GPUs Hands On 2 — J L Gouws

May 10, 2022 — **19G4436**

Each kernel ran 1000 times. The average runtime was found from NVVP. This usage of the profiler is rather bad. I should have coded the program so that it works out the average execution time for each kernel. Observations:

2D Block	Step 3		Step 4		Step 5	
	2D Grid Size	Execution Time ms	1D Grid size 1 datum per thread	Execution time ms	Grid Size 16 data per thread	Execution time ms
32×32	128×128	2.90074	16384	2.7162	1024	2.7919
32×16	128×256	2.72323	32768	2.70327	2048	2.7999
16×32	256×128	2.78609	32768	2.70757	2048	2.80921
16×16	256×256	2.75102	65536	2.72549	4096	2.80999
64×16	64×256	2.89524	16384	2.71913	1024	2.80161
16×64	256×64	2.92079	16384	2.71883	1024	2.83459
64×8	64×512	2.71029	32768	2.74134	2048	2.82161

Table 1: Average execution times of Kernel

I have realized that I only recorded three of my own block sizes; I do not have access to my Lab Computer at the time of writing this. Please excuse the lack of extra experiments.

On average, the performance seems to be worse with larger block sizes. Larger block sizes result in fewer blocks being launched. Fewer blocks being launched can result in unoccupied streaming multiprocessors. Less usage of the streaming multiprocessors results in fewer FLOPS–that is lower performance.

When doing 16 data items per thread, the threads must stride across the data in order to acheive decent performance. If no striding is done, the kernel performs poorly. This performance reduction is due to memory access in the GPU. I think that striding through the data allows the kernel to use the spatial locality of cache. This use of the cache's spatial locality would increase perfomance, because new data items do not have to be fetched from gobal memory every time one or two warps are executed.

With striding, all the kernels seem to perform similarly. The sixteen data per thread category seems to perform the worst on average. The one dimensional grid seems to perform the best on average.