CS475 Project 5
5/25/2023
Aaron Frost
frostaa@oregonstate.edu
933659237

1. Ran on DGX (developed on rabbit)
2. Probability is around 26.85%

3. Here is the .csv / table that I exported from my calculations on DGX:

Trials	Blocksize	MegatrialsPerSecond	Probability
1024	8	32.2581	25.59
1024	32	31.25	24.71
1024	64	31.25	25
1024	128	32.2581	24.9
4096	8	125	26.32
4096	32	125	27.25
4096	64	121.2121	26.1
4096	128	125	26.98
16384	8	444.4444	26.74
16384	32	484.8485	26.57
16384	64	516.129	26.29
16384	128	484.8485	27.32
65536	8	1362.6081	27.09
65536	32	1748.9326	26.91
65536	64	1788.6463	26.78
65536	128	1659.6435	26.71
262144	8	2831.6627	26.84
262144	32	4776.6763	27.13
262144	64	4998.1695	26.72
262144	128	4955.8378	26.7
1048576	8	4124.8741	26.9
1048576	32	9473.2581	26.83
1048576	64	9869.8798	26.85
1048576	128	10144.8919	26.88

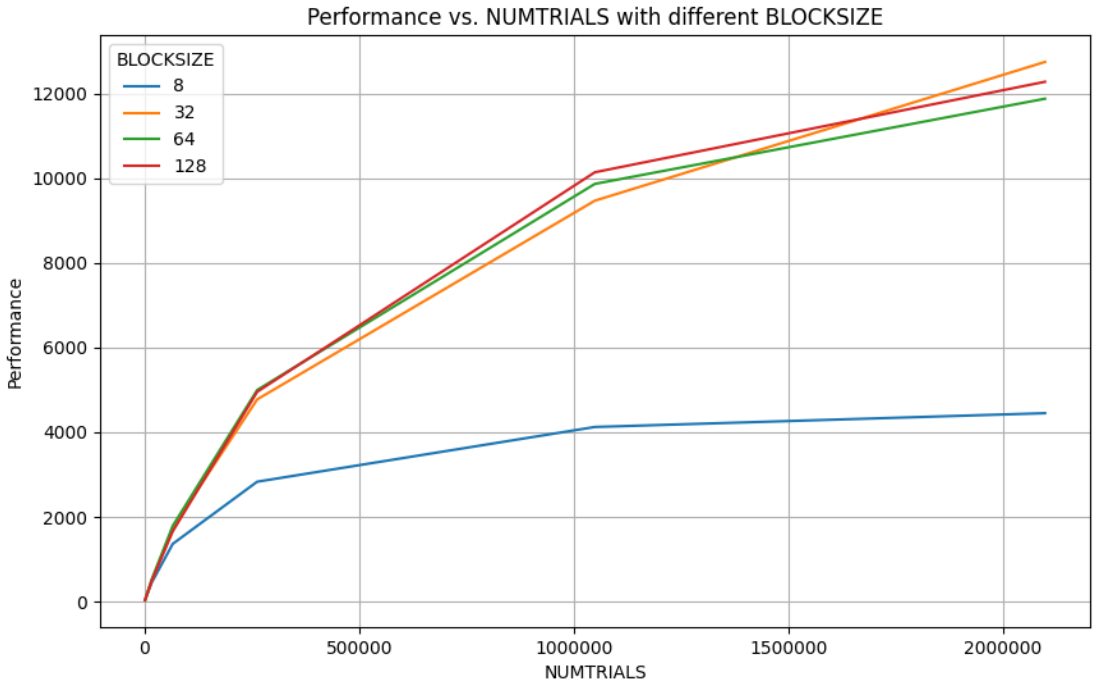
2097152	8	4450.9643	26.96
2097152	32	12752.6762	26.8
2097152	64	11883.2279	26.87
2097152	128	12284.1615	26.89

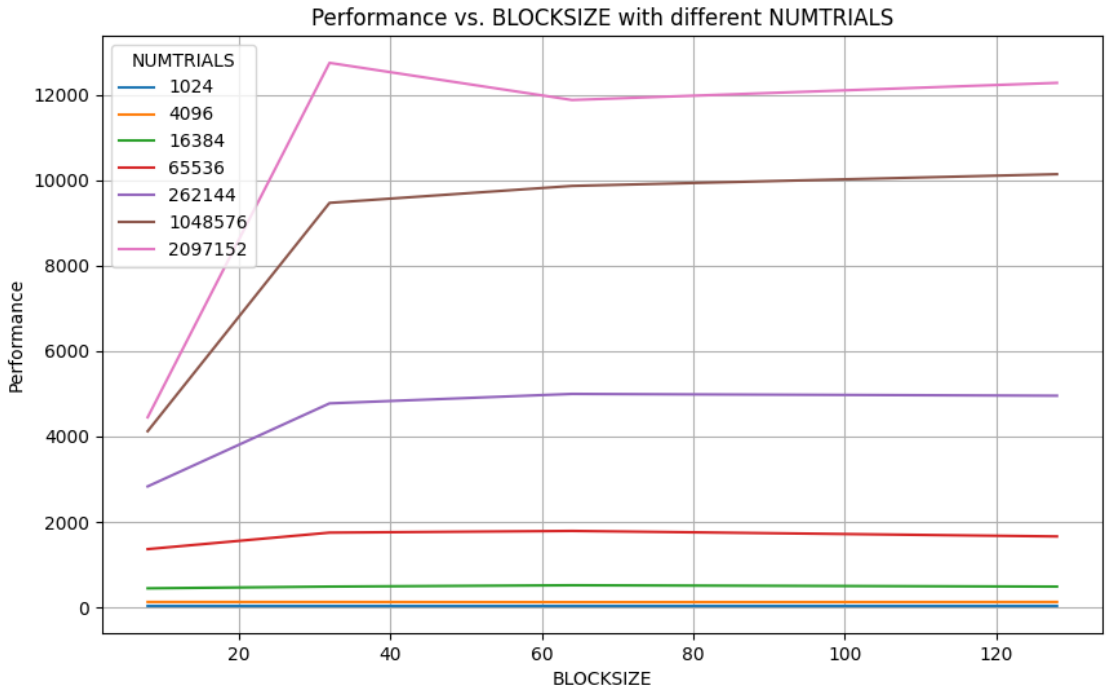
Here is the pivot table I created for the monte carlo performance table:

The rows are blocksize, the columns are the number of trials.

