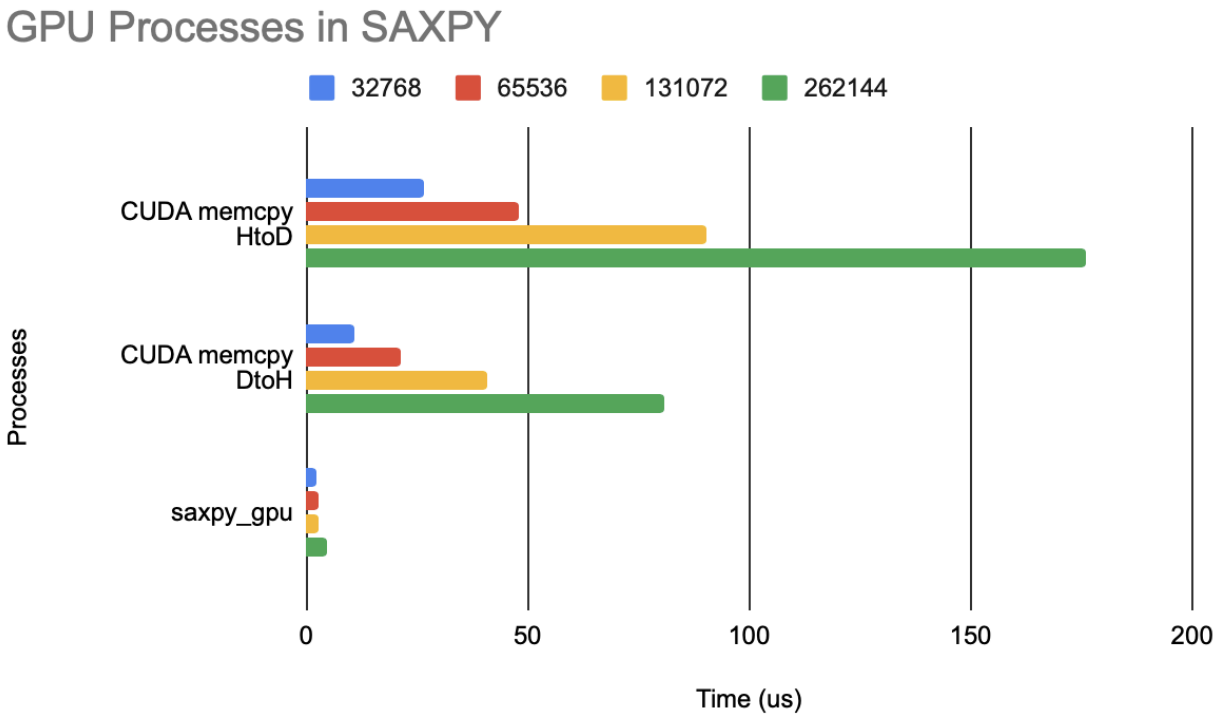
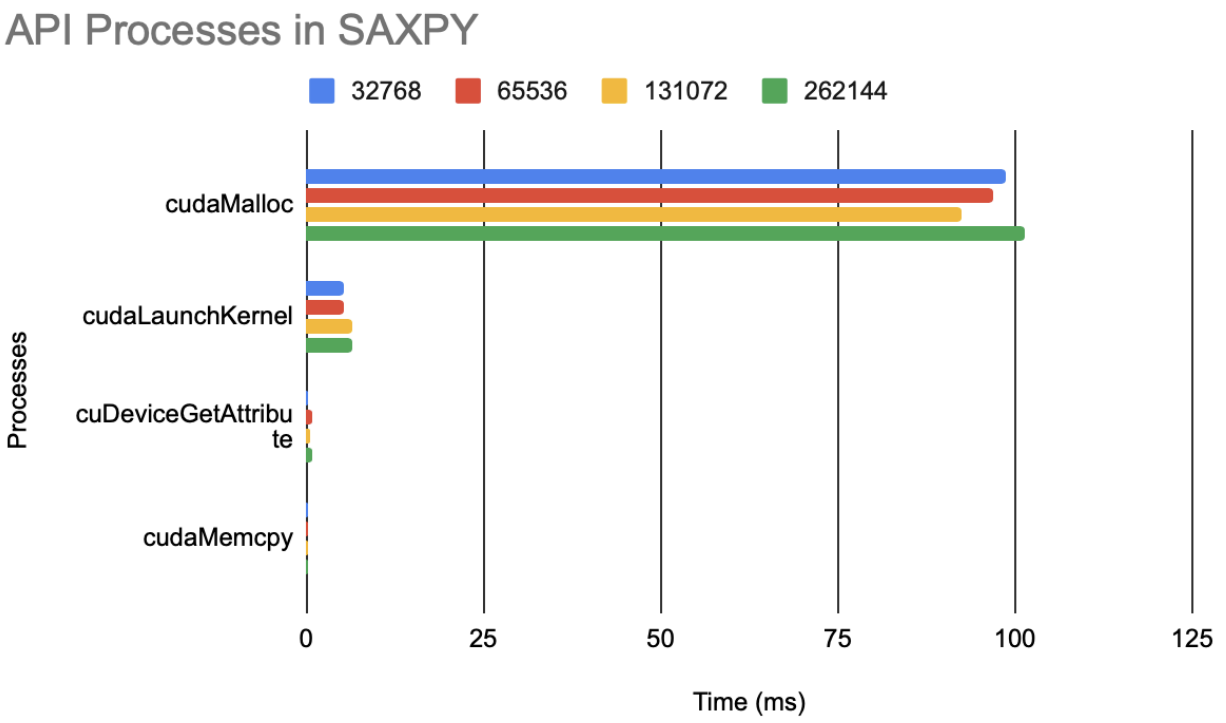
SAXPY Data:

