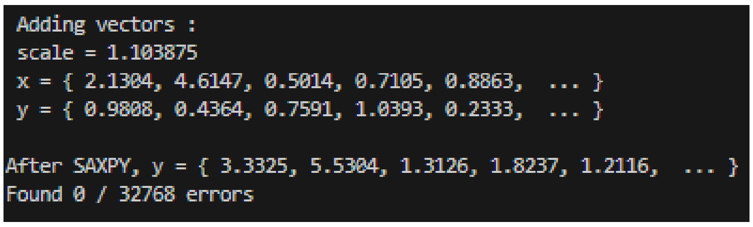
Assignment 1 Report

Name: Ashwin R Kidambi **Github ID: Ashwin28980**

Part A:

Sample Result:

Vector Size = 2^15

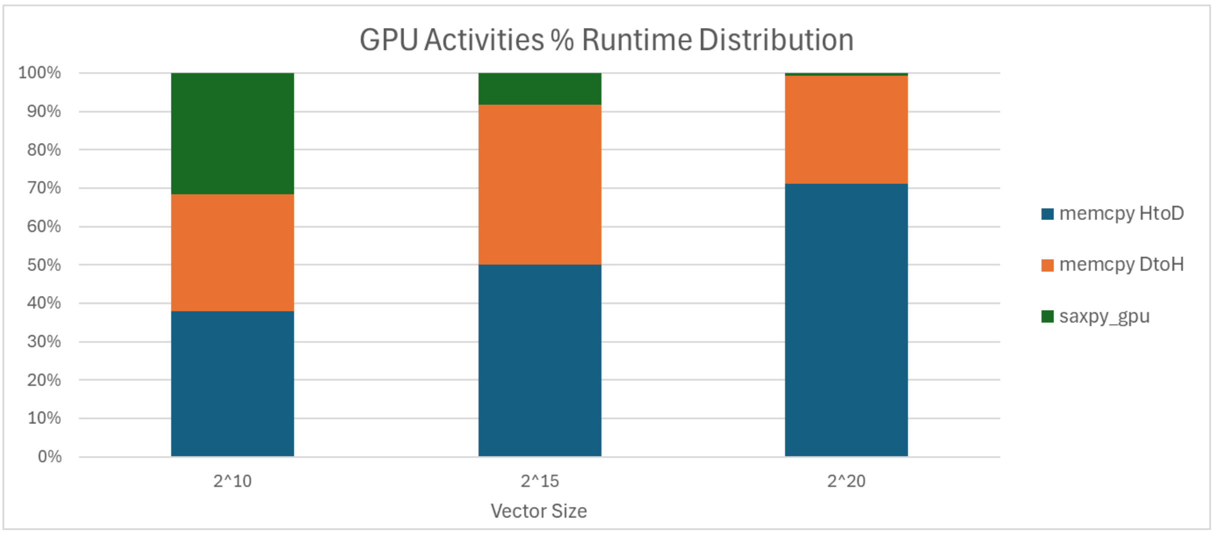


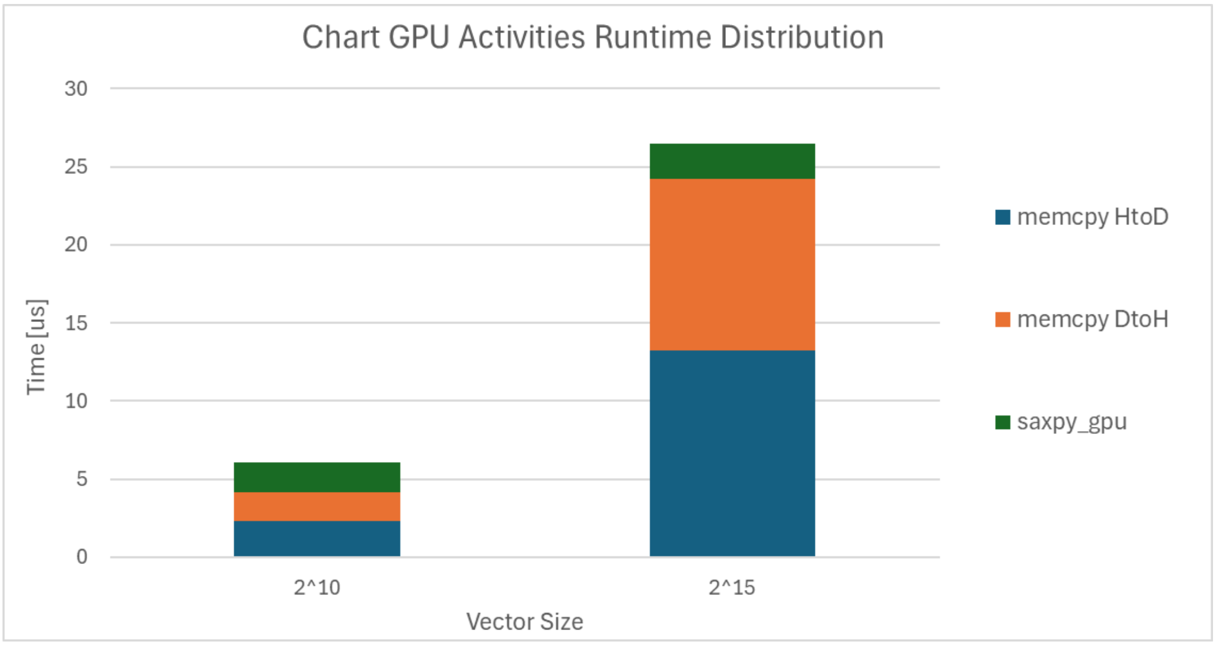
```
 Adding vectors :
 scale = 1.103875
 x = { 2.1304, 4.6147, 0.5014, 0.7105, 0.8863,  ... }
 y = { 0.9808, 0.4364, 0.7591, 1.0393, 0.2333,  ... }

After SAXPY, y = { 3.3325, 5.5304, 1.3126, 1.8237, 1.2116,  ... }
Found 0 / 32768 errors
```

GPU Activities:

Vector Size	**memcpy HtoD [in us]**	**memcpy DtoH [in us]**	**saxpy_gpu [in us]**
2^10	2.304	1.856	1.92
2^15	13.231	11.008	2.208
2^20	1538.7	608.03	16.736



