Project #5 – CUDA: Monte Carlo Simulation

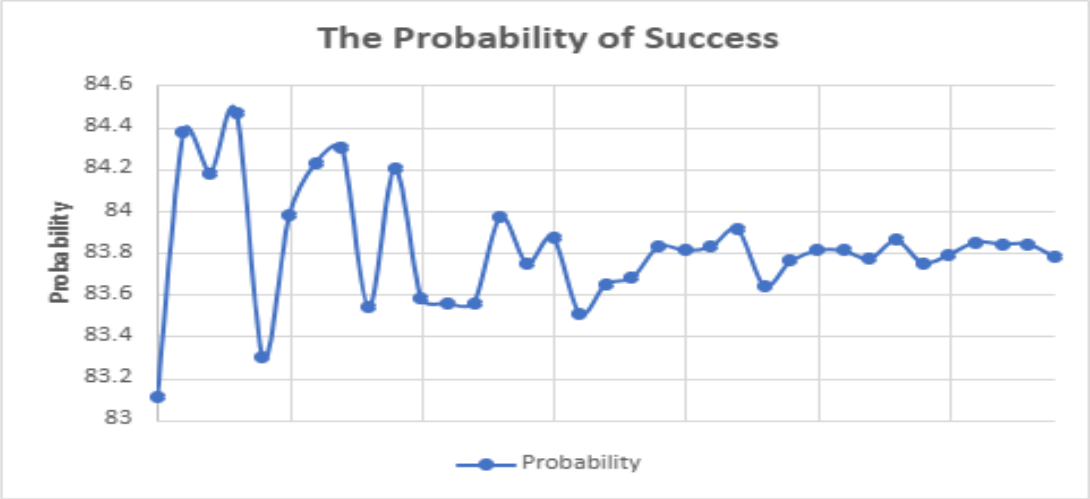
Hyun-Taek Oh
ohhyun@oregonstate.edu
23 May 2024

1. What machine you ran this on?

The program was conducted on a Predator HELIOS 300 (2022).

CPU:	12th Gen Intel(R) Core™ i9-12900H
Motherboard:	Mainboard PH315-55 Intel Ci912900H GN20-E6
Memory:	16GB DDR5 (8GB * 2)
Server:	rabbit.engr.oregonstate.edu, DGX

2. What do you think this new probability is?



< **Figure 1.** The probability of Success in each case >

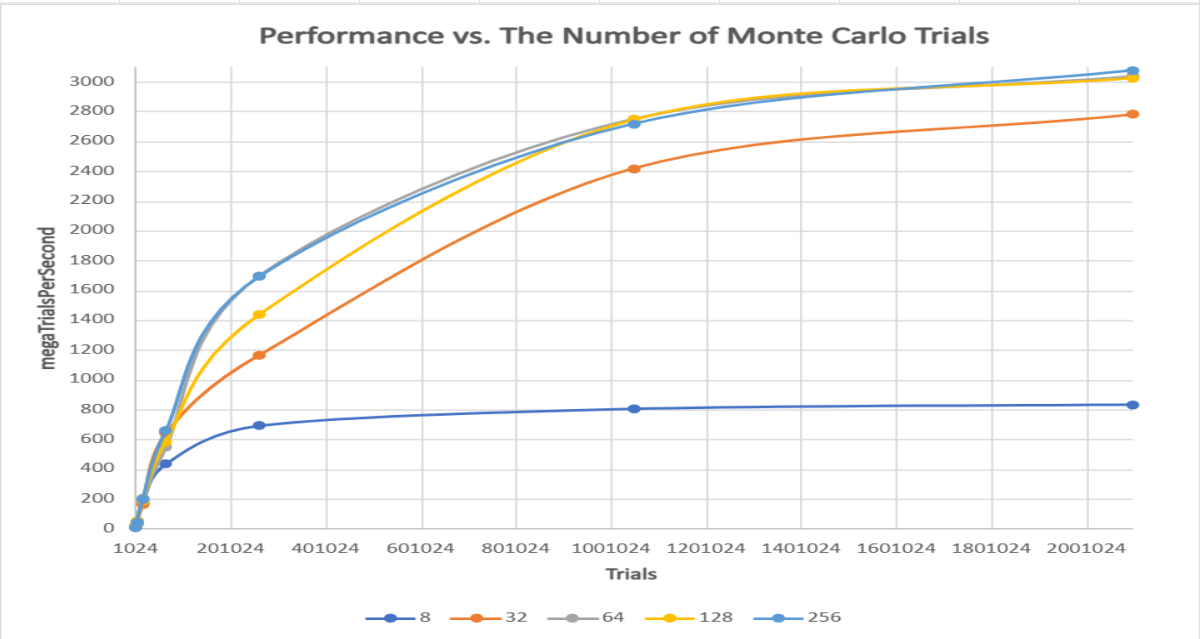
The program resulted new probabilities from 83 to 84. In my opinion, these new probabilities are derived from the increase in the value of Radius since the probability in Project #5 had as same value as one in Project #1 when I tried to change the value of Radius, which is equal to 3.f like Project #1. It means that Radius can affect the new probability directly.

