ECE 60827 Programming Assignment 1

Name : Kaarthikeya Karanam

Github id: kkaranam14

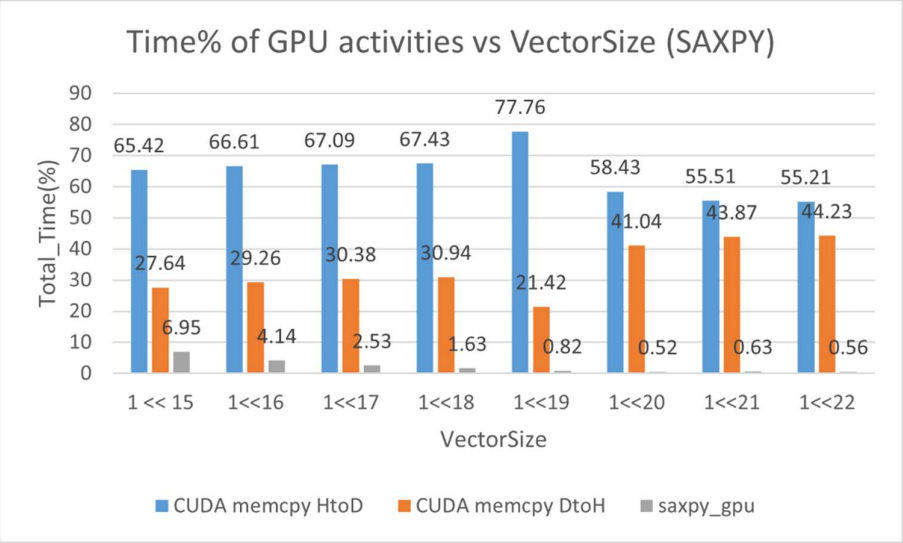
Part – A (SAXPY)

In the part A, after executing the SAXPY code on the GPU, we will be comparing it's Total_time(%) of different GPU activities like cudamemcpy Host To Device , cudamemcpy Device To Host and saxpy_gpu and different API activities like cudaMalloc and cudaLaunchKernal by varying the Vector size.

Key terms:

1) CUDA memcpy HtoD = cudamemcpy Host to Device
2) CUDA memcpy DtoH = cudamemcpy Device to Host

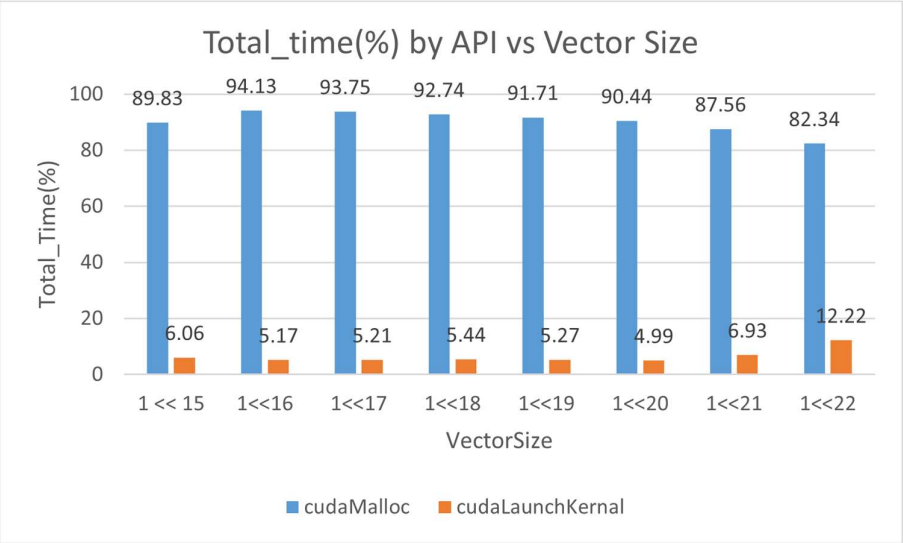
GPU Activities:



Observation:

From the above graph, it is evident that as the VectorSize goes from 1<<15 to 1<<19, the cudamemcpy Host to Device will be consuming a major potion of time and it increases with size (maximum at 1<<19) and cudamemcpy Device to Host is the 2nd largest in Total_Time (%) and it also increases with VectorSize (for cudamemcpy Device to Host upto 1<<18). But from 1<<20 to 1<<22, it is evident that the (%) Total_Time stacks of cudamemcpy Host to Device decreases (although it is the largest (Total_Time%)) and cudamemcpy Device to Host increases in Total_Time(%). When it comes to saxpy_gpu it's Total_Time(%) decreases with increases in vectorsize although it has a slight fluctuation in between 1<<20 and 1<<22.