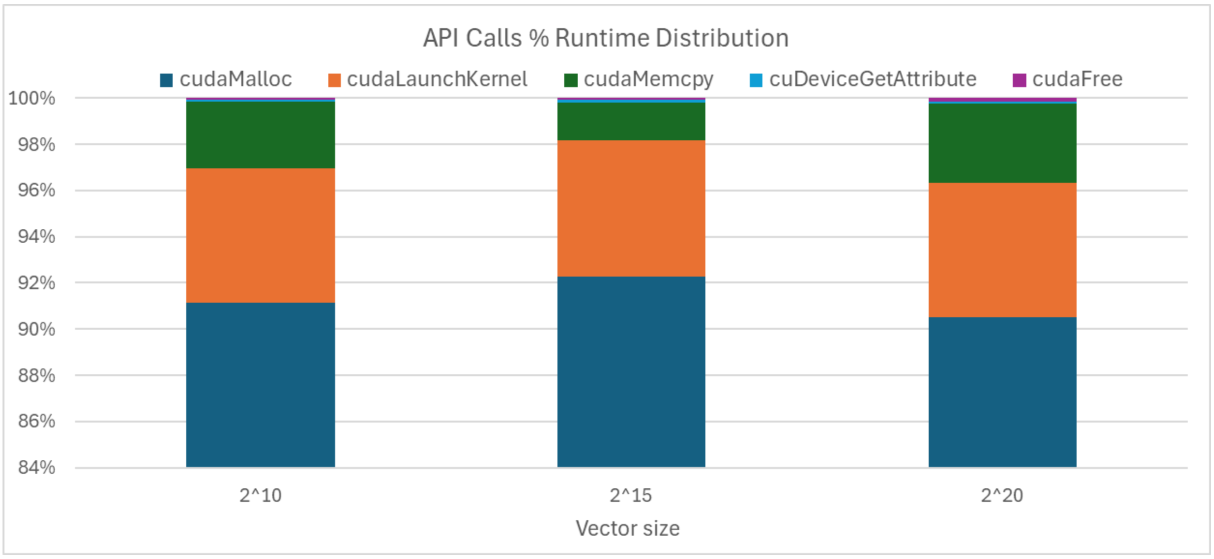


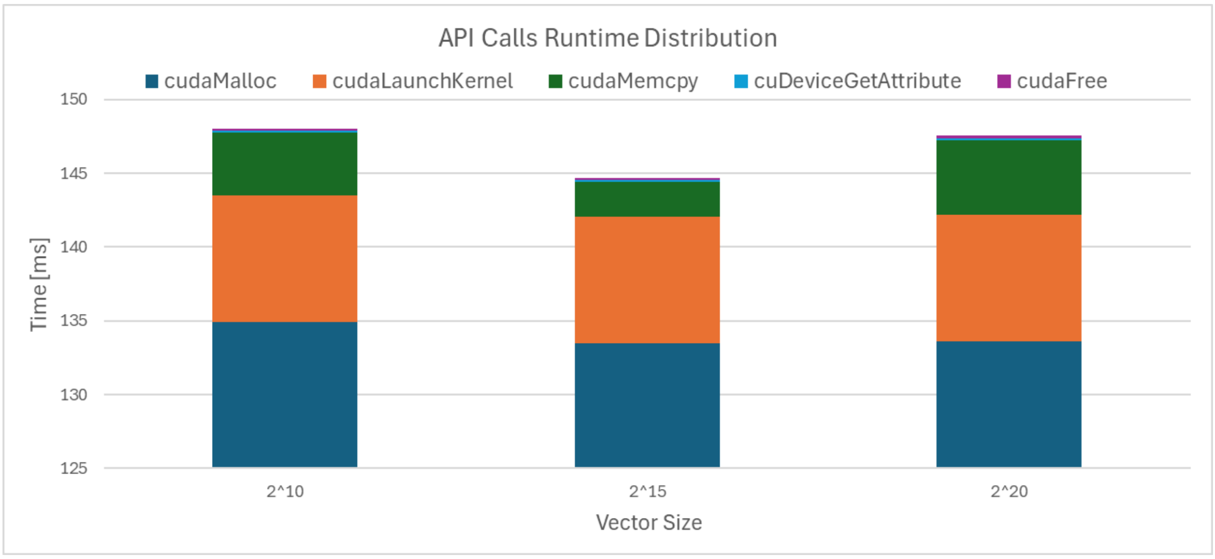
Runtime bar chart not shown for vector size 2^20 since the runtime of that is too large that it would make the other two bars insignificant.

Observations – Runtime for the device & kernel function and memcpy functions scale with the vector size as expected as the computation time depends on amount of data.

API Calls:

Vector Size	**cudaMalloc [ms]**	**cudaLaunchKernel [ms]**	**cudaMemcpy [ms]**	**cuDeviceGetAttribute [ms]**	**cudaFree [ms]**
2^10	134.94	8.5888	4.226	0.1469	0.11863
2^15	133.49	8.5492	2.3687	0.14305	0.12494
2^20	133.61	8.5761	5.0456	0.1316	0.21428



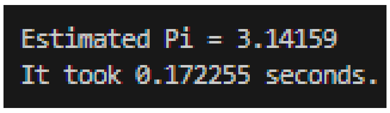


Observations – Runtime for cudaMalloc and cudaMemcpy scale with the vector size as expected as the computation time depends on amount of data.

<u>Part B:</u>

Sample output:

