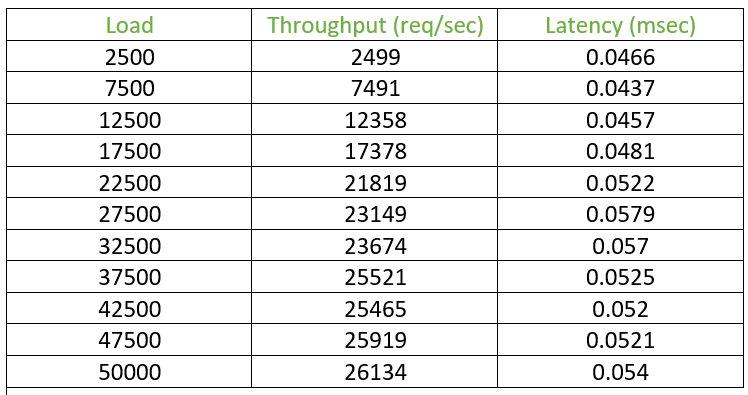
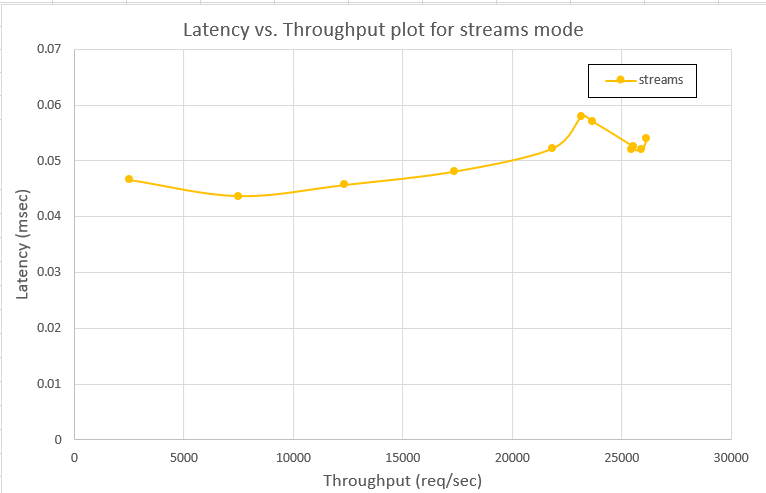
תרגיל בית2 מאיצים:

ספיר מלכה: 205794001

אלדד וינר: 304901069

1. **CUDA Streams:**
   1. code
   2. running the command “./ex2 streams 0” yielded:  
      throughput = 26086 (req/sec)
   3. 
   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.

1. **Producer Consumer Queues**:

**2.1** first we noted the amount of shared memory that we use per block.

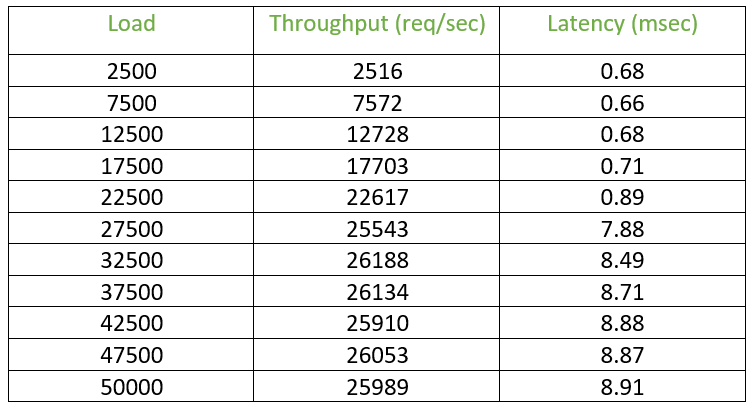
For each kernel invocation we used 2 int arrays and 1 uchar array (histogram, hist\_min, map), all of size 256, so

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

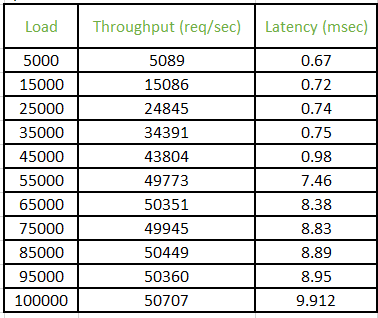
Overall, for each SM, we can calculate how many thread blocks it can support by:

So in total we can support:

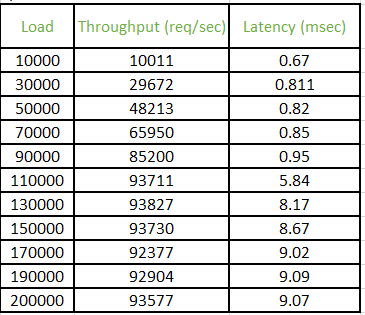
**2.4.1** running the command “./ex2 queue 1024 0” yielded:  
throughput = 25862 (req/sec)

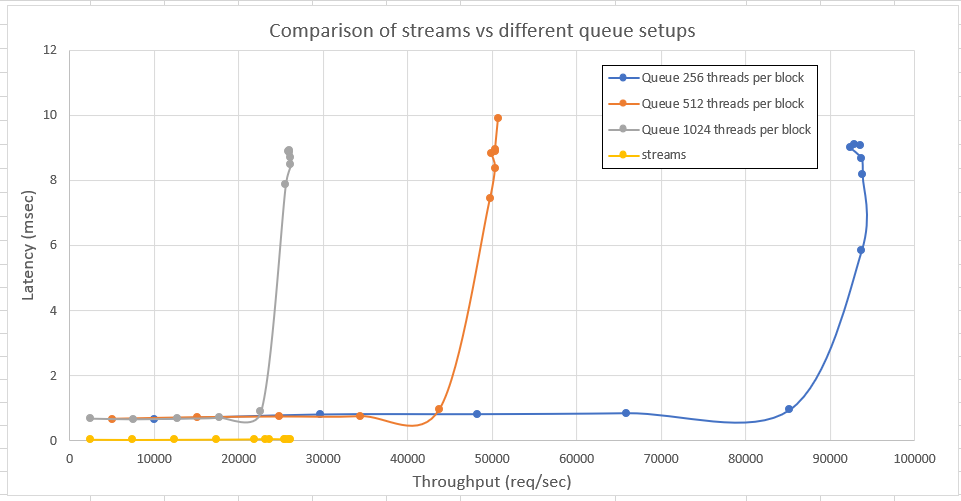
**2.4.2**

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

**2.5.2**

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

**2.6.2**

**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.