CS 575 Parallel Programming

Project #5

CUDA: Monte Carlo Simulation

Name: Rachel Xing (#934229189)

Email Address: xingru@oregonstate.edu

Description: This project uses CUDA to parallelize a Monte Carlo simulation of hitting the castle on top of a cliff from the cannon controlled by an amateur band of mercenaries. The given ranges for input parameters for the simulation are:

Variable	Meaning	Range
g	Ground distance to the cliff face	20. - 30.
h	Height of the cliff face	10. - 20.
d	Upper deck distance to the castle	10. - 20.
v	Cannonball initial velocity	10. - 30.
θ	Cannon firing angle in degrees	70. - 80.

The simulation itself is written in device code, so each set of trials within a simulation is executed on the GPU for parallel computation. It is run with five different block sizes (i.e., the number of threads-per-block) of 8, 32, 64, 128, 256 combined with the number of trials of 1024, 4096, 16384, 65536, 262144, 1048576, 2097152, 8388608.

Commentary:

1. **Tell what machine you ran this on**

   I ran the program on the **NVIDIA DGX-2 node from the OSU HPC cluster**. This machine is equipped with two 24-core 2.70 GHz Intel Xeon Platinum 8168 CPUs and 16 NVIDIA Tesla V100 GPUs. It also has 1.5 TB of system RAM and 28 TB of NVME. It is running the Linux system, and the compiler I used is nvcc.

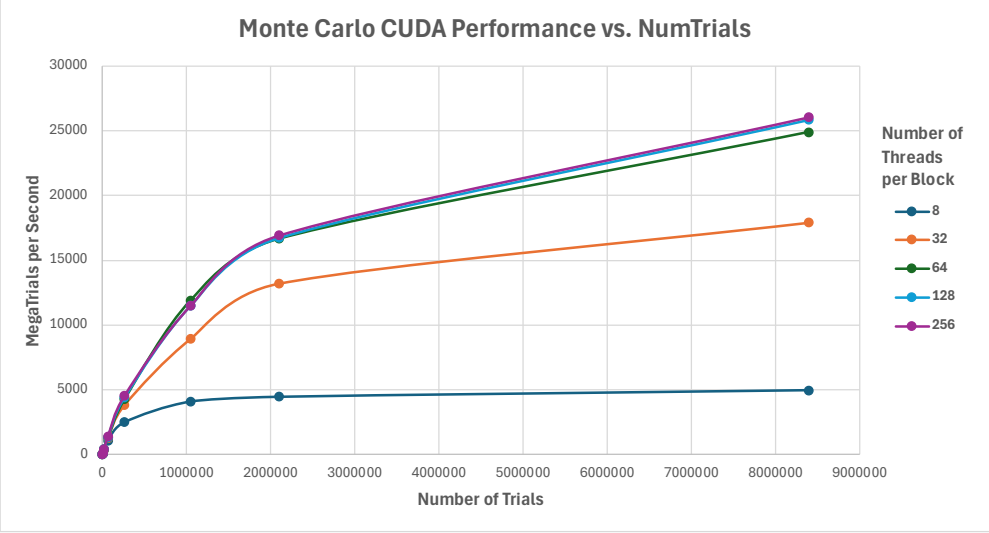
2. **What do you think this new probability is?**

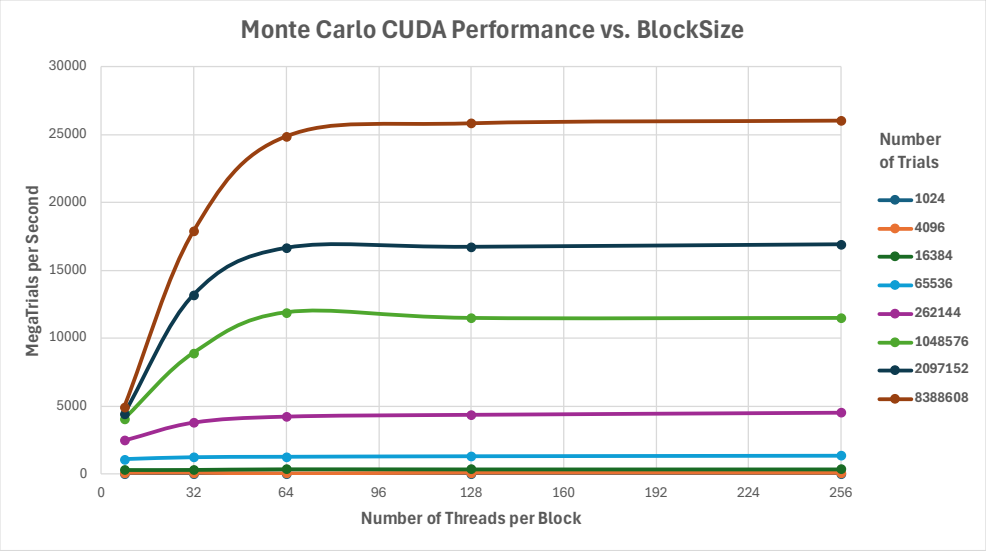
Probability Table (%)								
Trials # / Block Size	1024	4096	16384	65536	262144	1048576	2097152	8388608
8	8.691	8.838	8.557	8.832	8.790	8.730	8.798	8.745
32	8.496	8.740	8.978	8.699	8.819	8.770	8.739	8.768
64	8.789	8.643	9.113	8.827	8.810	8.836	8.738	8.744
128	8.984	8.545	8.472	8.777	8.781	8.697	8.777	8.737
256	8.691	9.766	8.868	8.818	8.791	8.756	8.747	8.768

The simulation results should converge to the actual probability as the number of trials increases, so the new probability should be the average of the five simulations with the largest number of trials (NUMTRIALS = 8,388,608), which is $p = \frac{8.745\%+8.768\%+8.744\%+8.737\%+8.768\%}{5} = 8.752\%$. Thus, the new probability for the Monte Carlo simulation should be **8.752%** (The closest result to it is **8.756%** obtained with **a block size of 256 and a number of trials of 1,048,756**).