For thread count of 1024 and 1e6 samples per thread



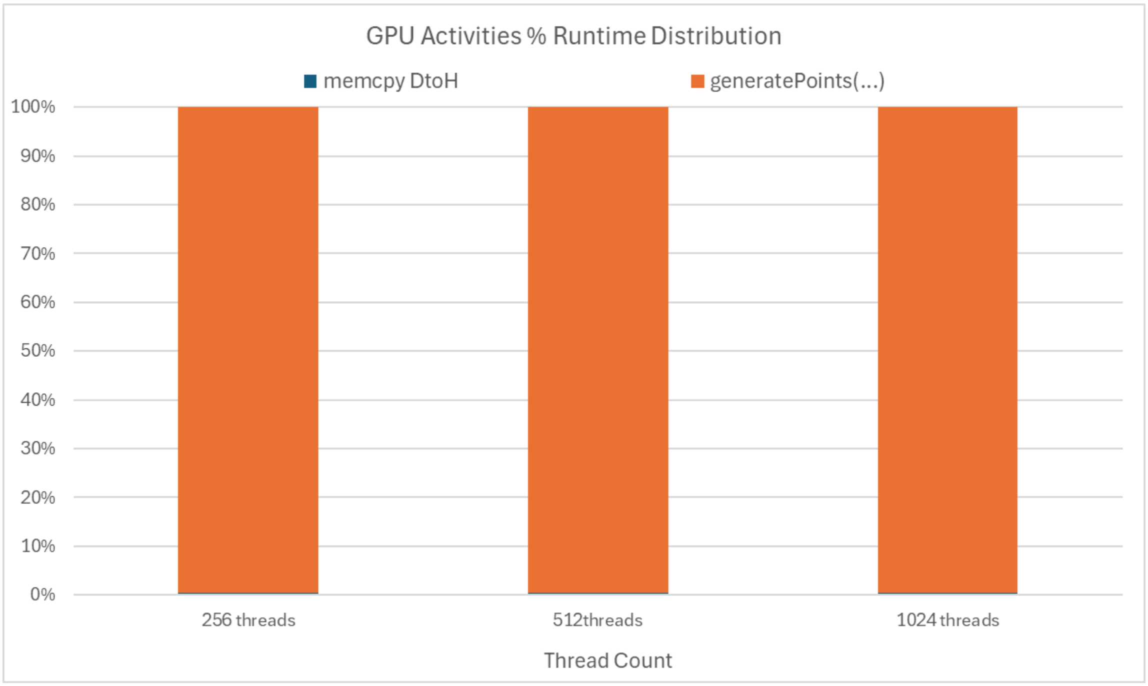
```
Estimated Pi = 3.14159
It took 0.172255 seconds.
```

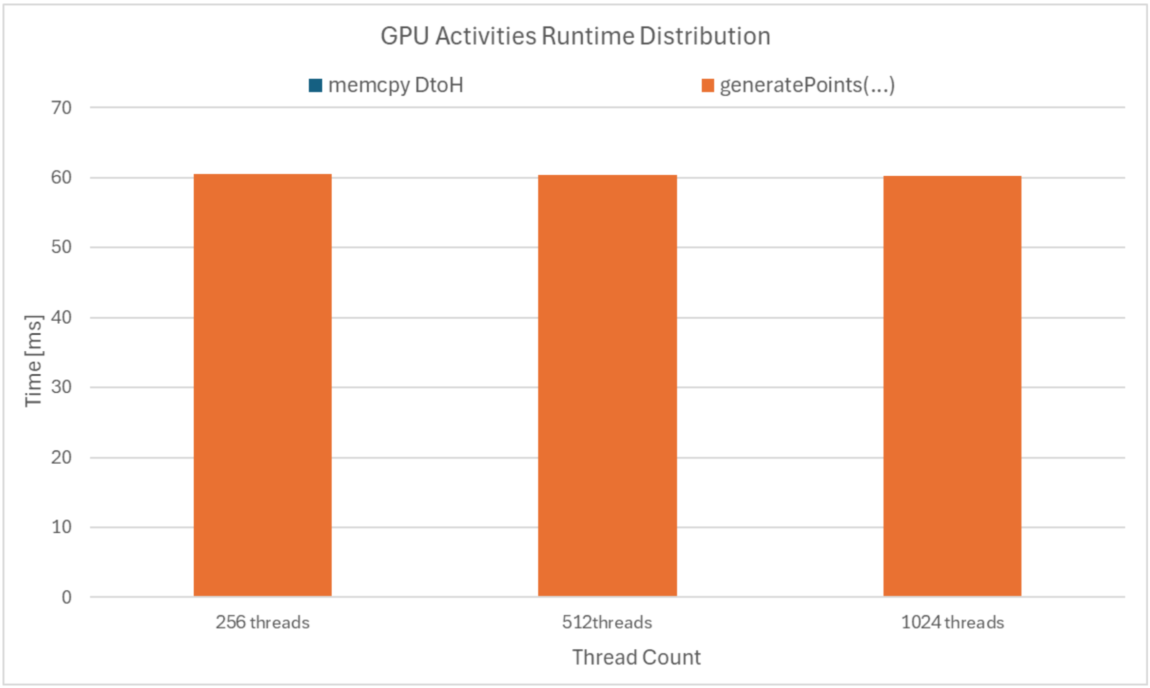
<u>Varying Generated thread count:</u>

This did not affect the accuracy of pi estimated.