API Activities:



Observation:

Now, this graph describes about the Total_time(%) taken by API activities with increase in Vector Size. From the graph, it is evident that cudaMalloc is the largest (%) and the next is cudaLaunchKernal. Although cudaMalloc is the largest Total_time(%), it gets fluctuated in the mean time and has the highest at 1<<16 i.e. 94.13% and similarly for the cudaLaunchKernal which gets it's highest value at 1<<22.

Part -2 Using Monte Carlo to get the value of Pi:

Here, too after executing the code to achieve the value of Pi, we will be comparing it's Total_time(%) of different GPU activities like cudamemcpy Host To Device , cudamemcpy Device To Host and saxpy_gpu and different API activities like cudaMalloc and cudaLaunchKernal by varying the samplesize, GenerateThreads and reduceSize.

Key Terms:

1) samplesize = sampleSize

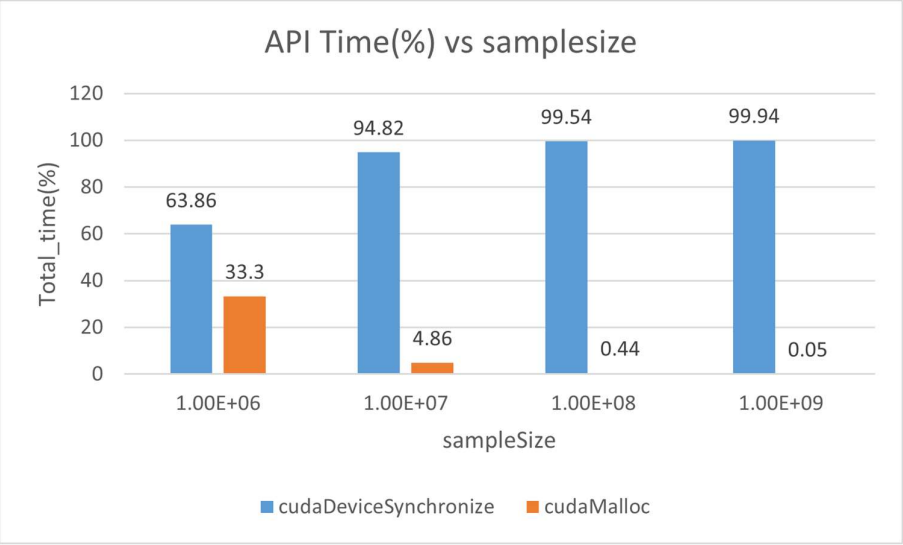
i) By Varying sampleSize:

Here the performance is compared with the varying sampleSize by keeping the values of reduceSize and Generate Thread as constant.

GPU Activities:

When it comes to GPU activities like generatePoints, reduceCounts and [CUDA memcpy DtoH], the Total_time(%) of reduceCounts and [CUDA memcpy DtoH] are negligible when compared with the Total_time (%) of generatePoints. Like the generatePoints is in the milli seconds to seconds as the sampleSize increases and the rest two activities are in the microseconds. Majority or equal to 100% will be of generatePoints.

API Activities:



Observation:

Although there are many API activities like cudaDeviceSynchronize, cudaMalloc, cudaFree, cudaLaunchKernal etc, cudaDeviceSynchronize and cudaMalloc are the most prominent ones, as their Total_time(%) is nearly approximately above 97%. From the above graph it is evident that as the sampleSize increases the Total_time(%) of cudaDeviceSynchronize increases and cudaMalloc's Total_time(%) decreases. There is a drastic increase in the Total_time(%) of cudaDeviceSynchronize from 1E6 to 1E7 and similarly a drastic decrease for cudaMalloc.

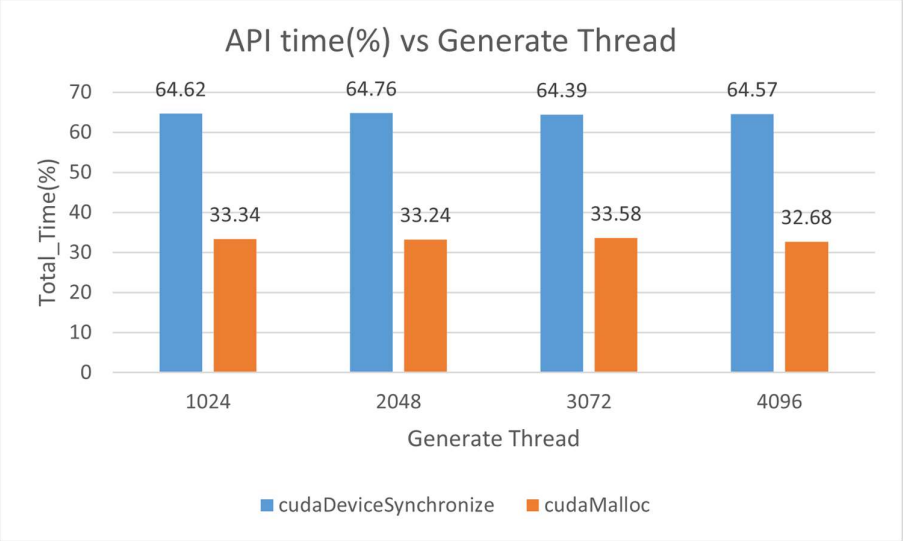
ii) By Varying the Generate Thread:

