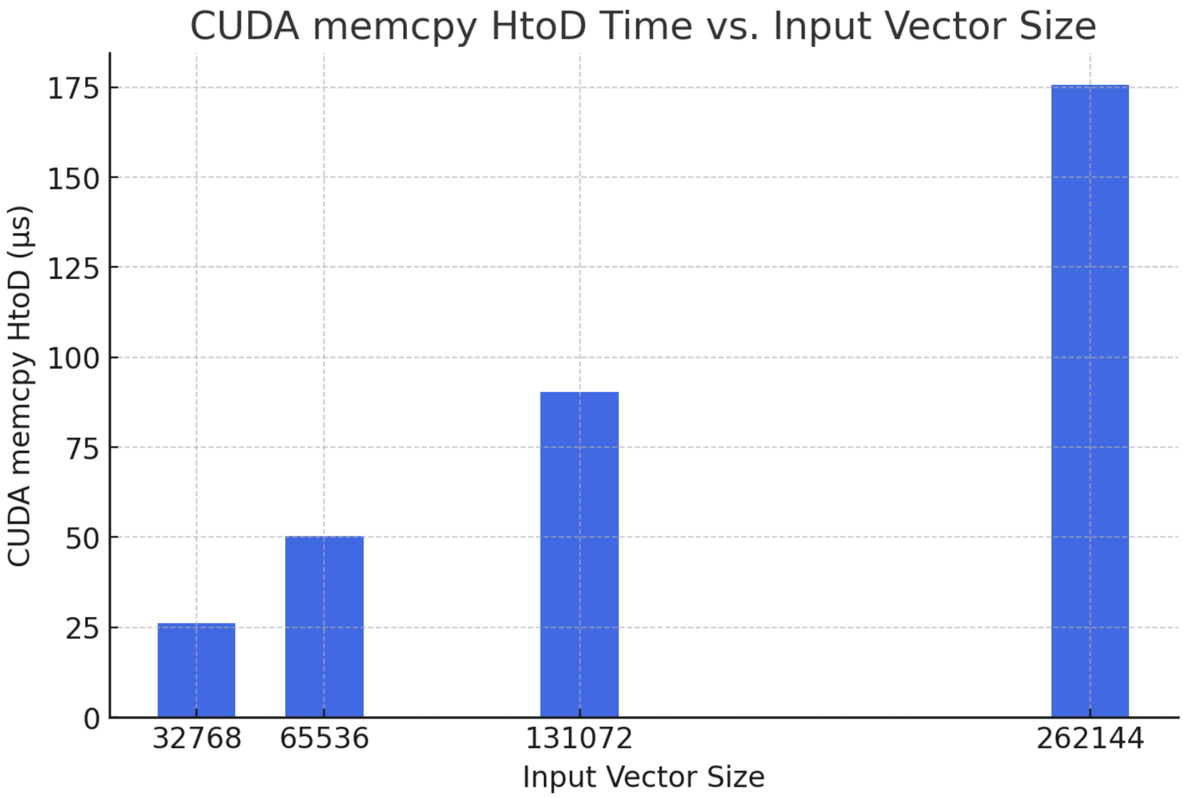
Takes vectorSize as argument, allocates dynamic memory (xHost, yHost, yresult) on host using new operator and initializes xHost, yHost with random data. cudaMalloc is then used to allocate memory on the GPU for xDevice and yDevice. cudaMemcpy is used to copy data from host to device for computation. saxpy_gpy kernel is called to perform operations. cudaMemCpy is used to copy result from GPU to CPU. Memory is freed from both CPU and GPU to avoid memory leak.

Table 1 is depicting how execution time of top 3 most time-consuming function varies with respect to variation in Vector size.



Input Vector Size	CUDA memcpy HtoD	CUDA memcpy DtoH	saxpy_gpu	Total_Top3
32768	26.176	11	2.24	39.416
65536	50.464	21.056	2.49	74.01
131072	90.272	40.96	3.39	134.622
262144	175.71	80.608	3.84	260.158

Table 1: *Variation(us) of top 3 time consuming functions w.r.t variation in Input Vector Size*



It can be observed that time consumed by the top 3 most time-consuming functions is proportional to the input vector size.