GPU Activities:

Thread Count	memcpy DtoH [ms]	generatePoints(...) [ms]
256 threads	0.2048	60.293
512threads	0.2112	60.226
1024 threads	0.2496	59.939

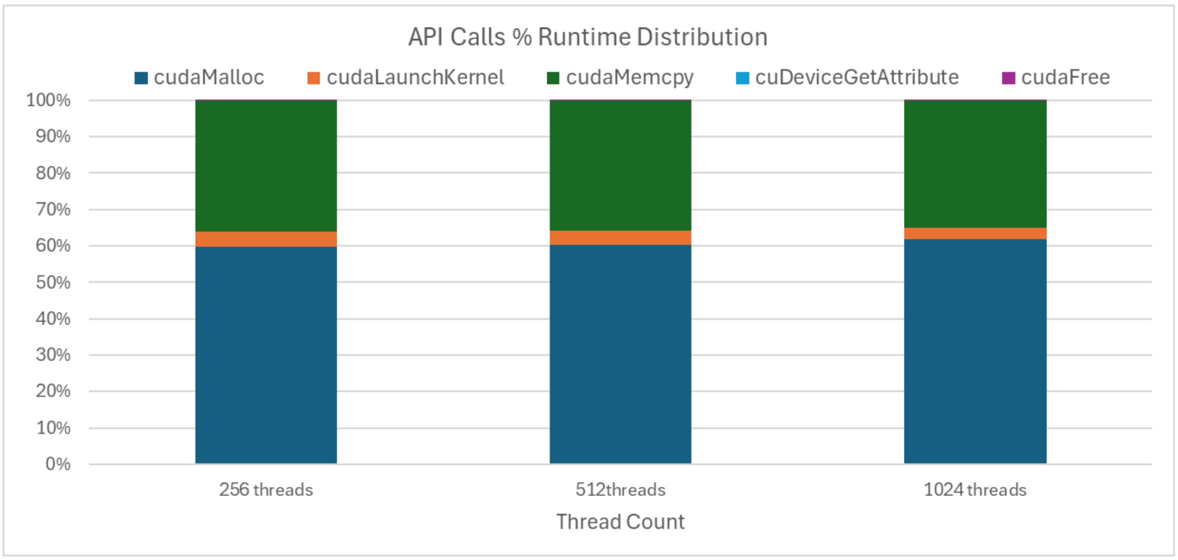


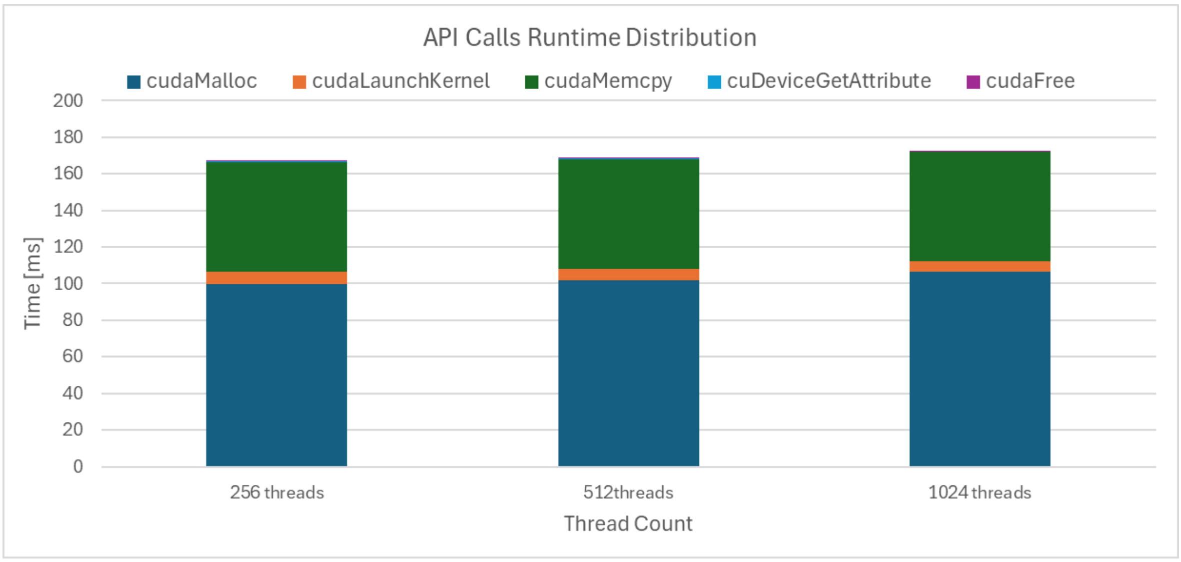


Observations – generatepoints() function essentially takes up most of the runtime and it does not depend on the thread count as there aren't any operations specifically for the assigned generated thread count.

API Calls:

