Oregon State University

CS 475 - Spring 2021

Prof. Mike Bailey
mjb@cs.oregonstate.edu

Project #1

OpenMP: Monte Carlo Simulation

by
Abdullah Saydemir
saydemia@oregonstate.edu
April 14, 2021

Setup:

This experiment is run on Windows 10 laptop with the following specifications.

- CPU: Intel i5-7300HQ @ 2.50GHz 4 cores
- RAM: 8 GB with 2400 MHz
- GPU: NVIDIA GeForce GTX 1050

I closed all the applications while running the script. However, the graph was not in a pretty good shape since Windows was still running background tasks. I also disabled some of the services but I don't think that it helped much. So, I set NUMTRIES to 30 hoping that some of the passes run when the background tasks are having a break.

NUMT = [1, 2, 3, 4] *// I don't know why I included 3*
NUMTRIALS = [1, 10, 100, 1000, 10000, 100000, 500000]

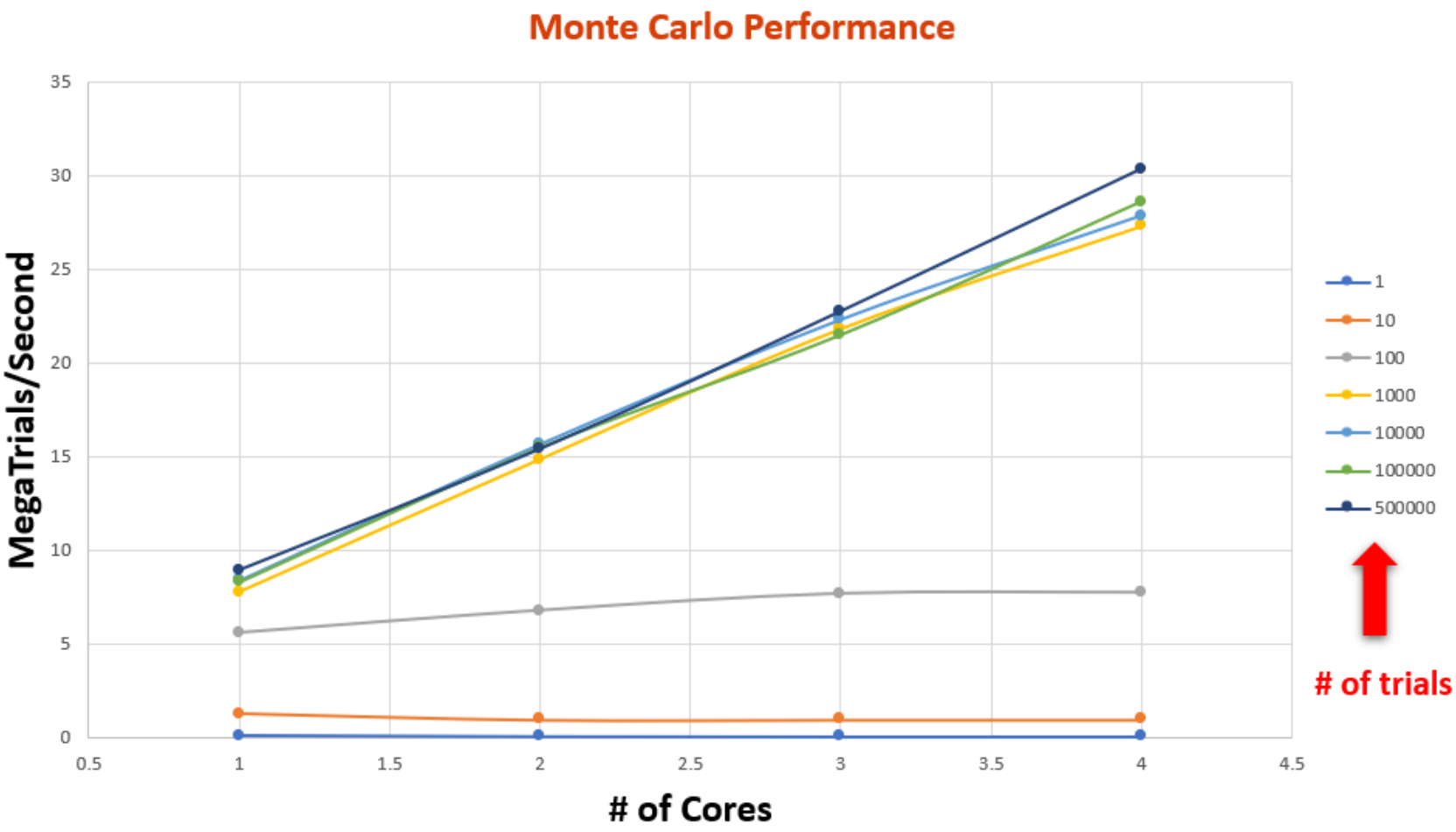
Results:

