Student Name: - ATHARVA DESHPANDE

Student Email: - deshpana@oregonstate.edu

Project No: - Project#1

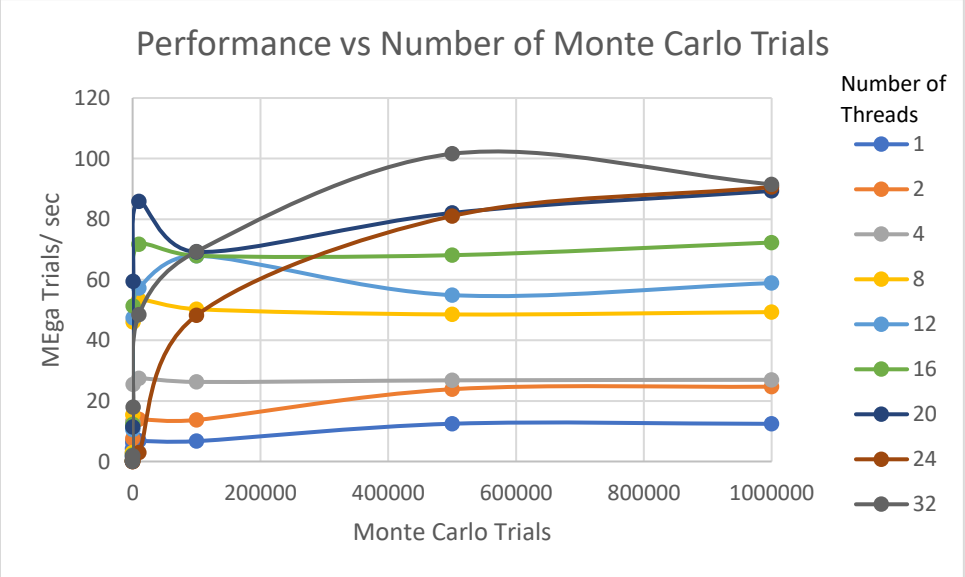
Project Name: - OpenMP: Monte Carlo Simulation

ATHARVA DESHPANDE **CS575** deshpana@oregonstate.edu

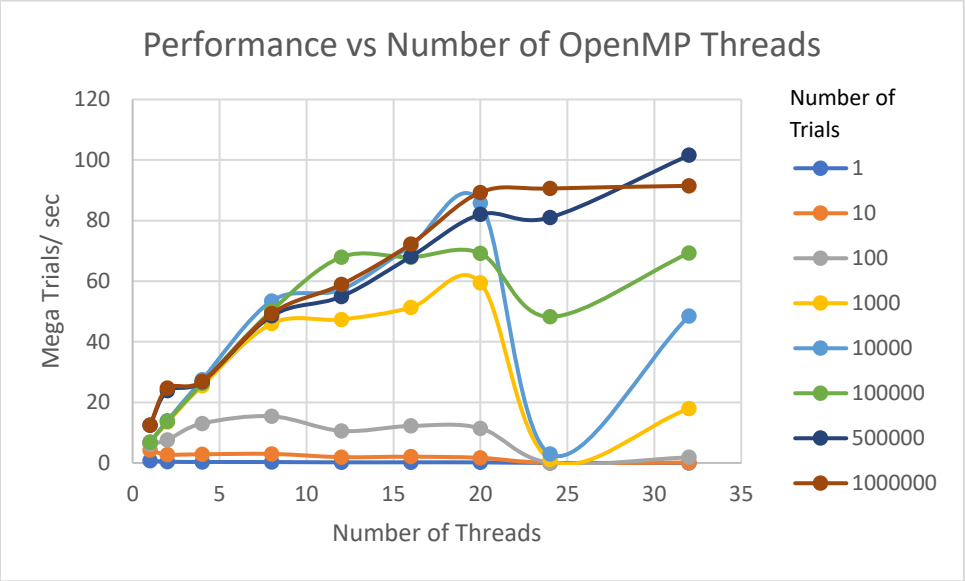
1. Data table of the performance numbers as a function of threads and NUMTRIALS.

