Intro to Parallel Programming

Project n°1 – OpenMP: Monte Carlo Simulation

Amine Gaizi
gaizia@oregonstate.edu
Student ID: 934-044-900

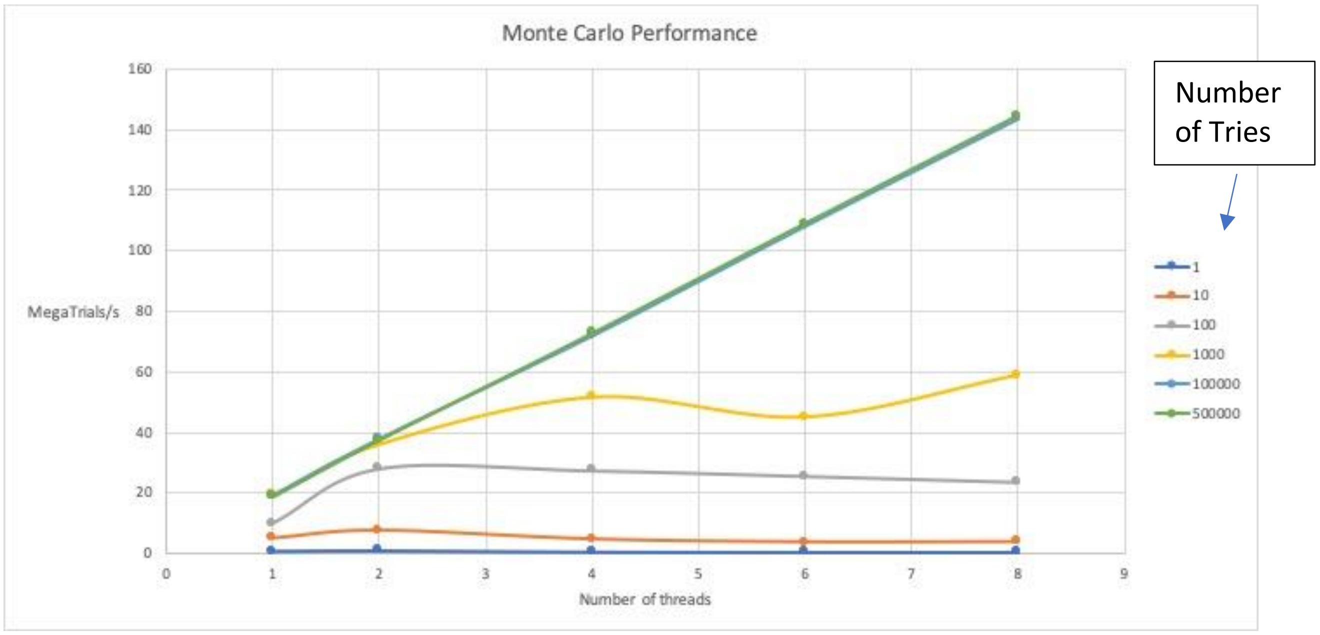
Which machine was this run on ?

