1 Machine Info

I submitted the code I wrote to the DGX server as a batch job. The DGX server ran the program with:

```
GPU Device 0: Tesla V100-DGXS-32GB with compute capability 7.0
```

2 Patterns Seen and Discussion

As a general trend, the performance decreased as the amount of trials ran was doubled. There are some outliers, though. Figure 1 shows the GigaTrials/sec (performance) vs. Number of Trials ran. There are certain spikes, such as 64K trials ran with a block size of 128 and 32. There is another spike on 256K trials with a block size of 32, as well.

Why do the patterns seem this way?
One thing that I'm still grappling with is that the Tesla V100 has 84 streaming multiprocessors (`SM`'s) each with four (4) blocks capable of running 32 threads each, so why don't we see performance increasing as we utilize more and more of the GPU? I'm not exactly sure, and I'm reading up on the architecture to eventually find out. I'm assuming that we start to reach the maximum number of floating points operations the GPU can handle per block. There is less 32-bit floating point cores than there are threads available (5376 FP32 Cores vs 10752 total available threads). The best explanation that I can come up with at this point is that there is a much larger overhead to what we are executing on the GPU for this program.

Why does a BLOCKSIZE of 16 perform worse?
One other trend that can be seen is that a block size of 16 (almost) always performs worse than every other block size. You can see this in Figure 2. This is because there are 32 threads in a `warp` on a GPU. If you run the Monte Carlo simulation with a `block size` (which is really number of threads per block) of 16, it won't even fill up a single `warp`. However, if you have 64 threads, for example, you have 2 warps worth of threads, allowing you can fill up each block to the max.

How does this compare to Project 1 Results?
An interesting difference between `Project #5`'s performance vs. `Project #1`'s is that performance is *decreased* as the number of trials *increased*. In `Project #1`, the performance is *increased* as the number of trials *increased*. The hypothesis mentioned earlier (the large overhead) might be the reason why we are seeing this behavior. CPU's are probably designed to devour the logic in `MonteCarlo()` while running on a GPU the overhead starts to eat into performance as the number of trials increased. Don't get me wrong, the actual performance in terms of operations per second is **MUCH** larger (factor of 100-1000).