3. Show the rectangular table and the two graphs

NumTrials / BlockSize	1024	4096	16384	65536	262144	1048576	2097152
8	7.2218	47.8863	168.5874	431.3395	689.4462	803.0782	832.2349
32	12.303	42.0638	163.526	637.609	1164.7946	2417.4106	2779.776
64	13.1094	33.1692	199.688	545.5514	1696.7689	2749.916	3035.4794
128	12.1998	52.5883	186.1818	584.1415	1437.6974	2744.8484	3024.6919
256	10.3627	43.8657	192.9891	659.5813	1692.9118	2717.9828	3077.0964

< **Table 1.** The performance table of the correlation between Block size and Trials >

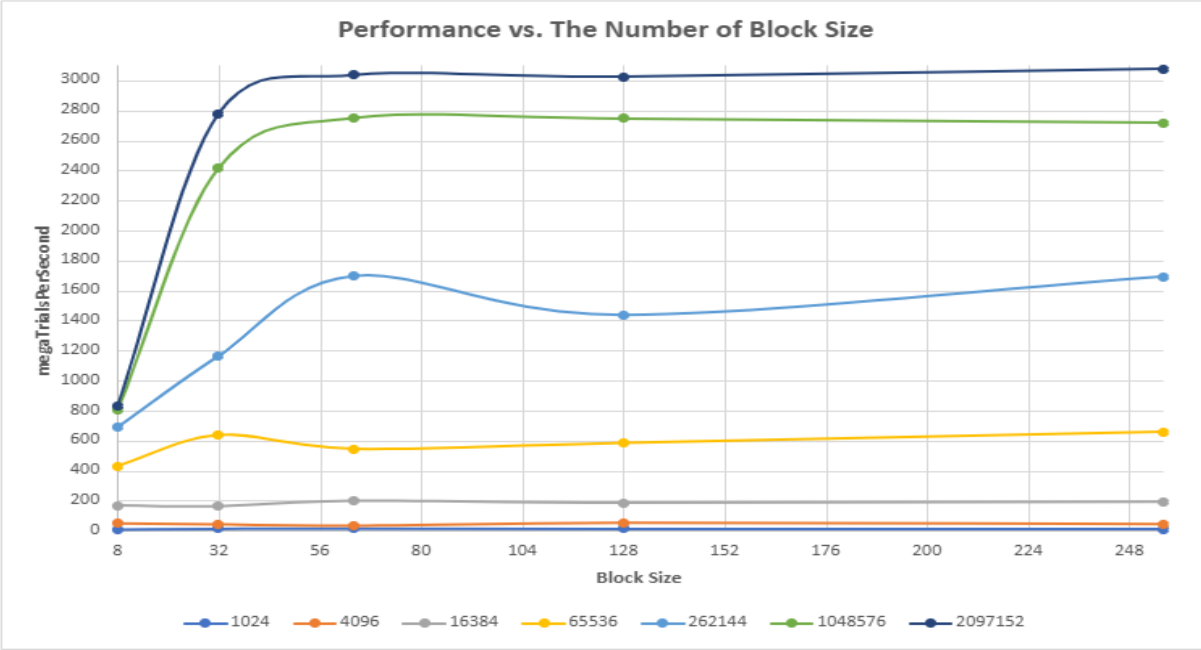
As shown in Figure 2, the largest block size “256” produces best performance in most case even though “128” block size has better performance at one moment when the number of trials is 1048576.



< **Figure 2.** The performance versus the number of trials, with block size >

Figure 3 shows the performance of each block size when the number of trials increases exponentially. The worst performance is resulted by 8 block size. On the

other hand, the best performance is shown by the case of 256 block size. At the largest number of trials, the performance of 64, 128, and 256 block size is quite close to each other.



< **Figure 3.** The performance versus the number of block size, with trials >

As can be seen in Figure 4, there are 5 different block size: 8, 32, 64, 128, and 256. The 8-block size performs worst among other block sizes in almost all cases, showing nearly less than 20 megaTrialsPerSecond. On the other hand, when the number of trials grows up exponentially and is large enough, 256 block size performs best among other block sizes, resulting over 3,000 megaTrialsPerSecond.

