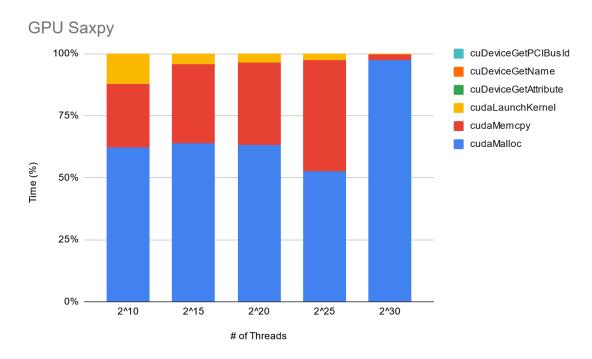
ECE 60827 - Karson Shields Cuda Assignment 1 Part A: GPU Saxpy



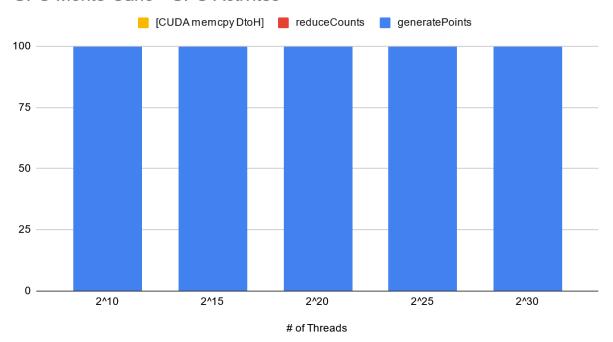
	GPU Saxpy (Time %)											
										cuMo		
				cuDev		cuDevi		cuDev		duleG		
# of	cuda		cudaL	iceGe	cuDevi	ceGet		iceGe		etLoa	cuDev	cuDevice
Threa	Mallo	cudaM	aunch	tAttrib	ceGet	PCIBu	cudaF	tCoun	cuDevice	dingM	iceGe	TotalMe
ds	С	emcpy	Kernel	ute	Name	sld	ree	t	Get	ode	tUuid	m
	62.2											
2^10	9	25.46	12.15	0.08	0.01	0	0	0	0	0	0	0
	63.6											
2^15	9	31.93	4.28	0.09	0.01	0	0	0	0	0	0	0
	63.0											
2^20	9	33.32	3.49	0.08	0.01	0	0	0	0	0	0	0
	52.5											
2^25	2	44.94	2.47	0.06	0.01	0	0	0	0	0	0	0
	97.3											
2^30	5	2.46	0.19	0	0.01	0	0	0	0	0	0	0

When running GPU Saxpy, while increasing the # of threads, the main observation I made is that it is memory bound and ramps quite quickly in the amount of memory time. The main API calls that take up execution time are the memory commands and the kernel launch itself. The rest of the API calls are negligible in execution time, thus depicting that majority of execution

time is in memory operations.

Part B: GPU Monte-Carlo Estimation of Pi





GPU Monte-Carlo - GPU Activities (Time %)								
# of Threads	generatePoints	reduceCounts	[CUDA memcpy DtoH]					
2^10	99.99	0	0					
2^15	99.99	0	0					
2^20	99.99	0	0					
2^25	99.99	0	0					
2^30	99.99	0	0					

GPU Monte-Carlo - API Calls 100% cuDeviceGetUuid cuDeviceTotalMem cudaGetDeviceCount cuModuleGetLoadingMode 75% cuDeviceGet cuDeviceGetCount cuDeviceGetPClBusId 50% cuDeviceGetName cudaFree cuDeviceGetAttribute cudaLaunchKernel 25% cudaMemcpy cudaMalloc 0% 2^10 2^15 2^20 2^25 2^30 # of Threads

	GPU Monte-Carlo - API Calls (Time %)											
			cuDev			cuDev			cuMod	cuda	cuDe	
cuda		cudaLa	iceGe		cuDevice	iceGe	cuDev		uleGet	GetDe	viceT	cuDev
Mallo	cudaM	unchK	tAttrib	cudaF	GetNam	tPCIB	iceGet	cuDevice	Loadin	viceC	otalM	iceGe
С	emcpy	ernel	ute	ree	е	usld	Count	Get	gMode	ount	em	tUuid
57.76	38.83	3.26	0.08	0.06	0.01	0	0	0	0	0	0	0
58.01	38.41	3.23	0.26	0.08	0.01	0	0	0	0	0	0	0
58.09	38.56	3.19	0.08	0.07	0.01	0	0	0	0	0	0	0
56.93	39.85	3.06	0.08	0.07	0.01	0	0	0	0	0	0	0
59.01	37.91	2.92	0.08	0.07	0.01	0	0	0	0	0	0	0

While running GPU Monte-Carlo, the main observation I notice is that the amount of execution time spent in memory is fairly consistent across the # of threads, indicating that this is a compute bound algorithm. The amount of time spent in the GPU is all on generating the points, with the other functions being negligible on execution time despite being important functions on the algorithm. In regards to the API calls, the time spent in memory has slight variations in the different functions being called, but overall the time spent is fairly consistent.