תרגיל בית 2 מאיצים:

ספיר מלכה: 205794001

אלדד וינר: 304901069

1. CUDA Streams:

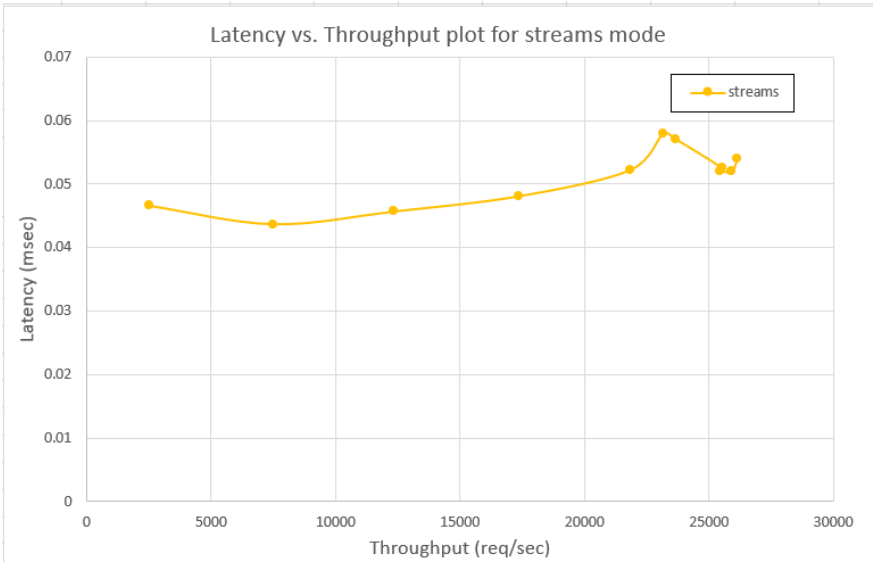
1.1 code

1.2 running the command "./ex2 streams 0" yielded:
throughput = 26086 (req/sec)

1.3

Load	Throughput (req/sec)	Latency (msec)
2500	2499	0.0466
7500	7491	0.0437
12500	12358	0.0457
17500	17378	0.0481
22500	21819	0.0522
27500	23149	0.0579
32500	23674	0.057
37500	25521	0.0525
42500	25465	0.052
47500	25919	0.0521
50000	26134	0.054

1.4



We can observe a small increase in latency as the throughput increases. This can be explained by the fact that, in order to increase the throughput, we have more GPU resources in use at the same time, so new requests might have to wait longer before resources are available to start handling them.

2. Producer Consumer Queues:

2.1 first we noted the amount of shared memory that we use per block.
For each kernel invocation we used 2 int arrays and 1 uchar array (histogram, hist_min, map), all of size 256, so $2 \cdot 4 \cdot 256 + 256 = 2304$

So, each thread block requires: 32 regs, 2304 shared memory bytes, and the amount of threads required per block by the user.

Overall, for each SM, we can calculate how many thread blocks it can support by:

$$BlocksPerSM = min(\frac{ThreadsPerSM}{threadsPerBlock}, \frac{shmemPerSM}{shmemPerBlock})$$

So in total we can support:

$$Total\ blocks = \#SM's \cdot BlocksPerSM$$

2.4.1 running the command "./ex2 queue 1024 0" yielded:
throughput = 25862 (req/sec)

2.4.2

Load	Throughput (req/sec)	Latency (msec)
2500	2516	0.68
7500	7572	0.66
12500	12728	0.68
17500	17703	0.71
22500	22617	0.89
27500	25543	7.88
32500	26188	8.49
37500	26134	8.71
42500	25910	8.88
47500	26053	8.87
50000	25989	8.91

2.5.1 running the command "./ex2 queue 512 0" yielded:
throughput = 50644 (req/sec)

2.5.2

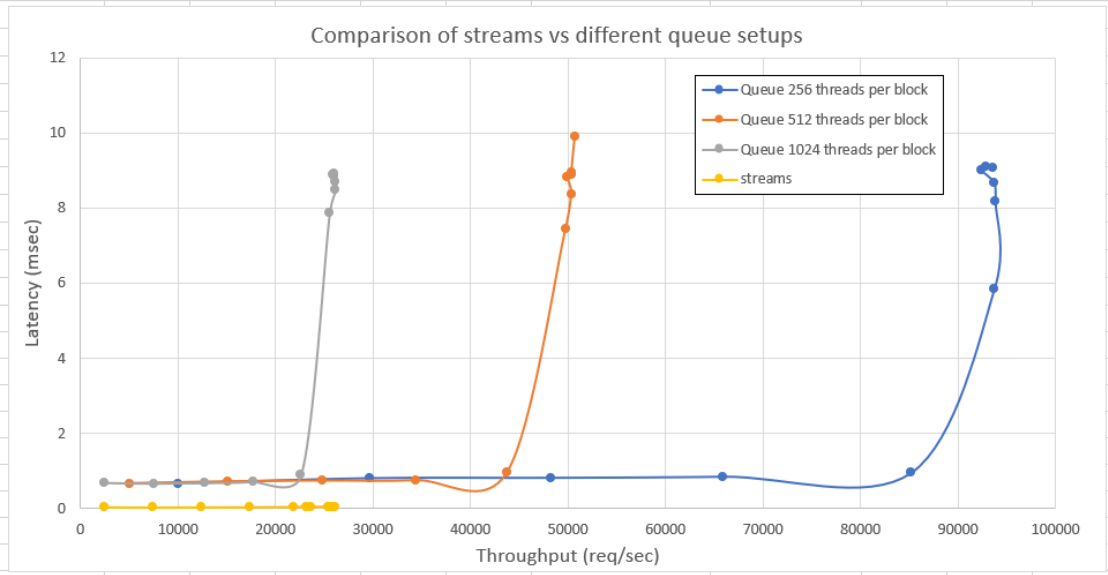
Load	Throughput (req/sec)	Latency (msec)
5000	5089	0.67
15000	15086	0.72
25000	24845	0.74
35000	34391	0.75
45000	43804	0.98
55000	49773	7.46
65000	50351	8.38
75000	49945	8.83
85000	50449	8.89
95000	50360	8.95
100000	50707	9.912

2.6.1 running the command "./ex2 queue 256 0" yielded:
throughput = 94878 (req/sec)

2.6.2

Load	Throughput (req/sec)	Latency (msec)
10000	10011	0.67
30000	29672	0.811
50000	48213	0.82
70000	65950	0.85
90000	85200	0.95
110000	93711	5.84
130000	93827	8.17
150000	93730	8.67
170000	92377	9.02
190000	92904	9.09
200000	93577	9.07

2.4.3 + 2.5.3 + 2.6.3



2.7 we can see in all 3 graphs, that as long as we send the jobs at a rate lower than the max throughput, we see a pretty stable and consistent latency,

and when we surpass that rate, we see an exponential increase at around the same rates for all 3 thread count choices.

We also see that since the kernel does not benefit much from using over 256 threads, using those threads in other thread blocks helped much more, hence increasing the performance when we decreased the threads per block count.

2.8 We would expect better performance because, the CPU only writes to the CPU to GPU queue, and the GPU only reads from it.

Since over PCIe, reads are non-posted, the GPU has to wait for each read to complete.

On the other hand, the writes are posted, so the CPU does not have to wait for the writes to complete, and will feel the added latency much less.

2.9 the GPU would have to expose a part of it's global memory to MMIO over PCIe with a bar. That bar would be sent to the CPU, then the CPU's mmu would have to map that physical address onto a virtual address which would be used as the queue's address.