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

The Figure 3 and 4 imply that block size is needed to increase for better performance when the program deals with a large amount of data. When a set of data is small, all the block size has similar performance. However, when the amount

of data is large enough, the performance can be affected by the block size significantly. Then, the number of data is larger than 210M, all the performance in the graphs level-off and even decrease.

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

We can guess why the graphs show the different results among different block sizes. It means that the block size plays a significant role in calculating a large set of data. In the Class Notes “CUDA”, each block has its own threads, and these threads share their work efficiently and synchronize their execution through a low latency shared memory.

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

The reason why a BLOCKSIZE of 8 so much worse than the others is that it may not constitute a “Warp”. In the Class Note “CUDA”, it states that every 32 threads constitute a “Warp”. Each thread in a Warp simultaneously executes the same instructions on different pieces of data. This means that 8-block size is too small to consist of a Warp.

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

In the code of Project #5, BLOCKSIZE is defined as the number of threads per block, and NUMBLOCKS is defined as the number of trials divided by BLOCKSIZE. In the Project #1, on the other hand, the different number of threads where the maximum number of threads is 8, are used to calculate the performance. We can compare these results of two different projects because they use different threads

numbers. Project #5 uses more threads and gets much better performance mentioned above. Even though the codes of Project #1 and Project #5 are executed in different environments such as CPU and GPU, we know that the number of threads can affect the performance of calculations.

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

When we need to handle a large number of data set, using GPU parallel computing can show much better performance. Using only CPU cannot perform as like what GPU did because Monte Carlo Simulation project requires much more computing resources from computers to calculate massive data fast and accurately.

9. **[Extra Comment] What are the results using DGX in this program?**

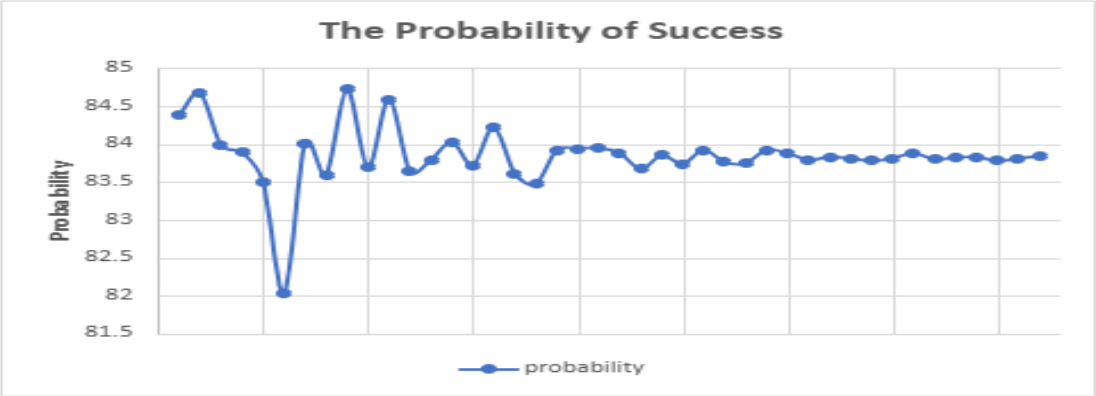
I'm just curious about the performance when using DGX. The results of using DGX are surprised me since the performance of DGX shows much higher than rabbit server.

NumTrials / BlockSize	1024	4096	16384	65536	262144	1048576	2097152
8	18.1715	78.4314	250	981.3129	2420.8038	3835.6551	4271.3941
16	20	68.9655	258.0645	1074.5016	2984.3352	6246.283	7565.0471
32	19.6078	78.4314	320	1153.1531	3831.6184	9338.2732	12369.9513
64	20	70.1754	313.7255	1251.069	3761.2488	11770.1149	17979.698
128	19.2308	72.7273	307.6923	1073.3753	4114.5153	12956.9003	19441.1151
256	20	80	320	1213.2701	4447.3397	12681.1151	19481.5695

< **Table 2.** The performance table of the correlation between Block size and Trials in DGX >