Thread Count	**cudaMalloc [ms]**	**cudaLaunchKernel [ms]**	**cudaMemcpy [ms]**	**cuDeviceGetAttribute [ms]**	**cudaFree [ms]**
256 threads	99.788	6.7933	60.043	0.12836	0.13357
512threads	101.63	6.3802	60.034	0.15214	0.11245
1024 threads	106.69	5.4424	59.94	0.1341	0.11606





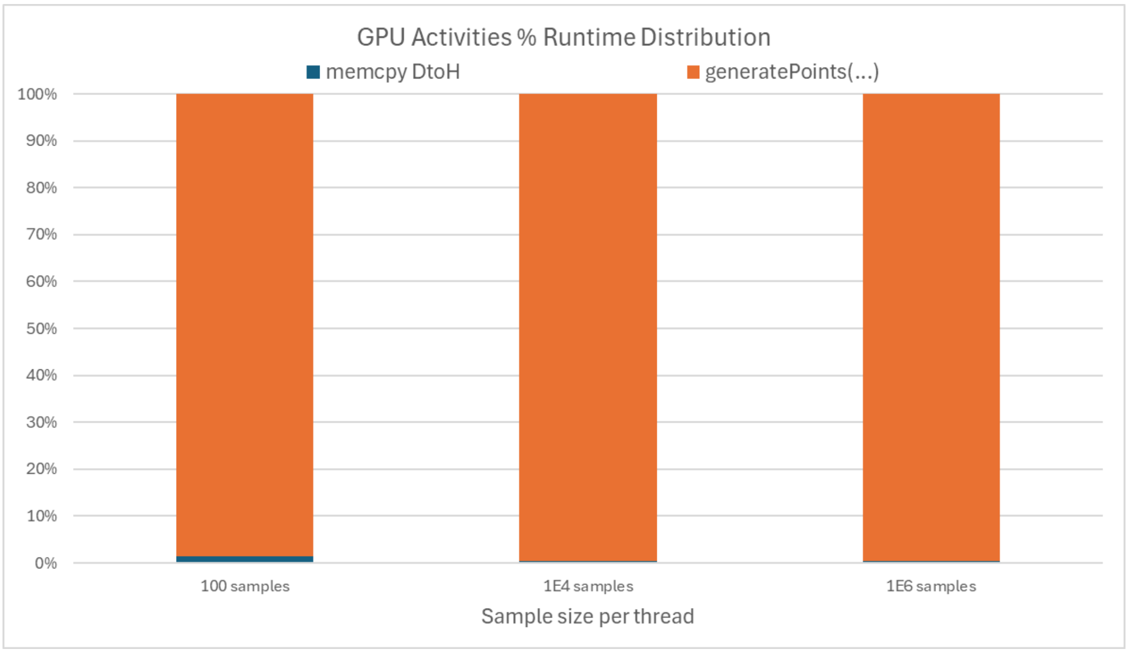
Observations – Runtime does not scale with threads much. Since the operation is embarrassingly parallel and there aren’t many operations, the runtime seems to not scale much with thread count.