I estimated that the probability these "doofuses" will actually hit the castle is 6.57%. I got this result when I run the code with NUMTRIALS = 1 billion. Here is the screenshot.

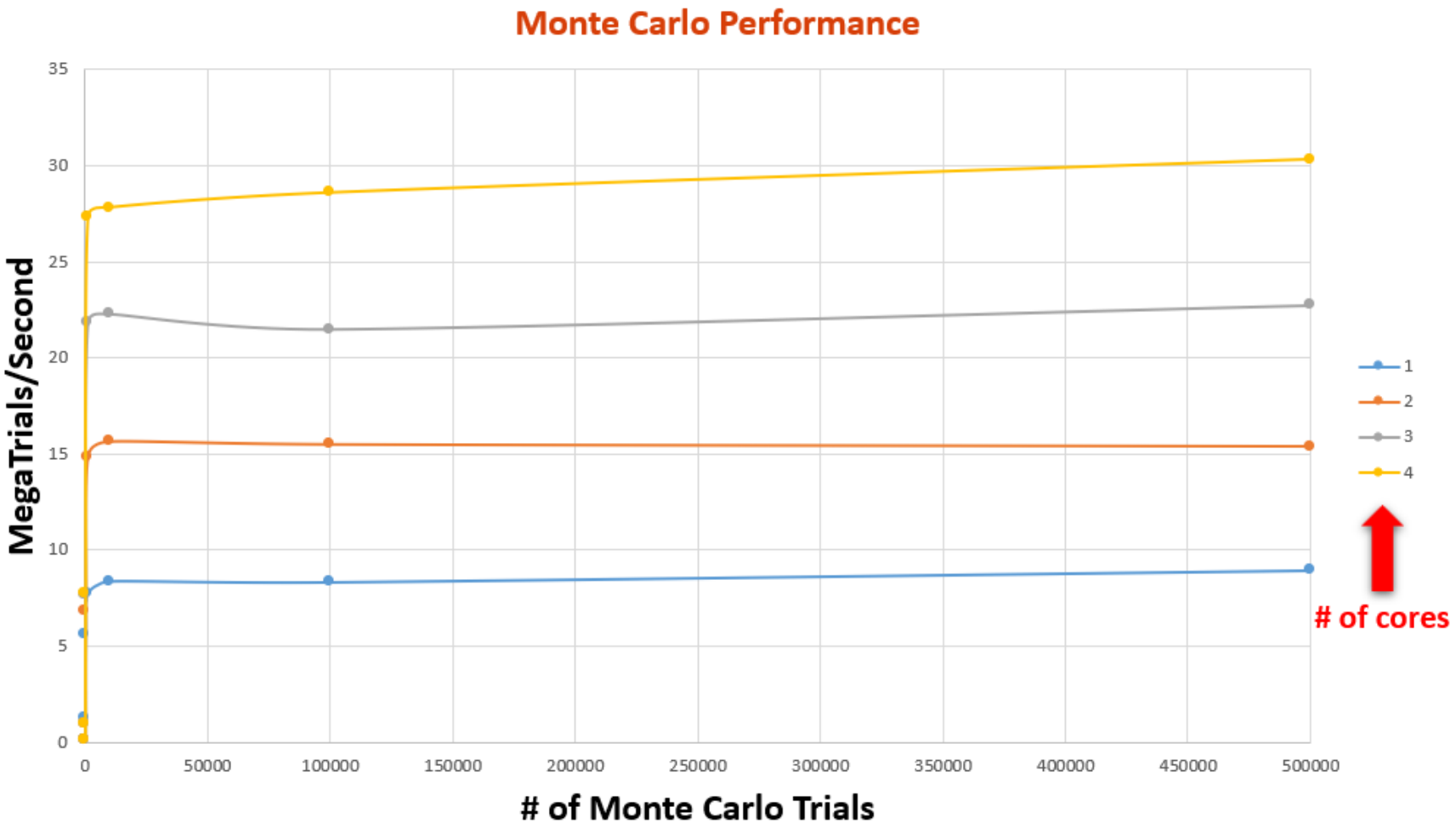
```
PS C:\Users\abdul\Desktop> .\project.ps1
 4 threads : 1000000000 trials ; probability =   6.57% ; megatrials/sec =  10.93
```