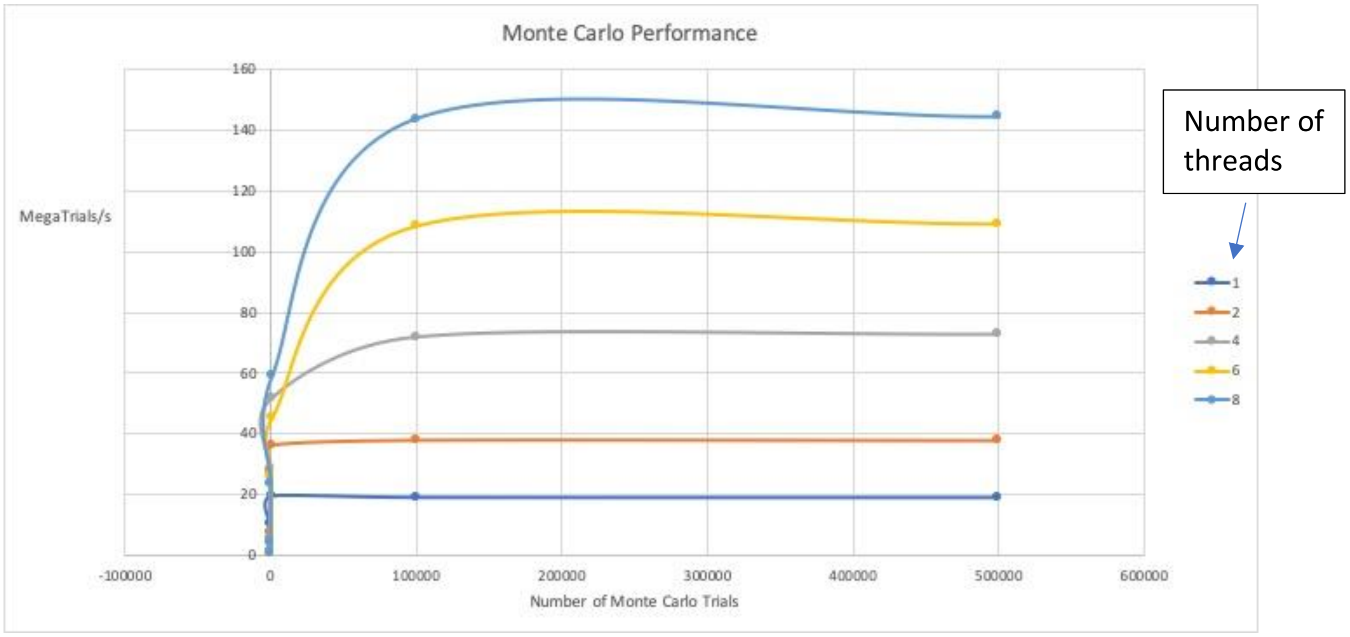
This project was run on the university's flip server (using a ssh connection). The load average at that moment was: 5,55 6,41 6,19. I could not find a way to run the simulation during a moment (even late at night) when the load was lower since everyone is using the CoE servers. Although the load is high, the results stay coherent.

What was the performance for the different simulations ?
Trials refers to the number of Monte Carlo simulations.

Trials/Threads	1	10	100	1000	100000	500000
1	0,889	4,987	10,007	19,406	19,039	18,992
2	1,036	7,49	28,106	36,089	37,718	37,63
4	0,602	4,659	27,403	51,514	71,847	71,789
6	0,495	3,738	25,562	45,073	108,179	108,9
8	0,473	3,803	23,579	58,819	143,744	144,375

These simulations were run with a number of tries of 4096.





Interpretation of the results

From these results seems very coherent with the concepts taught in class. Indeed, when the number of tries is too low, we can clearly see a drop in performance when the number of threads is increasing. For instance, from 1 to 100 tries of the Monte Carlo simulation, the performance is better with 2 threads than with more threads. As a matter of fact, for 10 trials, 2 threads are almost twice as fast as 8 threads.
However, the more the number of trials increases, the more the parallelism's efficiency takes effect until it becomes stable for over 100.000 trials. The difference in performance for 500.000 to 100.000 is minimal, and definitely not worth the computation power necessary.

What is the estimation of the probability ?

The run with 500000 trials using 8 threads gave a probability of: $P = 0{,}131486$
Two other simulations under the same conditions gave the probabilities: $P_1 = 0{,}131178$ and $P_2 = 0{,}131296$.
The average of the three values is $P = 0{,}131217$.

What is the Parallel Fraction value ?

The results from the simulations give us the 1-thread-to-8 Speedup result:

$$S = \left(\frac{Performance\ 8\ threads}{Performance\ 1\ thread} *\right) = \frac{144.375}{18.992} = 7.601 \quad \text{*for 500000 trials.}$$

Therefore, following the formula seen in class:

$$F_{parallel} = \frac{n}{n-1} * \left(1 - \frac{1}{Speedup}\right) = \frac{8}{8-1} * \left(1 - \frac{1}{7{,}601}\right) = 0{,}9925$$

This means that 99,25% of the program was run in parallel. In this case, although the parallel fraction is very high, it does not reach 100%. One of the reasons that is, is because of the part of the program that takes care of initializing the arrays.

If we pay attention to the values of the performance with less trials, it is clear that Gustafson's observation is respected. For instance, if we calculate the parallel fraction for 8 threads and 1000 trials (when the parallelism with 8 threads is starting to become beneficial), we find:

$$S = \left(\frac{Performance\ 8\ threads}{Performance\ 1\ thread} *\right) = \frac{58{,}819}{19{,}406} = 3.03$$

$F_{parallel} = \frac{n}{n-1} * \left(1 - \frac{1}{Speedup}\right) = \frac{8}{8-1} * \left(1 - \frac{1}{3{,}03}\right) = 0{,}7656$ which is clearly inferior and less efficient than the simulation with 500.000 trials.