Oregon State University

CS 475 - Spring 2021

Professor

Mike Bailey

mjb@cs.oregonstate.edu

Project #5

CUDA Monte Carlo

by

Abdullah Saydemir

saydemia@oregonstate.edu

May 16, 2021

Setup:

This experiment is run on *DGX* servers. I don't think we need an uptime command output since *DGX* allocates resources dedicated to me only during the run.

NUMTRIALS = [1024, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576]

BLOCKSIZE = [8, 16, 32, 64, 128]

GigaTrials / Second = How many billions of trials are executed in a second.

Results:

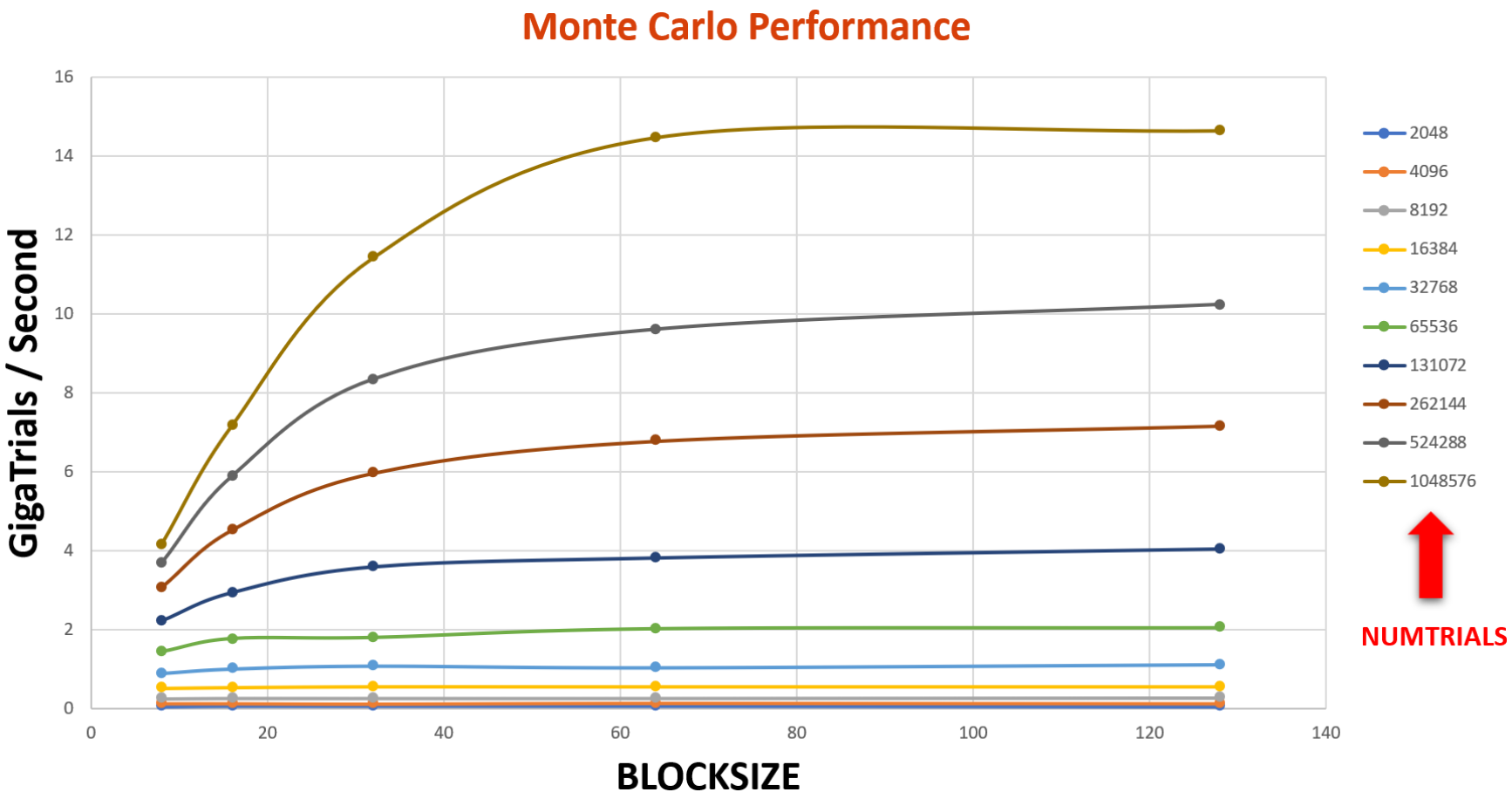
I estimated that the probability these "doofuses" will actually hit the castle is 10.04%. I got this result when I ran the code with NUMTRIALS = 1M. Here is the screenshot.

```
saydemr@DESKTOP-SJCFH0H:/mnt/c/Users/abdul/desktop/w/oregon/cs475/hw/hw5$ sh max.sh
NUMTRIALS = 10485760 BLOCKSIZE = 128 PERFORMANCE = 1.6299 PROBABILITY = 0.1003948212
```