3. Show the rectangular table and the two graphs

Performance Table (megatrials/sec)								
Trials # / Block Size	**1024**	**4096**	**16384**	**65536**	**262144**	**1048576**	**2097152**	**8388608**
8	23.81	93.02	326.53	1083.60	2477.92	4072.08	4435.90	4946.58
32	24.39	95.24	333.33	1254.13	3803.16	8928.61	13202.26	17915.80
64	24.39	81.63	**380.95**	1292.11	4246.76	**11902.65**	16650.41	24892.60
128	22.73	**95.24**	372.09	1339.44	4380.75	11485.45	16726.90	25857.56
256	**25.00**	93.02	372.09	**1385.66**	**4535.99**	11485.45	**16912.52**	**26052.87**





4. **What patterns are you seeing in the performance curves?**

   We can observe from the two plots in problem 3 that:

   (1) As the number of trials increases, the overall performance yields a better logarithmic-like growth across all block sizes.
   (2) Differences in performance with different block sizes become more noticeable after the number of trials exceeds 65,536. For smaller numbers of trials, there is no obvious performance gain from parallelization even with larger block sizes.
   (3) With the number of trials > 65,536, increasing the block size from 8 to 64 can lead to a significant improvement in performance. However, beyond a block size of 64, the performance enters a plateau with no more significant performance gain.

5. **Why do you think the patterns look this way?**

   (1) ***As the number of trials increases, the overall performance yields a better logarithmic-like growth across all block sizes***.

   Explanation: This aligns with Gustafson's observation that parallelization can yield larger performance gains as problem size increases. That is because increasing the number of trials (i.e., the problem size) for a fixed block size allows more GPU threads to be scheduled and executed in parallel, so the parallelizable portion of the simulation increases and results in a larger performance gain. However, performance can hit a plateau or even degrade a bit when the problem size becomes large enough to exceed the GPU's capacity.

   (2) Differences in performance with different block sizes become more noticeable after the number of trials exceeds 65,536. For smaller numbers of trials, there is no obvious performance gain from parallelization even with larger block sizes.

   Explanation: That is because with a smaller number of trials (≤ 65,536), the overheads for launching CUDA kernel and transferring data between CPU and GPU can exceed the performance gains from GPU parallel processing.

   (3) With the number of trials > 65,536, increasing the block size from 8 to 64 can lead to a significant improvement in performance. However, beyond a block size of 64, the performance enters a plateau with no more significant performance gain.

   Explanation: That is because once the number of trials is large enough to effectively utilize the GPU cores, the net benefits for GPU processing become positive, and the effect of block size on performance starts to become obvious. Since a warp in CUDA consists of 32 threads, when increasing the block size from 8 to 64, we go from ¼ warp to two full warps per block. This leads to more efficient warp scheduling to hide the latency for memory access, so we can observe a significant performance boost. However, performance enters into plateau when the block size exceeds 64. There can be two

possible reasons: (1) the GPU's computational and scheduling resources per streaming multiprocessor become saturated (2) more warps per block can also reduce the number of blocks that run concurrently. Nevertheless, for large problem sizes (e.g., 2,097,152 and 8,388,608 trials), the highest performance is observed with a block size of 256 (even with no significant improvement after a block size of 64), so increasing warps per block can still provide performance gains when the problem size is large enough.

6. **Why is a BLOCKSIZE of 8 so much worse than the others?**

   As discussed in Problem 5, a warp in CUDA consists of 32 threads. When the block size is only 8, each warp is only partially occupied. There will be fewer active warps available for the GPU to schedule during the execution, and it is more likely for them to stall while waiting for memory access. Thus, a block size of 8 cannot fully achieve the GPU's ability to hide memory latency through warp scheduling. That is why it yields much worse performance compared to larger block sizes that can fill or exceed full warps per block.

7. **How do these performance results compare with what you got in Project #1? Why?**

   The best performance I achieved in Project #1 with OpenMP was **161.27 megatrials per second** with **8 cores and 100,000 trials**, and the performance starts to degrade with a higher number of trials. For the same Monte Carlo simulation, the best performance I obtained in this project with CUDA is **26052.87 megatrials per second** with **a block size of 256 and 8,388,608 trials**. Thus, the GPU implementation can achieve **a speedup of ~162x** compared to the CPU implementation, even with a much larger problem size.

   Such a significant performance gain with the GPU parallelization is because of the different architecture designs for the CPU and the GPU. A modern CPU usually has 8 physical cores, which allows the execution of at most 8 instructions in parallel, while a modern GPU contains thousands of physical cores, which allows the execution of thousands of instructions in parallel. That is why we can observe over 100x speedup in the GPU parallelization compared to the CPU parallelization.

8. **What does this mean for what you can do with GPU parallel computing?**

   This means that GPU parallel computing is ideal for computationally intensive, highly data-parallel tasks like the Monte Carlo simulations. However, since the GPU is inherently designed for tasks with regular data access patterns and uniform control flow, its cores are pure ALUs without complex control logic and enough cache space like a CPU. Thus, for tasks with irregular data structures or needing task-level parallelism, CPU parallelization is preferred over GPU parallelization.

[*Extra Graph*: 3D visualization of CUDA parallel performance with respect to NUMTRIALS and BLOCKSIZE in the Monte Carlo simulation]