Estimating Pi using GPU

Descriptions:

1)generatePoints:

Kernel that generates random points in the unit square using curand_uniform(). Counts how many points fall inside the unit circle and stores the counts in pSums for each thread.

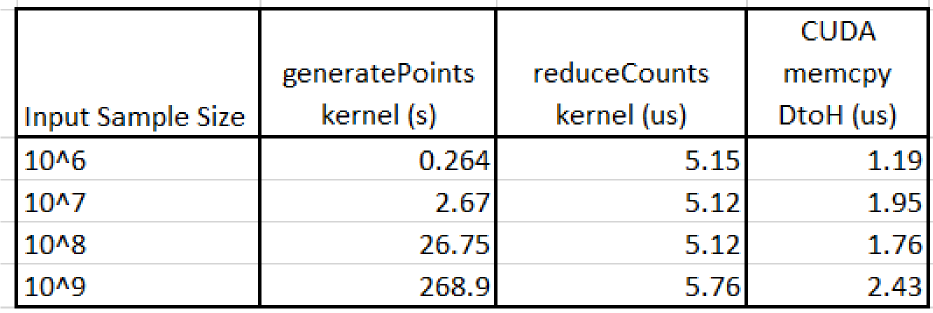
2)reduceCounts:

Kernel that reduces pSums by summing up the arry in totals, aggregating counts of points in the unit circle from multiple threads.

4)estimatePi:

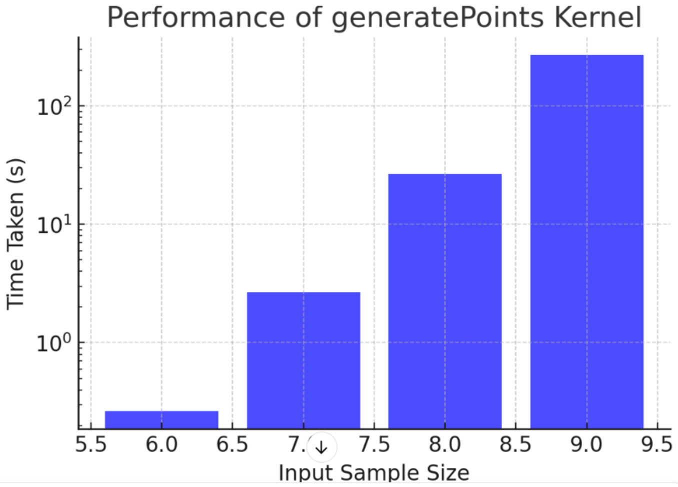
Allocates memory on GPU, launches generatePoints and reduceCounts kernels, copies the reduced results back to the host, computes and approximate value of pi using Monte Carlo integration and returns it.

Table 2 is depicting how execution time of top 3 most time-consuming function varies with respect to variation in Input Sample size (MC_SAMPLE_SIZE from lab1.cuh).

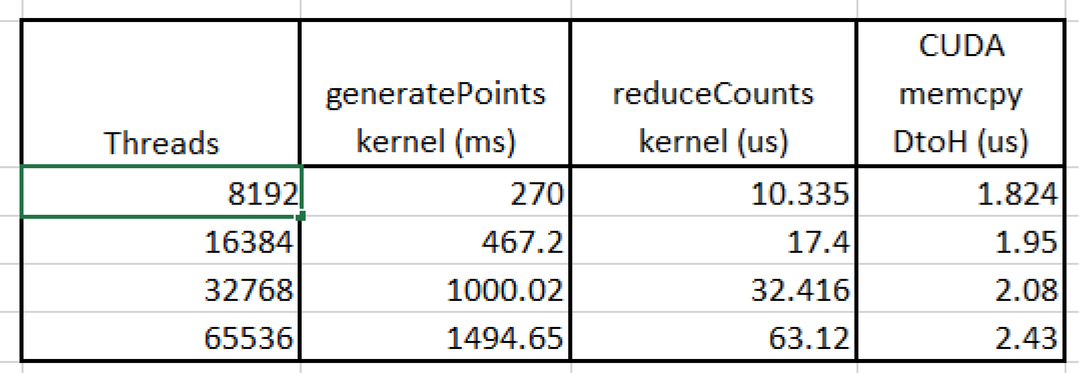


Input Sample Size	generatePoints kernel (s)	reduceCounts kernel (us)	CUDA memcpy DtoH (us)
10^6	0.264	5.15	1.19
10^7	2.67	5.12	1.95
10^8	26.75	5.12	1.76
10^9	268.9	5.76	2.43