Here the performance is compared with the varying of Generate Thread by keeping the values of reduceSize and sampleSize as constant.

GPU Activities:

When it comes to GPU activities like generatePoints, reduceCounts and [CUDA memcpy DtoH], the Total_time(%) of reduceCounts and [CUDA memcpy DtoH] are negligible when compared with the Total_time (%) of generatePoints. Like the generatePoints is in the milli seconds to seconds as the sampleSize increases and the rest two activities are in the microseconds. Majority or equal to 100% will be of generatePoints.

API Activities:



Observation:

Although there are many API activities like cudaDeviceSynchronize, cudaMalloc, cudaFree, cudaLaunchKernal etc, cudaDeviceSynchronize and cudaMalloc are the most prominent ones, as their Total_time(%) is nearly approximately above 98%. From the above graph it is evident that as the Generate Thread increases from 1024 to 4096, the cudaDeviceSynchronize and cudaMalloc functions stays at the relatively same value (although small percentage changes) of Total_Time(%). There are no much evident changes among them.

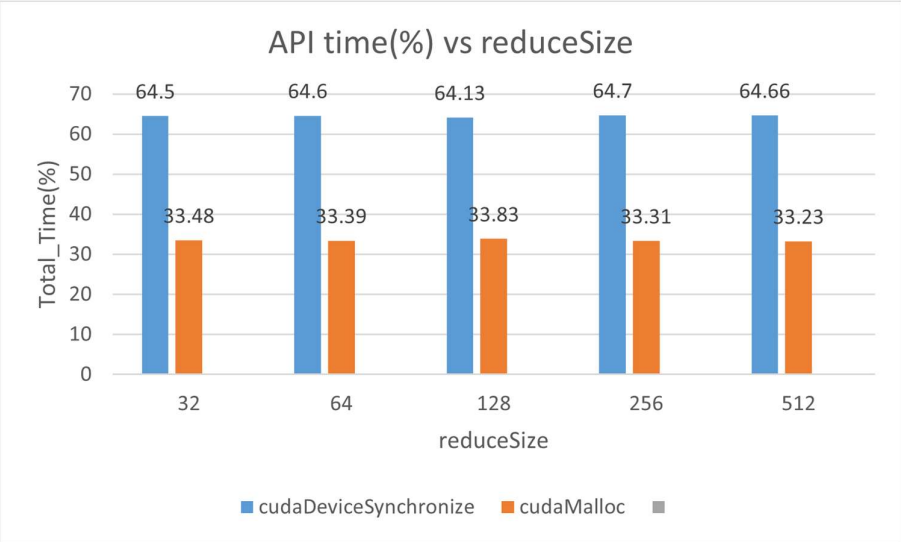
iii) By Varying the reduceSize:

Here the performance is compared with the varying of reduceSize by keeping the values of Generate Thread and sampleSize as constant.

GPU Activities:

When it comes to GPU activities like generatePoints, reduceCounts and [CUDA memcpy DtoH], the Total_time(%) of reduceCounts and [CUDA memcpy DtoH] are negligible when compared with the Total_time (%) of generatePoints. Like the generatePoints is in the milli seconds to seconds as the sampleSize increases and the rest two activities are in the microseconds. Majority or equal to 100% will be of generatePoints. There is a slight change of reduceCounts as it contributes 0.01% of Total_Time when the reduceSize is 512 and for other sizes it is approximately zero.

API Activities:



Observation:

Although there are many API activities like cudaDeviceSynchronize, cudaMalloc, cudaFree, cudaLaunchKernal etc, cudaDeviceSynchronize and cudaMalloc are the most prominent ones, as their Total_time(%) is nearly approximately above 98%. From the above graph it is evident that as the reduceSize increases from 32 to 512, the cudaDeviceSynchronize and cudaMalloc functions stays at the relatively same value (although small percentage changes) of Total_Time(%). There are no much evident changes among them.