Jeremy Udarbe - udarbej@oregonstate.edu
CS 475 - parallel programming
Project 5 - CUDA Monte Carlo Simulation

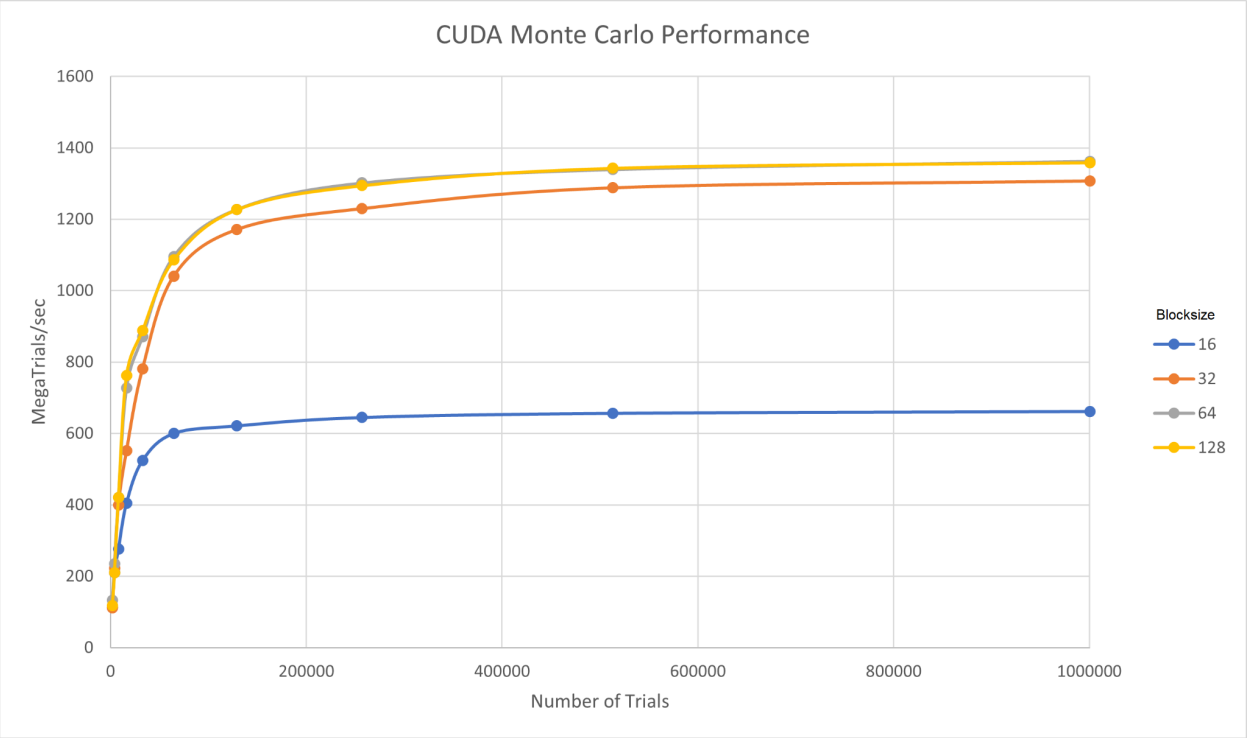
1. The program was compiled with Visual Studio 2019 on PC using a MSI GeForce GTX 1070 8G Graphics Card with 1920 CUDA cores.

2. a. table of performances in MegaTrials/sec computed from number of trials by blocksize