3 Results: Graphs

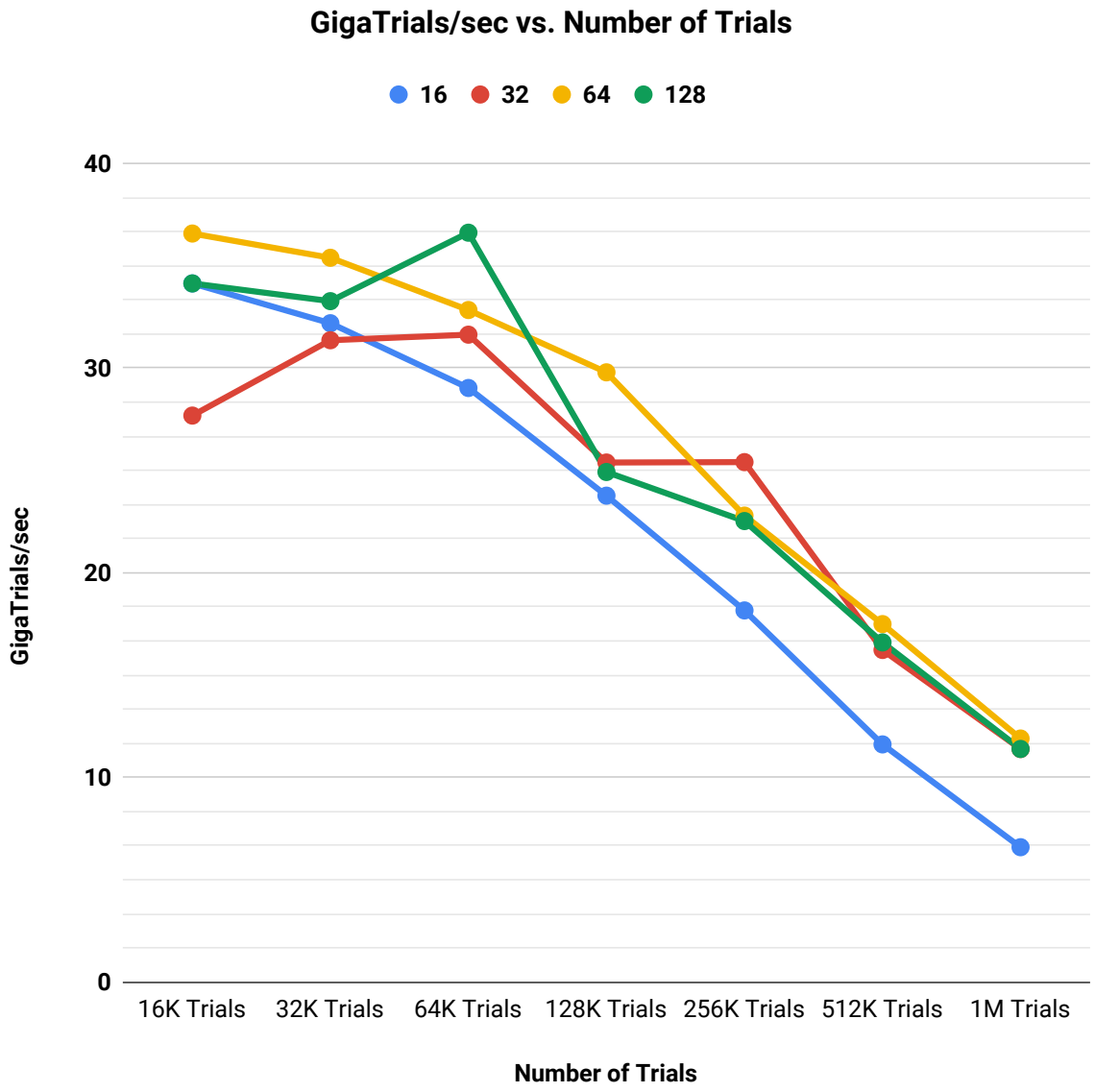

Figure 1: Performance vs. NUM_TRIALS with multiple curves of BLOCKSIZE.