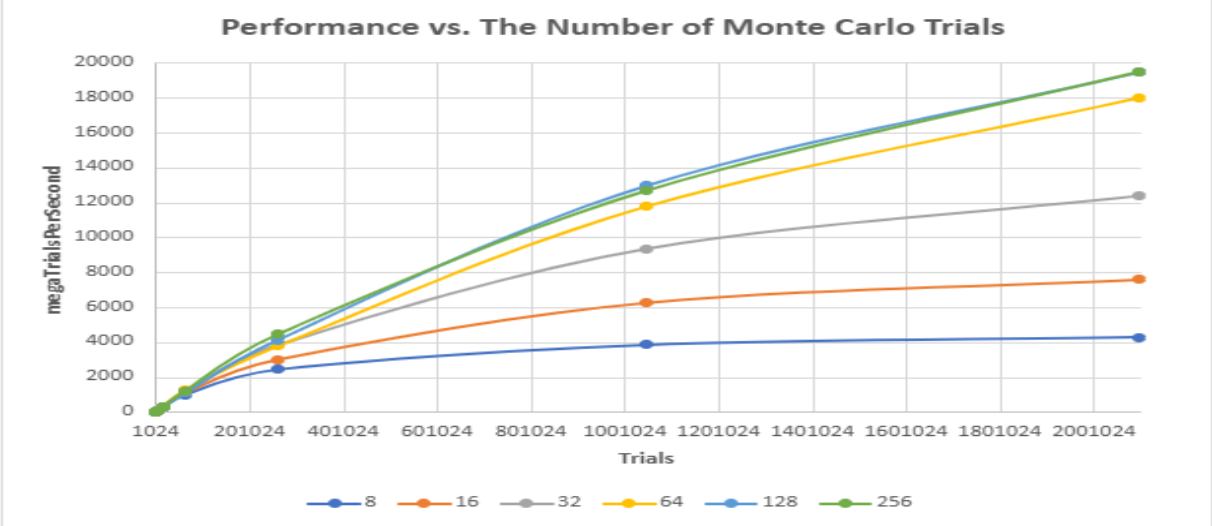
Like rabbit server, as shown in Table 2, the highest performance is 256-block size case when the number of trials is equal to 2097152. The lowest performance is 8-

block size case.



< **Figure 4.** The probability of success in each case using DGX >

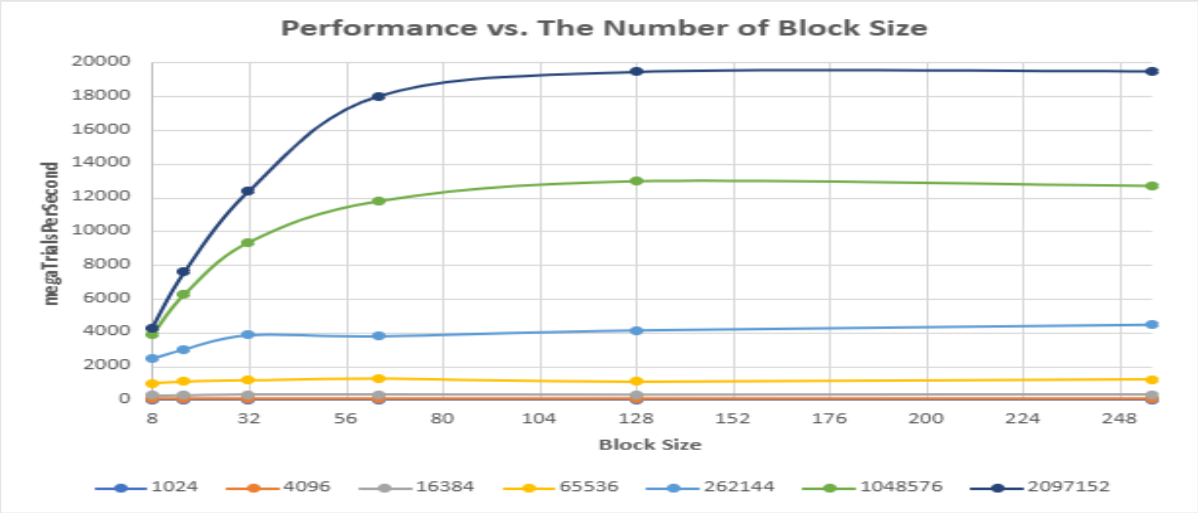
The probability of success in DGX machine is not different from one in rabbit server. This means that the probability itself depends on the random seed in the code and Radius.



< **Figure 5.** The performance versus the number of trials, with block size using DGX >

Figure 5 shows the performance of the relationship between the number of trials and block size in DGX machine. As we can see, the performance in DGX machine is much faster than one in rabbit machine, which is nearly 6-7 times. In the Class Note “DGX”, there is a picture showing two different performance of each

machine, which is DGX-2's Tesla.



< **Figure 6.** The performance versus the number of block size, with trials using DGX >

As can be seen in Figure 6, there are 5 different block size: 8, 32, 64, 128, and 256 displaying their own performance, depending on the number of trials. These graphs have similar patters with those in rabbit server because the only difference is the speed of calculations.