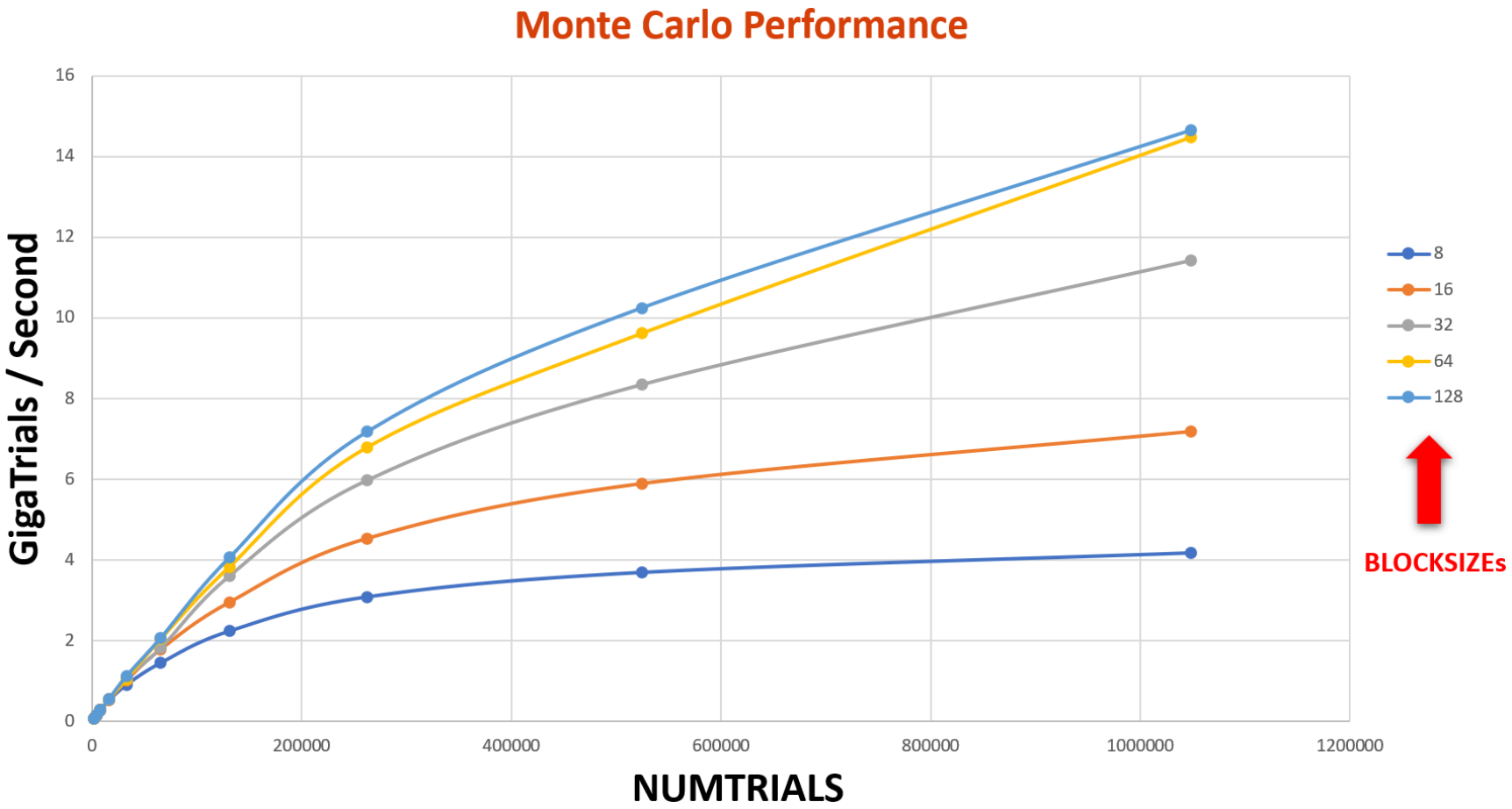
I ran the above experiment on my *WSL - Ubuntu 20.04* just to try if CUDA actually works on *WSL*. It was very smooth!