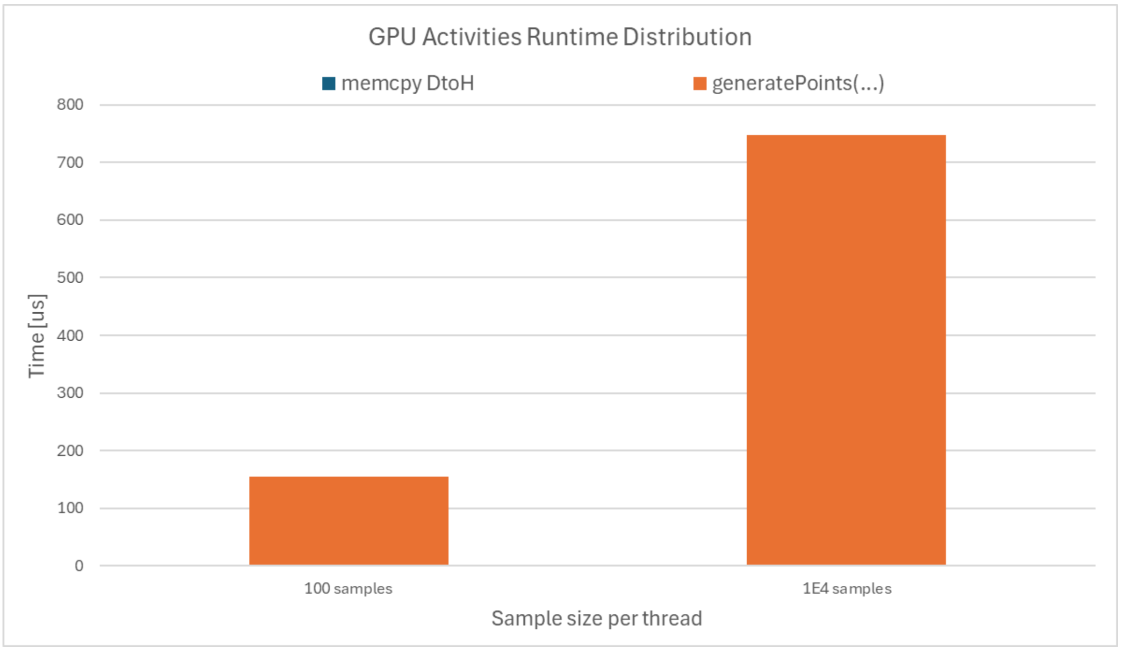
Varying sample points:

Results for 1e6 and 1e4 did not vary much, however for 100 samples, the accuracy started decreasing slightly. For 100 samples, the third decimal was now incorrect occasionally while for the other two, it was accurate for 3 decimals.

GPU Activities:

Samples	**memcpy DtoH [us]**	**generatePoints(...)[us]**
100 samples	2.112	152.26
1E4 samples	2.368	744.51
1E6 samples	249.6	59939