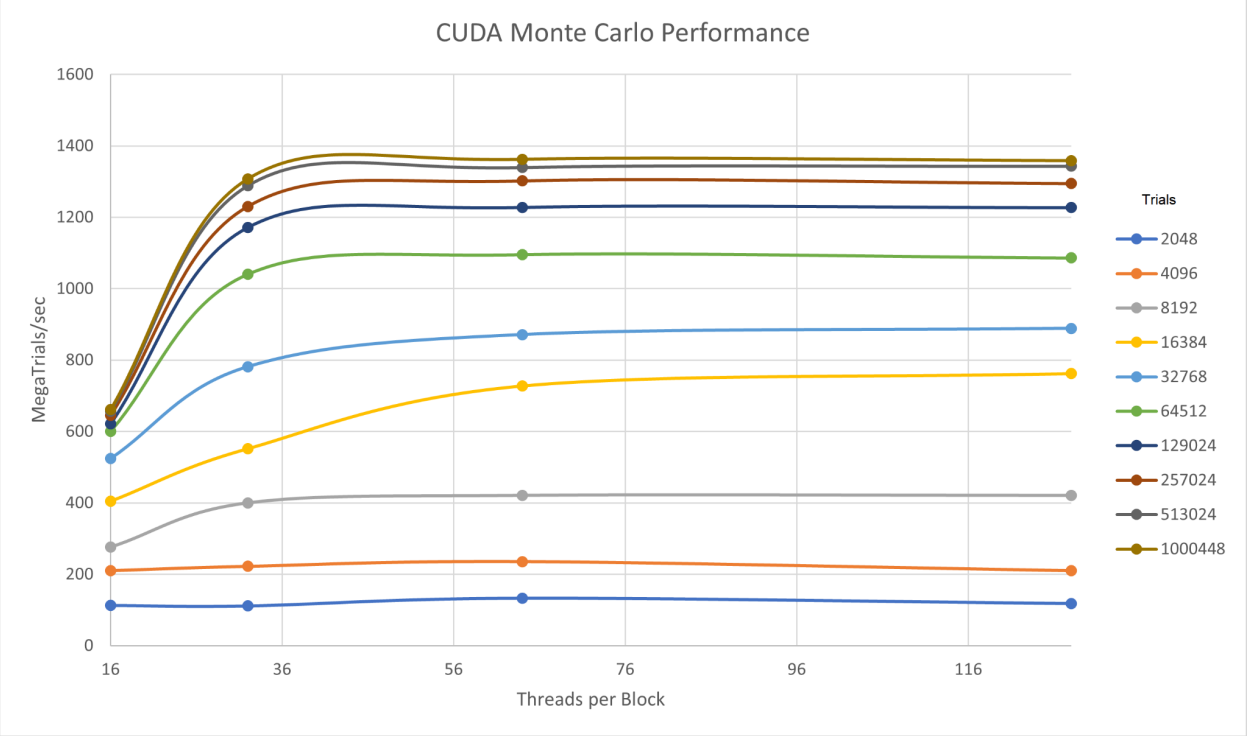
| Blocksize/ Trials | 16 | 32 | 64 | 128 |
|---|---|---|---|---|
| 2048 | 112.48 | 111.11 | 133.33 | 118.08 |
| 4096 | 210.53 | 222.22 | 235.29 | 210.53 |
| 8192 | 275.86 | 400 | 421.05 | 421.05 |
| 16384 | 404.74 | 551.72 | 727.27 | 761.9 |
| 32768 | 524.59 | 781.68 | 871.49 | 888.89 |
| 64512 | 600.36 | 1040.78 | 1095.65 | 1086.21 |
| 129024 | 621.74 | 1171.75 | 1227.77 | 1227.02 |
| 257024 | 645.24 | 1230.39 | 1301.78 | 1294.44 |
| 513024 | 656.97 | 1288.75 | 1339.57 | 1343.16 |
| 1000448 | 662.13 | 1307.9 | 1362.62 | 1359.01 |

Jeremy Udarbe - udarbej@oregonstate.edu
CS 475 - parallel programming
Project 5 - CUDA Monte Carlo Simulation

b. Graph of Performance vs. NUMTRIALS with multiple curves of BLOCKSIZE




c. Graph of Performance vs. BLOCKSIZE with multiple curves of NUMTRIALS



Jeremy Udarbe - udarbej@oregonstate.edu
CS 475 - parallel programming
Project 5 - CUDA Monte Carlo Simulation

3 & 4. Following the curve patterns of most speedups, the CUDA timing method shares a similar distribution as the OpenMP data from project 1 where the performance would logarithmically rise and eventually level towards a horizontal asymptote around 200,000 Trials. In both graphs above, it shows how the speedup will eventually reach a processing threshold limited by the number of cores available to the GPU. In the first graph displaying the performance by block size, the speedup has a positive correlation with the number of subdivisions inside the block's grid. Generally, this is because the process has more threads to execute on the GPU's workload. The only outlier is the blocksize 16 curve which peaks at a significantly lower horizontal asymptote than the other performance curves (explanation in question 5). The second graph shows the performance groups by Trials sharing the same logarithmic distribution pattern which levels around 64 Threads per block. In this data representation, the curve differential increases as the number of trials increases while smaller trial groups tend to share a flatter distribution.

5. Since we are limiting the amount of threads to 16 in one block, we are also limiting the performance due to resource occupancy. Due to how threads of 32 are placed into warps by block, smaller sizes cannot be warped. This restricts thread synchronization in executing and reduces efficiency in sharing data through a low latency shared memory.

6. Compared to the performances in project 1, the curves generally share the same distribution where the performance would peak and stay relatively constant at some number of threads or trials. In the graph with thread curves, the OMP timing had more variation as where the horizontal asymptote peaked per group of number threads whereas the CUDA groups shared a common performance peak for groups greater than 32 threads. The second graph with trial curves basically had the same patterns as more trials are more efficiently parallelizable.

7. In order of maximize efficiency in parallelizing processes in GPU computing and based on how modern GPU workloads are organized, it is most optimized to set multiple threads in 1 block rather than setting multiple blocks of one thread. This is due to the reason that every 32 threads make up a warp which executes simultaneously on different sets of data. This also means that the number of threads must be set as a multiple of 32.