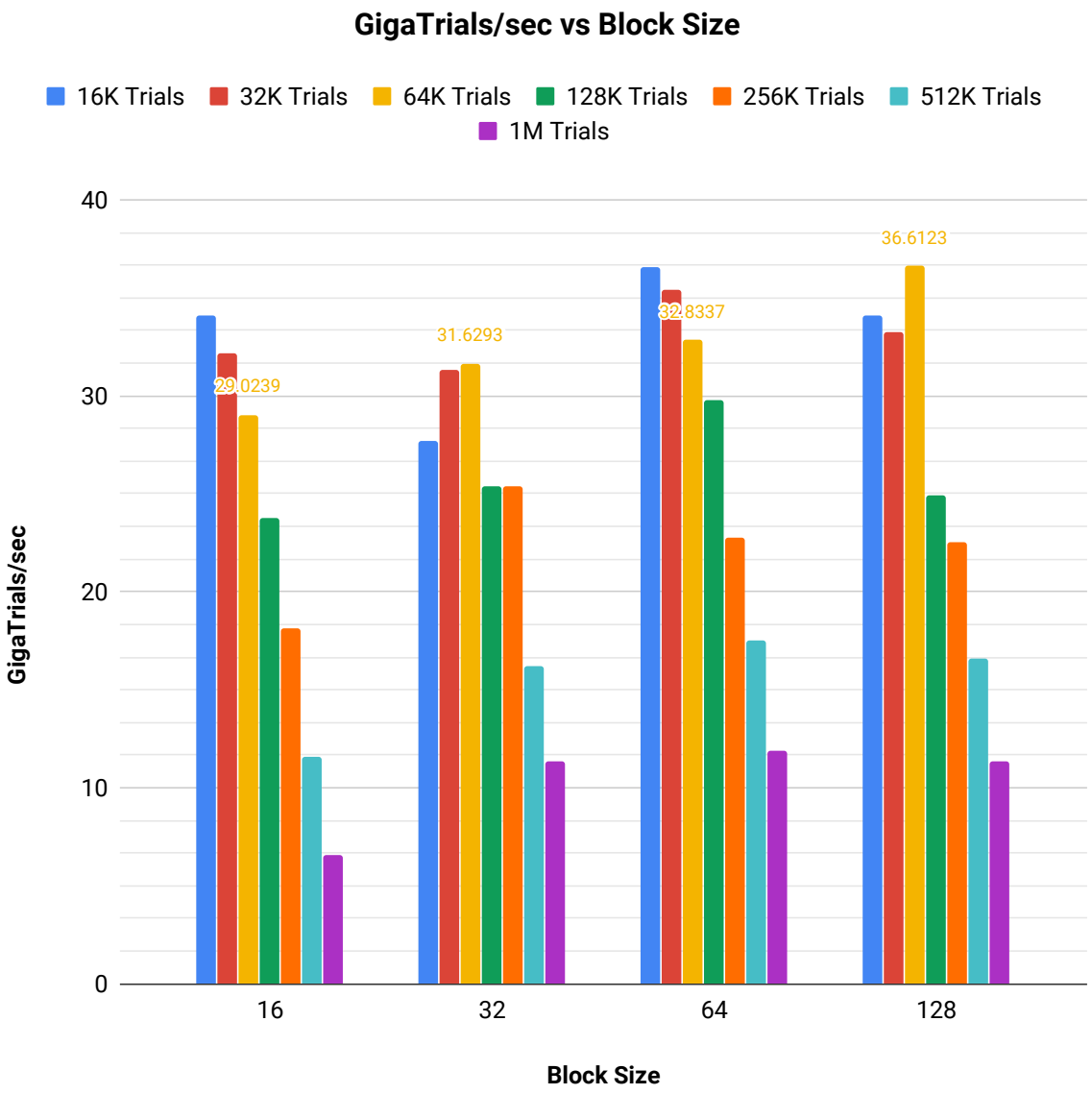

Figure 2: Performance vs. BLOCKSIZE with multiple curves of NUM_TRIALS.

4 Results: Tables

Table 1 shows the results of the CUDA Monte Carlo simulation.

Block Size	Number of Trials	GigaTrials/sec
16	16384	34.1333
16	32768	32.1886
16	65536	29.0239
16	131072	23.7621
16	262144	18.1540
16	524288	11.6075
16	1048576	6.5839
32	16384	27.6757
32	32768	31.3569
32	65536	31.6293
32	131072	25.3819
32	262144	25.4015
32	524288	16.2218
32	1048576	11.3817
64	16384	36.5714
64	32768	35.3866
64	65536	32.8337
64	131072	29.7891
64	262144	22.7872
64	524288	17.4856
64	1048576	11.8940
128	16384	34.1333
128	32768	33.2670
128	65536	36.6123
128	131072	24.9186
128	262144	22.5210
128	524288	16.5914
128	1048576	11.3857

Table 1: Results of the Monte Carlo Simulation.

5 What Does This Mean for Proper Use of GPU Parallel Computing?

Essentially, you should try to make sure your code running on the GPU is as serial as possible. There should be very little logic and a lot of mathematical operations. The logic in the Monte Carlo simulation might actually mess up the performance of the GPU because different threads are starting and ending at the same time. For example, you end a thread because it failed the first if statement but another went all the way through. Actually... now that I say that, I should try removing if statements from the kernel!

5 minutes later...

So I tried removing the if statements in the kernel, and it actually ran slower! Well, there goes that hypothesis.