No. of Threads/ Mega Trials per sec	1	10	100	1000	10000	100000	500000	1000000
1	0.75	4.19	6.21	6.88	6.83	6.71	12.47	12.43
2	0.43	2.79	7.64	13.49	13.96	13.71	23.86	24.7
4	0.32	2.87	13.03	25.48	27.49	26.26	26.79	26.97
8	0.3	2.98	15.41	46.02	53.49	50.29	48.55	49.32
12	0.2	1.92	10.58	47.38	57.24	67.93	54.93	58.9
16	0.21	2.06	12.21	51.29	71.71	67.93	68.11	72.29
20	0.19	1.68	11.45	59.46	85.83	69.18	82.05	89.32
24	0	0	0.03	1.16	2.95	48.24	81.04	90.61
32	0.02	0.12	1.85	17.97	48.52	69.32	101.58	91.49

2. Two performance graphs-
   1) Performance versus the number of Monte Carlo trials, with the colored lines being the number of OpenMP threads.



2) Performance versus the number OpenMP threads, with the colored lines being the number of Monte Carlo trials.



3. Your estimate of the Probability.
   ➔ 26.87

4. Your estimate of the Parallel Fraction (show your work!).
   ➔ The estimate of parallel Fraction (average) was calculated to be **0.88**.

Threads	Speedup	Fp
2	1.987128	0.993522
4	2.169751	0.718823
8	3.96782	0.854826
12	4.738536	0.860688
16	5.815768	0.883257
20	7.185841	0.906145
24	7.289622	0.900333
32	7.360418	0.892014

Parallel function Fp = (No. of Threads/ (No. of Threads-1)) * ((1-1/Speedup))

For n = 2 and s = 1.987128, Fp = (2 / (2-1)) * ((1-1/1.987128)) = 0.993522

For n = 4 and s = 2.169751, Fp = (4 / (4-1)) * ((1-1/2.169751)) = 0.718823

For n = 8 and s = 3.96782, Fp = (8 / (8-1)) * ((1-1/3.96782)) = 0.854826

For n = 12 and s = 4.738536, Fp = (12 / (12-1)) * ((1-1/4.738536)) = 0.860688

For n = 16 and s = 5.815768, Fp = (16 / (16-1)) * ((1-1/5.815768)) = 0.883257

For n = 20 and s = 7.185841, Fp = (20 / (20-1)) * ((1-1/7.185841)) = 0.906145

For n = 24 and s = 7.289622, Fp = (24 / (24-1)) * ((1-1/7.289622)) = 0.900333

For n = 32 and s = 7.360418, Fp = (32 / (32-1)) * ((1-1/7.360418)) = 0.892014

5. Why do the graphs look the way they do? What are they telling you?

➔ The performance graphs provide significant information about the performance improvements with the use of parallel programming. After studying the graphs, we can say that-

1) Performance does not improve significantly if the number of threads or number of trials are low as seen when 1 or 2 threads are used.
2) Performance improvements are first seen when the number of threads and trials are 8 and 1000 respectively. We can say that the experiment has crossed the performance overhead barrier.
3) Performance does not always increase with the increase in number of threads and trials. We can see significant increases in performance up to a point where 20 threads are used. Later, the performance falls drastically when 24 threads were being used. This could be due to thread contention or synchronization overhead on the flip servers.
4) Later, performance is not picked up significantly except when 10000 trials were made use of. This suggests that the system had crossed the point of maximum gain with the help of parallel programming.
5) It is clear from graph 2 that the optimal performance is achieved when the number of threads used is 20.
6) Having understood thread distribution and execution across multiple cores, performance should not have shown an increasing pattern when 24 cores were used. However, the graph still showed a performance rise for most of the cases when 32 threads were used. This shows there was some performance gain.
7) When the experiment was performed with 36 threads, even though there were some increases in performance, it still could not cross the performance that it attained when 24 threads were being used. This meant that there would not be any performance gain after using 24 threads and when the number of trials would be more than 1000.