Performance table of the experiment is as follows. Performance is measured by *GigaTrials / Second* and data are collected on *DGX*.

		NUMTRIALS									
		2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576
BLOCKSIZE	8	0.069	0.143	0.267	0.516	0.897	1.444	2.233	3.080	3.687	4.171
	16	0.071	0.143	0.276	0.533	1.008	1.786	2.951	4.533	5.894	7.186
	32	0.071	0.138	0.276	0.552	1.080	1.812	3.602	5.975	8.351	11.433
	64	0.071	0.148	0.276	0.552	1.040	2.038	3.828	6.787	9.615	14.480
	128	0.069	0.143	0.285	0.552	1.115	2.062	4.055	7.173	10.246	14.655



First thing is the noticeable performance loss for small BLOCKSIZEs. For each NUMTRIAL curve, the performance drastically increases when the BLOCKSIZE increases. Because, in the GPU each warp has 32 threads and if we use 8 or 16 threads, we are not utilizing all of the allocated resources. For BLOCKSIZE of 64 and 128 the performance is pretty much the same since we are utilizing the allocated resources fully. Also, it is important to note that if the data size is too small, there is no reason throwing more threads to it. That's why for smaller data sizes the performance curves stay flat and never increase.



This graph shows that increasing the BLOCKSIZE generally contributes to performance for larger data sizes and does not really affect the performance for the smaller ones. Biggest leap occurs when the block size increases from 16 to 32 because we start to use a full warp. BLOCKSIZE of 64 is better than 32 because having two warps ensures that we have another warp waiting if a warp swaps out. After block size of 64 the increase in performance is really small (compared to other gaps) that we can ignore.

5) *Why is a BLOCKSIZE of 16 so much worse than the others?*

A warp in GPU consists of 32 threads. If we put 16 threads per block, the GPU does not merge two blocks in one. They will be executed either in different blocks or in different time slots. Since we are not utilizing some of the threads, efficiency of the program decreases.

6) *How do these performance results compare with what you got in Project #1? Why?*
Compare?! No way! I just checked my result from *Project #1* and the best performance value I got was *30 MegaTrials / Second*. In this experiment the worst performance value I got is *69 MegaTrials / Second*. GPU rocks!

GPU is specialized to be used on isolated structured computations and if you use it in a proper job then you get the best results. It takes some data and applies the instruction. There is no trick. Get the data, apply the instruction in parallel, put it back. On the other hand, the CPU should handle branching and function calls which is not very suitable for parallel computing most of the time. Though we can do some parallel work with CPU, it is limited. That's why we see a huge difference.

CPU => all-around, can do anything, slow for parallel computing
GPU => specialized, it has one job, extremely fast in its job

7) *What does this mean for the proper use of GPU parallel computing?*

This means if your program is highly parallelizable and you are a good programmer, you can save a lot of time by just using a GPU. Considering that today we mostly use big data, which means our programs tend to be highly parallelizable, proper use of GPU parallelism is a must.

Remember, GPU cannot do jobs that are not parallelizable. For example, if you have a dependency between data, you should handle dependent parts with a CPU. *(logic is also not well suited for the GPU)*