GPU	Time per Process (us)			
Vector Size	CUDA memcpy HtoD	CUDA memcpy DtoH	saxpy_gpu	
32768	26.368	11.104	2.368	
65536	47.872	21.12	2.784	
131072	90.432	40.799	3.008	
262144	175.9	80.799	4.64	
API	Time per Process (ms)			
Vector Size	cudaMalloc	cudaLaunchKernel	cuDeviceGetAttribute	cudaMemcpy
32768	98.833	5.3224	0.155	0.149
65536	96.793	5.4454	0.722	0.239
131072	92.573	6.4908	0.465	0.142
262144	101.35	6.5263	0.986	0.139

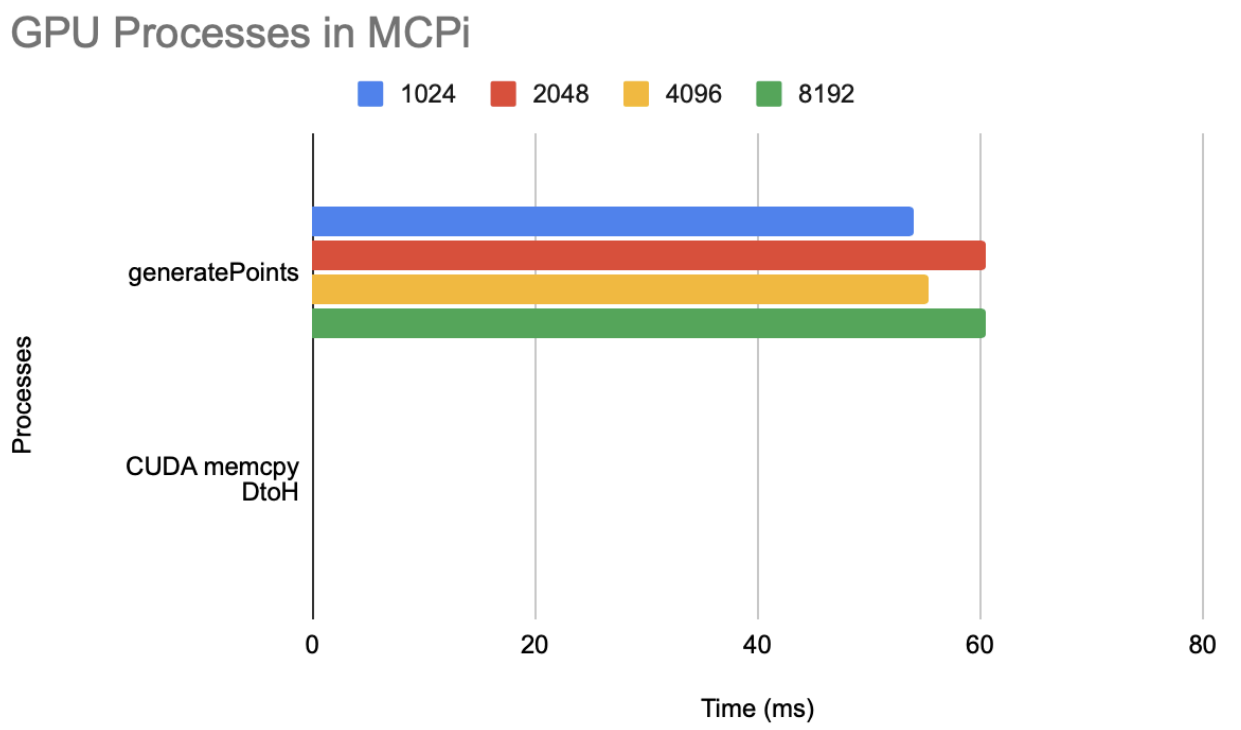


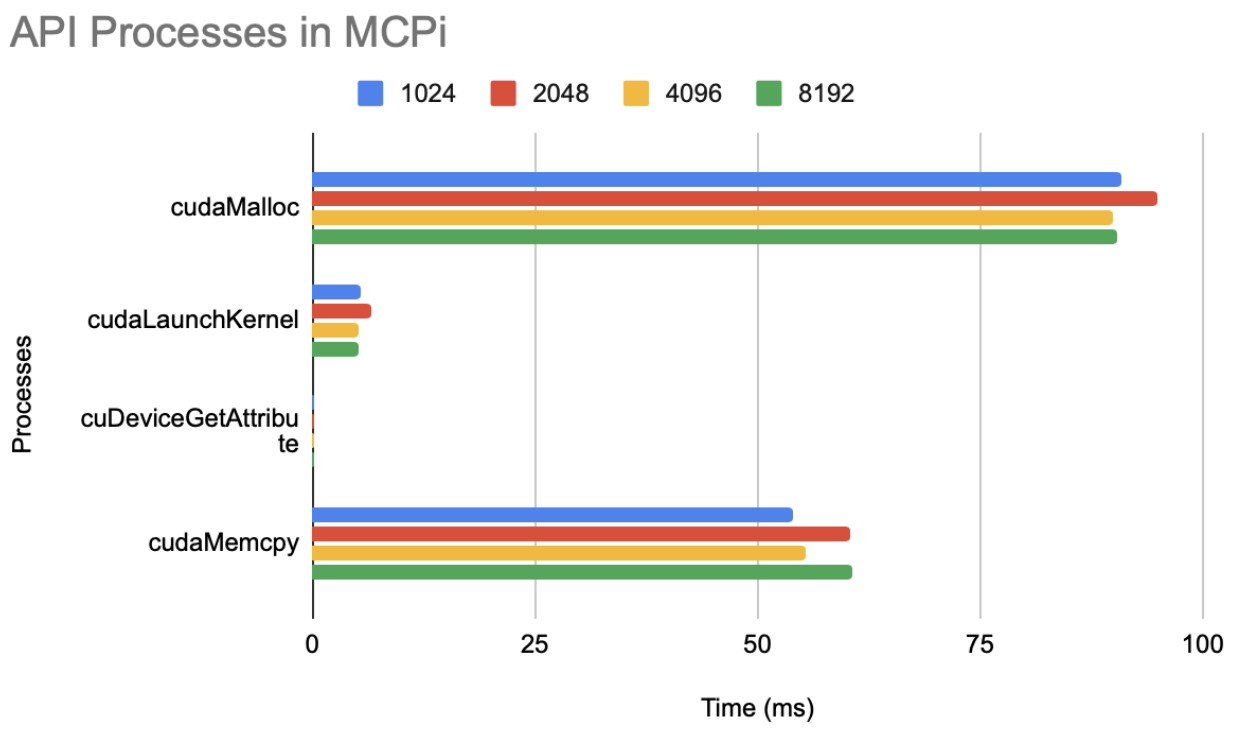


Legend indicates vector size

MCPi Data:

GPU	Time per Process (ms)			
Threads Used	generatePoints	CUDA memcpy DtoH		Difference from pi
1024	26.368	0.00211		0.00002734641021
2048	60.556	0.00266		0.00001265358979
4096	55.381	0.0039		0.000002653589793
8192	60.592	0.00637		0.000002653589793
API	Time per Process (ms)			
Threads Used	cudaMalloc	cudaLaunchKernel	cuDeviceGetAttribute	cudaMemcpy
1024	90.769	5.5536	0.129	54.046
2048	94.815	6.7095	0.125	60.409
4096	89.91	5.1828	0.137	55.414
8192	90.367	5.2566	0.124	60.652





Legend indicates number of threads used

Observations:

Looking at the GPU graphs for SAXPY, we can see that there isn't much variation in the actual runtime of saxpy_gpu as vector size is doubled, but a fairly exponential increase for the two memcpy processes. This makes sense, as saxpy_gpu is at most performing one single calculation and a store. Meanwhile, the memcpy operations had to deal with double the data each time, making them also take twice as long. For API processes, we can see that the times across all vector sizes stay relatively consistent. This also makes sense, as none of the top time consuming processes are time dependent on the size of the vector.

Next, we can look at the graphs for MCPi. To start, data was not collected for variance of sample size within each thread, as this would do nothing but increase the time taken by generatePoints. I was more curious to see how times would be affected by increasing the number of generate threads used (no reduce threads were used). We can see that the memcpy on the GPU side performed similarly to memcpy in SAXPY, where it directly increases with the size of the vector being copied over. In this case, the size was the number of generate threads being used. Our main process generatePoints took significantly longer than SAXPY's, by the order of around a thousand. This makes perfect sense, as instead of one simple calculation, we are generating 2 random numbers, multiplying them, adding them, and performing a check on them before choosing to add to our total hit count for the thread. We can also see that there isn't much variation between the times taken for different numbers of threads used. This also tracks, as the number of threads used shouldn't significantly impact the time taken by each thread. As mentioned before, we can expect to see a straightforward increase in time taken by generatePoints if the sample size of 1e6 was increased. As for API calls, we can see that just like SAXPY, there are no significant changes in time used. As increasing the sample size should only affect the number of loops happening within a thread, it also should have no significant effect on API process times.