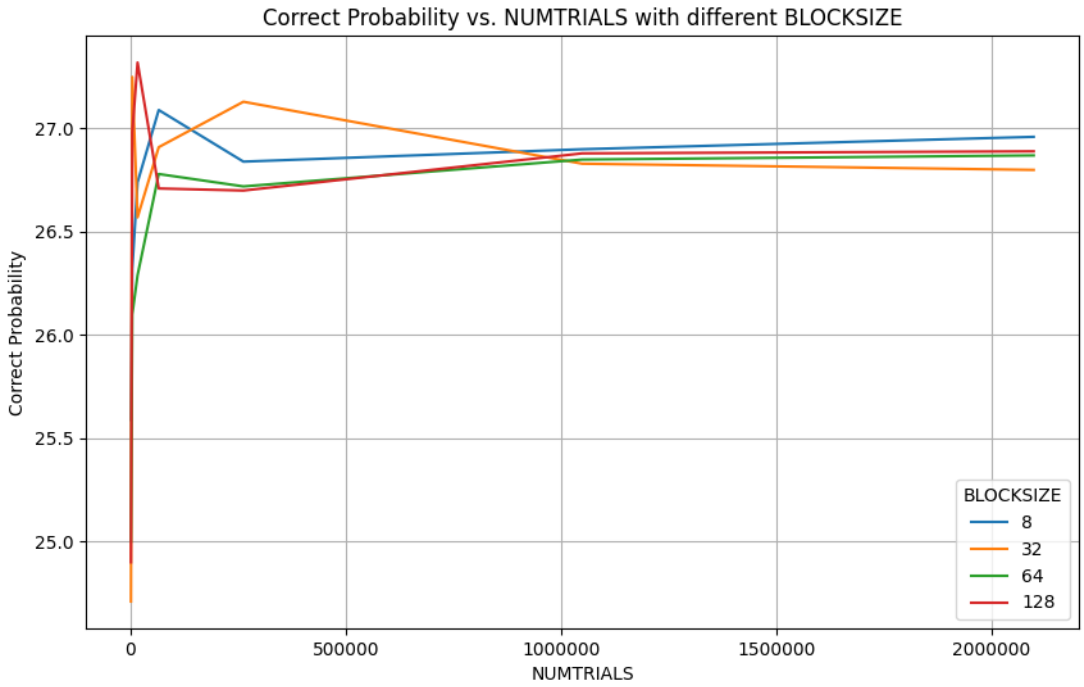
	Trials						
Threads	1024	4096	16384	65536	262144	1048576	2097152
8	32.2581	125	444.4444	1362.6081	2831.6627	4124.8741	4450.9643
32	31.25	125	484.8485	1748.9326	4776.6763	9473.2581	12752.6762
64	31.25	121.2121	516.129	1788.6463	4998.1695	9869.8798	11883.2279
128	32.2581	125	484.8485	1659.6435	4955.8378	10144.8919	12284.1615

Here are the graphs I generated (performance measured in megatrials per second):





I also made this graph to see how the probability relates to trials and block size.



4. From the graphs, it seems that the performance generally increases as both the number of trials and the block size increases. In both graphs, the rate of increase seems to be somewhat logarithmic. Also, increasing the blocksize makes a greater difference in performance as the number of trials increases, up to around a blocksize of 25, the performance gains reduce.

5. I think that the patterns look this way because the Monte Carlo simulation benefits from parallelization. It can be split into independent tasks such that more tasks, and larger blocks (more data processed in parallel) allow a better utilization of the GPU's compute power. The diminishing returns of increasing the block size and num trials likely occurs because the GPU begins to work near capacity and will not improve given different block size or numtrials given the workload. I think that the performance gains reduce after a block size of around 25 because the GPU is not fully utilized until each multiprocessor reaches it's capacity in terms of the number of threads each can handle.

6. Notice that the block sizes increase from 8 to 32. The block size of 8 is much worse than 32 and the other even larger block sizes because it underutilized teh GPU cores. Other potential reasons include that the GPU does not benefit from latency hiding for memory access with such a low block size, and also that the smaller block size can lead to fewer opportunities for memory coalescing (combining accesses to memory into a single transaction), which is known to improve performance.

7. Compared to my results from Project 1 where we compared the performance to the number of trials using multithreading, the performance using GPU parallel computing grows with the number of trials in all contexts (block size in this Project, threads in Project 1). Using multithreading, there is little to no performance gain when you increase the number of trials with a fixed number of threads. However, with GPU parallel computing, there is a significant performance gain when you increase the number of trials with a fixed block size. Also, increasing the block size seems to only increase performance, whereas in Project 1, increasing the number of threads can actually significantly decrease performance according to the data I collected.

8. This means that with GPU parallel computing, you greatly benefit from data-parallel tasks by executing the same operation on many, many, data points. Things like rendering or processing images (grids of colors) and deep learning are great candidates for GPU parallel computing. Multithreading is great for CPU intensive tasks which can be broken down, but there is a limited number of cores available (usually less than 100) to most computers which limits the amount of simultaneous execution you can perform.