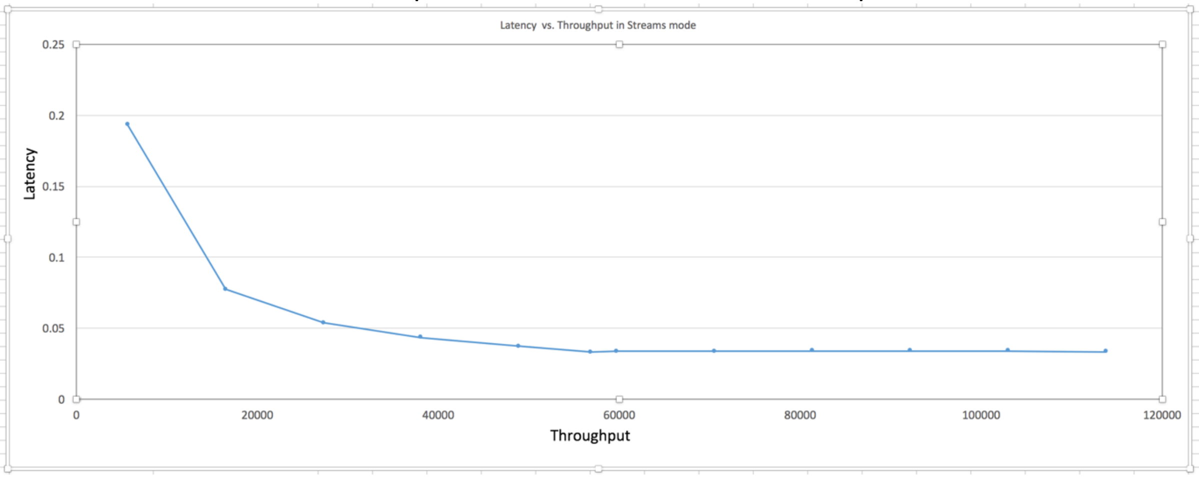
1.

1.4 Maximal throughput reached by “./hw2 streams 0” was: 56898 req/sec.
The result graph for different loads shows that streams helps optimally load GPU.
In underload case we observe higher latencies as result of synthetic delays in our simulated client-server environment. Once reached maximal throughput the latency doesn’t grow, it shows that in our simulation request overload doesn’t affect GPU performance.

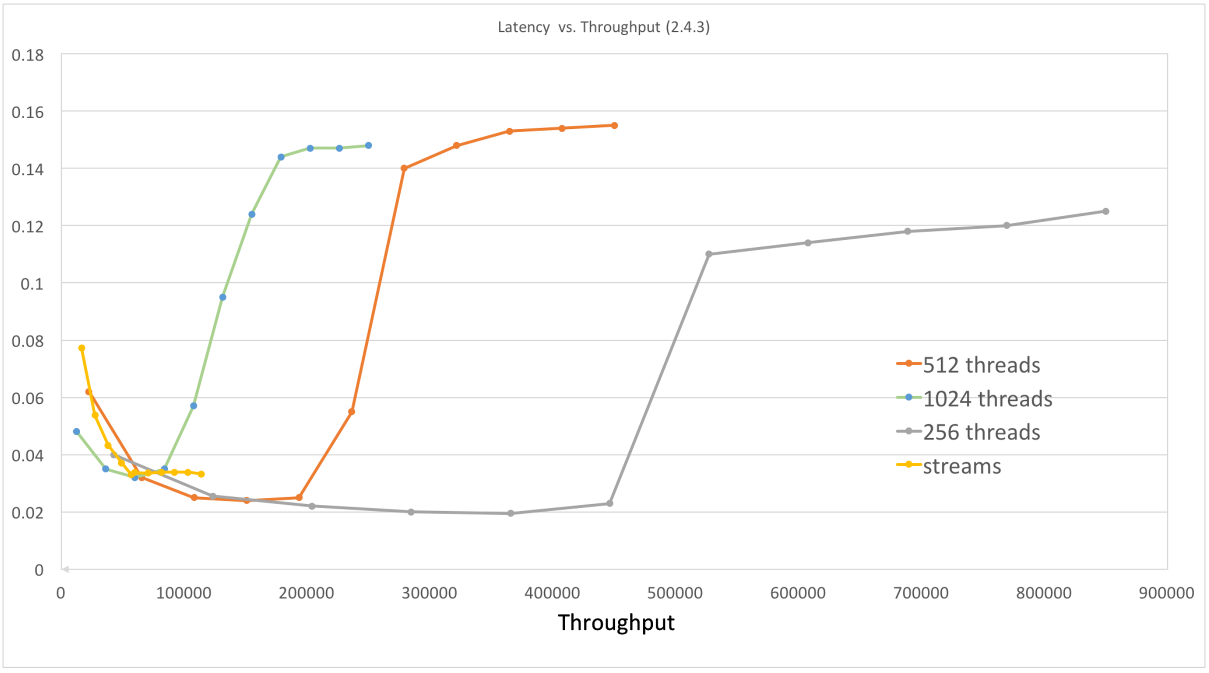


2.

2.1 In order to get maximal possible number of TBs, I check if:

1. Is single TB could contain 1024 threads by maxThreadsPerBlock property of device.
2. Check if it has enough registers by regsPerBlock property.
3. Check if it has enough shared memory per TB by sharedMemPerBlock property.

2.4.3



2.7

The graph show that starting from certain throughput the queue mode become ineffective because of increased latency. The best results shows queue mode with 256 threads per TB due to optimal load of SMs (8 queues with 256 threads per TB).

2.8
Good idea, the consumer-producer queue should be placed in consumer physical memory because PCIe read transaction is longer than write transaction, in other words GPU-CPU queue should be created in CPU RAM and CPU-GPU queue should be in GPU physical memory.

2.9
Physical memory on CUDA device could be accessed as MMIO by one device’s BAR, additionally its physical memory address should by translated to virtual addresses in user-space by OS (CUDA driver).

Notes:
1.
I couldn’t get in part 2 (queue mode) result similar to CPU and Serial GPU calculations.
I’m sure that calculations are correct, but lack of correct synchronization causes such behavior, please see comment on line 309. Please suggest what I have missed?

2. I failed to run in queue mode on Windows 10, something prevents GPU to access mapped physical CPU memory and queues couldn’t be managed in right manner.