***Table 2**: Variation of top 3 most time-consuming functions w.r.t Sample Size*

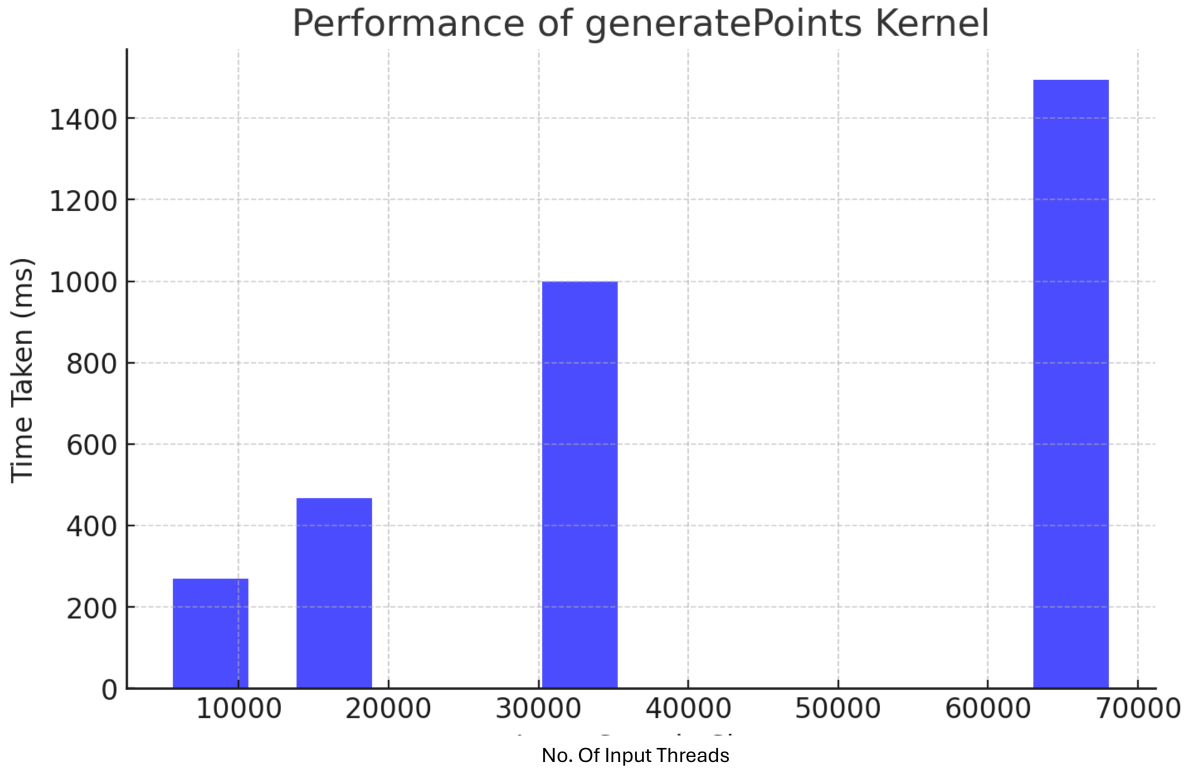


It can be observed that time consumed by the generatePoints kernel is proportional to the input sample size.



Threads	generatePoints kernel (ms)	reduceCounts kernel (us)	CUDA memcpy DtoH (us)
8192	270	10.335	1.824
16384	467.2	17.4	1.95
32768	1000.02	32.416	2.08
65536	1494.65	63.12	2.43

***Table 3**: Variation of top 3 most time-consuming functions w.r.t Number of Threads*



It can be observed that time taken by top 3 time-conuming functions is proportional to the number of input threads.