The table came out from this experiment as follows.

	# of Trials						
# of Cores	1	10	100	1000	10000	100000	500000
1	0.14	1.28	5.62	7.77	8.39	8.34	8.75
2	0.10	0.98	6.80	14.86	15.65	15.50	15.40
3	0.09	0.98	7.69	21.83	22.29	21.50	22.74
4	0.09	0.98	7.75	27.32	27.83	28.60	30.33



Trials less than 1000 do not work very well for parallel structure. Because the size is very small, we cannot gain performance by simply dividing the work among the cores. But after 1000, the graph shows that parallelism is worth doing.



I expected that the graph to be non-decreasing since the maximum number of trials is a reasonable amount. However, for 3 cores, the performance decreases as I increase the number of trials from 10,000 to 100,000. In fact, the line for 3 cores should have been smooth, since in low CPU allocation computers use only one core. On the other hand, for 4 cores, the performance increases in the same interval. I think since I cannot fully free the first core from tasks, It might have decreased the performance for small numbers.

Parallel Fraction *(F_P)* **and Speedup** *(S)*

I used the performance values of 500,000 trials in my calculations.

$$S = Performance\ with\ n\ threads\ /\ Performance\ with\ one\ thread$$
$$Fp = (\ n\ /\ (n-1)\ )\ x\ (1\ -\ (\ 1\ /\ S\ )\ )$$

- 1 to 2 cores : $S = 15.40\ /\ 8.75 = 1.7600$
  $Fp = (\ 2\ /\ (2-1)\ )\ x\ (1\ -\ (\ 1\ /\ 1.7600\ )\ ) = 0.8636$
- 1 to 3 cores : $S = 22.74\ /\ 8.75 = 2.5988$
  $Fp = (\ 3\ /\ (3-1)\ )\ x\ (1\ -\ (\ 1\ /\ 2.5988\ )\ ) = 0.9228$
- 1 to 4 cores : $S = 30.33\ /\ 8.75 = 3.4662$
  $Fp = (\ 4\ /\ (4-1)\ )\ x\ (1\ -\ (\ 1\ /\ 3.4662\ )\ ) = 0.9486$

Then we can calculate F_P by taking the average of resulting values.

$$Fp = (0.8636 + 0.9228 + 0.9486)\ /\ 3$$
$$= 0.9117$$

I believe 0.91 is an acceptable measure for a highly parallelizable code. The code is really good to run in multiple cores but setting the for loop, scheduling the threads and passing an iteration to a core etc. eats the parallel fraction. I was expecting F_P to be bigger than 0.95 but It seems like my computer (and/or OS) is not really good at handling that.