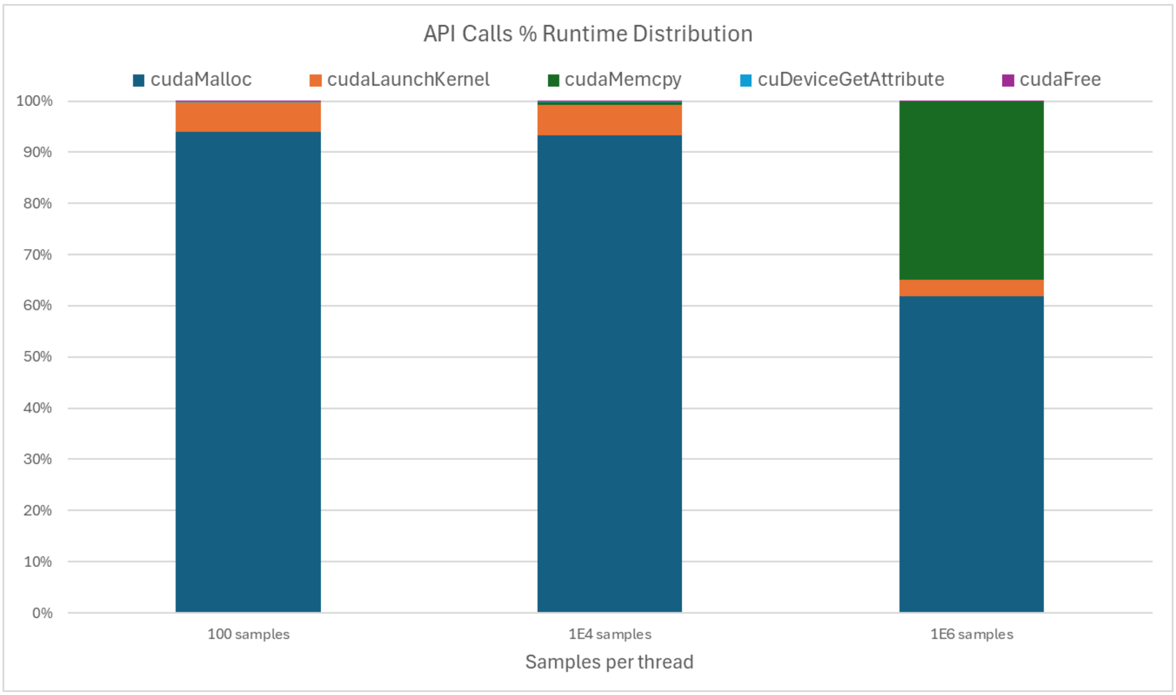


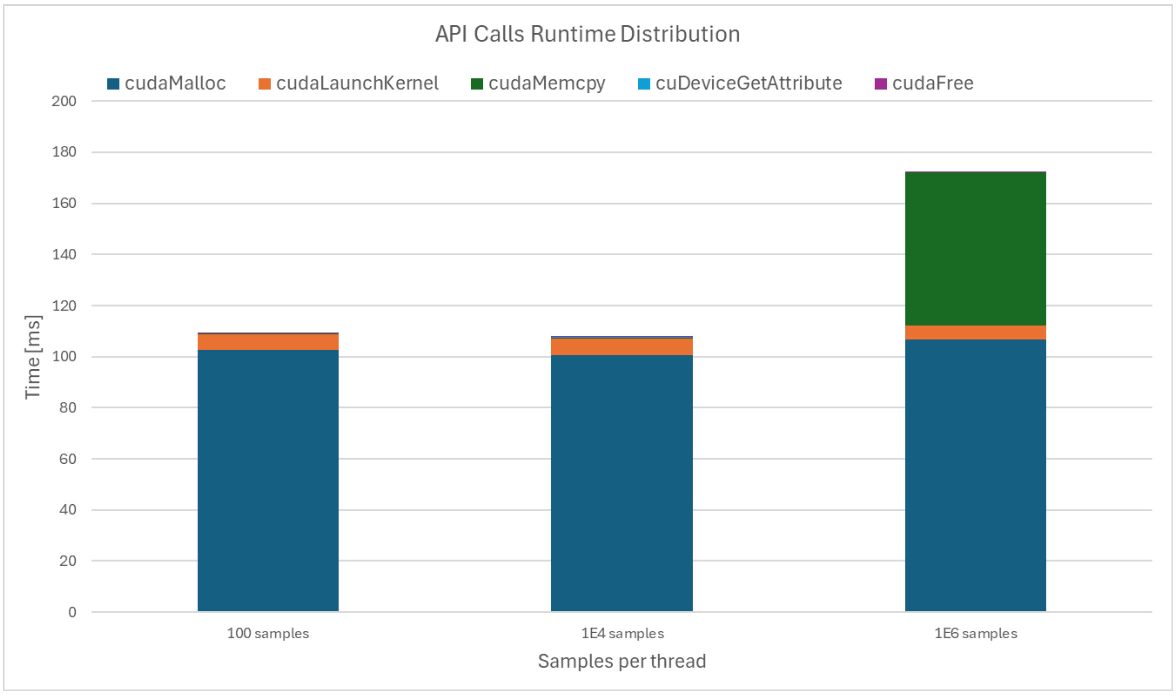
Runtime bar chart not shown for 1e6 sample size since the runtime of that is comparitively too large that it would make the other two bars insignificant.

Observations – Runtime scales with sample size per thread. More data => Longer computation time

API Calls:

Samples	**cudaMalloc [ms]**	**cudaLaunchKernel [ms]**	**cudaMemcpy [ms]**	**cuDeviceGetAttribute [ms]**	**cudaFree [ms]**
100 samples	102.49	6.3947	0.03138	0.12546	0.11368
1E4 samples	100.43	6.4975	0.54147	0.1348	0.11249
1E6 samples	106.69	5.4424	59.94	0.1341	0.11606





Observations – Runtime of cudaMemcpy scales with sample size per thread. More data => Longer computation time. However, there is no significant change in cudaMalloc runtime