Project 5: Cuda Monte Carlo Simulation

By Caleb Knight

knightca@oregonstate.edu

Introduction

This is a technical write-up for observing the Monte Carlo Simulation using Cuda. I ran these tests on my mid 2012 MacBook Pro. It has 16gb of RAM, with the 2.3 GHz Quad-Core Intel Core i7 processor.

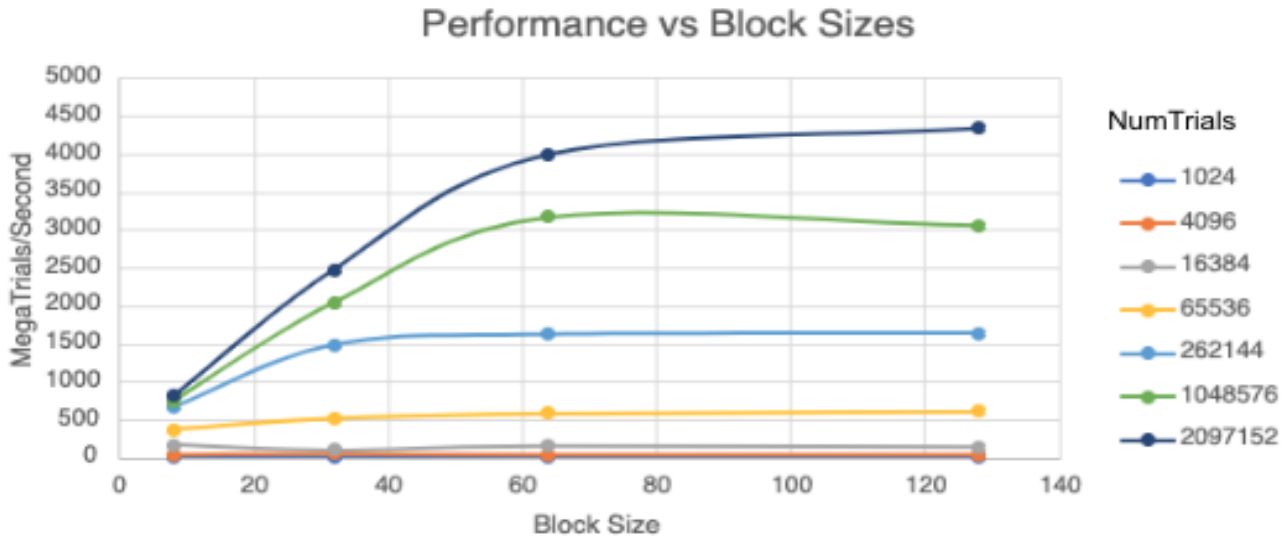
Table

Table of Block Size (Left) and Number of Trials (Top) with MegaTrials/Second as Values

	1024	4096	16384	65536	262144	1048576	2097152
8	13.0612	40.5192	170.1562	373.1098	666.1788	747.5134	807.8896
32	8.3138	49.6702	97.8032	514.1853	1488.9131	2040.9842	2477.3569
64	9.5238	38.016	155.2456	577.8781	1623.4642	3164.4615	3989.5296
128	9.5722	36.4361	142.9369	604.6649	1640.3685	3052.4454	4333.8182

Graphs


Performance vs Trials
MegaTrials/Second
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
500000
1000000
1500000
2000000
2500000
NumTrials
Block Size
8
32
64
128


Performance vs Block Sizes
MegaTrials/Second
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100
120
140
Block Size
NumTrials
1024
4096
16384
65536
262144
1048576
2097152

Observations

As we can see from the graphs above, for the most part, as the block size and the number of trials increased, we see a fairly even increase in performance. In our top graph, Performance vs Trials, we see that all curves follow a pretty similar pattern. Up until 262k trials, they all seem to increase. At 262k trials, we see the curves change a bit. Smaller block sizes like 8 and 32 slowly continue to climb but Block sizes 64 and 128 each continue to climb pretty intensely. Block size 128 is pretty linear and even while 64 has a bit more of a curve to it and seems that it will flatten out if more data points are used.

In the bottom graph, Performance vs Block Size, I see a couple of patterns. First, I notice that trials under 262k all show low performance and are fairly linear over all block sizes. Once we get to 262k and up, we can see that the best performance increase is up to 64 Block Size, and then at that point, our data begins to plateau and even begin to subside. Now I am not 100% sure on this, but I think the reason the performance is so effective up to 64 is because the max dimensions of a block in Cuda is 64…? If I remember correctly, 1024x1024x64 is the largest dimension of an optimal block. If this is true, this would make sense since we see on both graphs how well Block Size 64 does.

As we see on both graphs, we can see that there are certain data points that show the most growth, but are not necessarily the highest performance. For example, using block size 64 or 128, seem like they will continue to increase performance the larger the number of trials. But, when using block size 8, we can see that it has peaked and essentially plateaued by 2mil number of trials and likely, if continued, would decrease in performance. The same goes for using a specific number of trials. We see that the optimal ability of trials less than 262k is pretty awful. Where as above 262k trials, the performance growth from 32 blocks to 64 is the most optimal.

The reason a block size of 8 is the worst is because it is the only block size we have that is less than 32 which is the size of a warp. Therefore, I believe that this would lead to part of the warp being idle and decreasing performance/efficiency.

Comparing this to project 1, I see different graphs, and smaller numbers, but still similar patterns. We can see that in each case, there was optimal block sizes/number of threads as well as number of trials. I believe that the reason we see a steadier performance in this project is because Cuda uses block sizes. In project 1, we were using a large number of threads, but all of those threads were not working within blocks like we have here. Using Cuda, we can group a large amount of threads to all work towards the same goal at the same time. Thus, giving us a much more linear data set compared to a bunch of threads all trying to complete tasks at the same time but getting blocked or killed in the process.

Since threads can sync up within blocks, what if we could then run the blocks parallelly? Then, we could have all of these threads working together, already synced to be the most efficient, and on top of that, we can have two or four or more of these blocks then running in parallel to double, quadruple, etc that performance. I feel like this is where we could see some exponential growth in our performance and efficiency when working with the GPU.