Sanketh Karuturi
karutusa@oregonstate.edu

CS 575 Parallel Programming
Project#1: OpenMP: Monte Carlo Simulation

This program was run on Flip server. The number of tries (NUMTRIES) is set to 20.

1. Table.1 : Results from running the program:

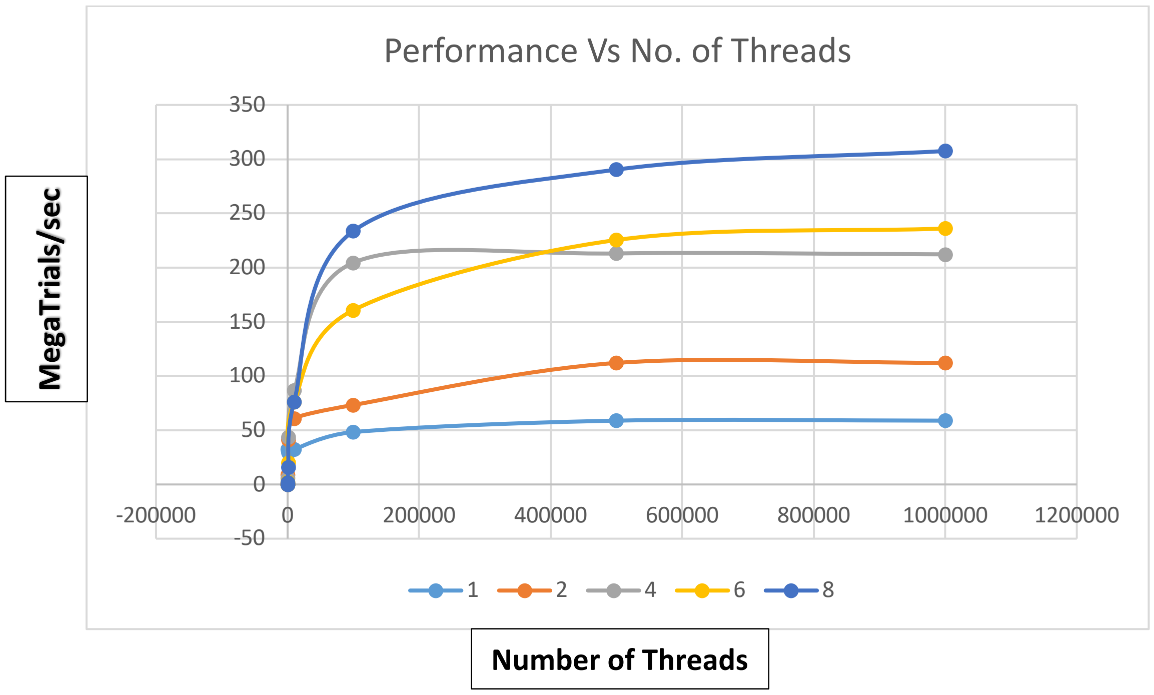
Number of Threads	Number of Trials	Probability	MegaTrialsPerSecond
1	1	0	0
1	10	40	0
1	100	55	32.26
1	1000	56.2	29.54
1	10000	56.02	32.26
1	100000	56.83	48.3
1	500000	56.93	58.85
1	1000000	57.01	58.85
2	1	0	0.13
2	10	40	1.23
2	100	55	9.12
2	1000	56.2	41.53
2	10000	56.02	60.96
2	100000	56.83	73.3
2	500000	56.93	112.08
2	1000000	57.01	112.08
4	1	0	0.08
4	10	40	0.63
4	100	55	5.91
4	1000	56.2	43.46
4	10000	56.02	87.02
4	100000	56.83	204.5
4	500000	56.93	213.04
4	1000000	57.01	212.22

6	1	0	0.03
6	10	40	0.26
6	100	55	2.62
6	1000	56.2	19.97
6	10000	56.02	76.26
6	100000	56.83	160.76
6	500000	56.93	225.43
6	1000000	57.01	236.01
8	1	0	0.02
8	10	40	0.18
8	100	55	1.78
8	1000	56.2	16.13
8	10000	56.02	76.26
8	100000	56.83	233.67
8	500000	56.93	290.34
8	1000000	57.01	307.52

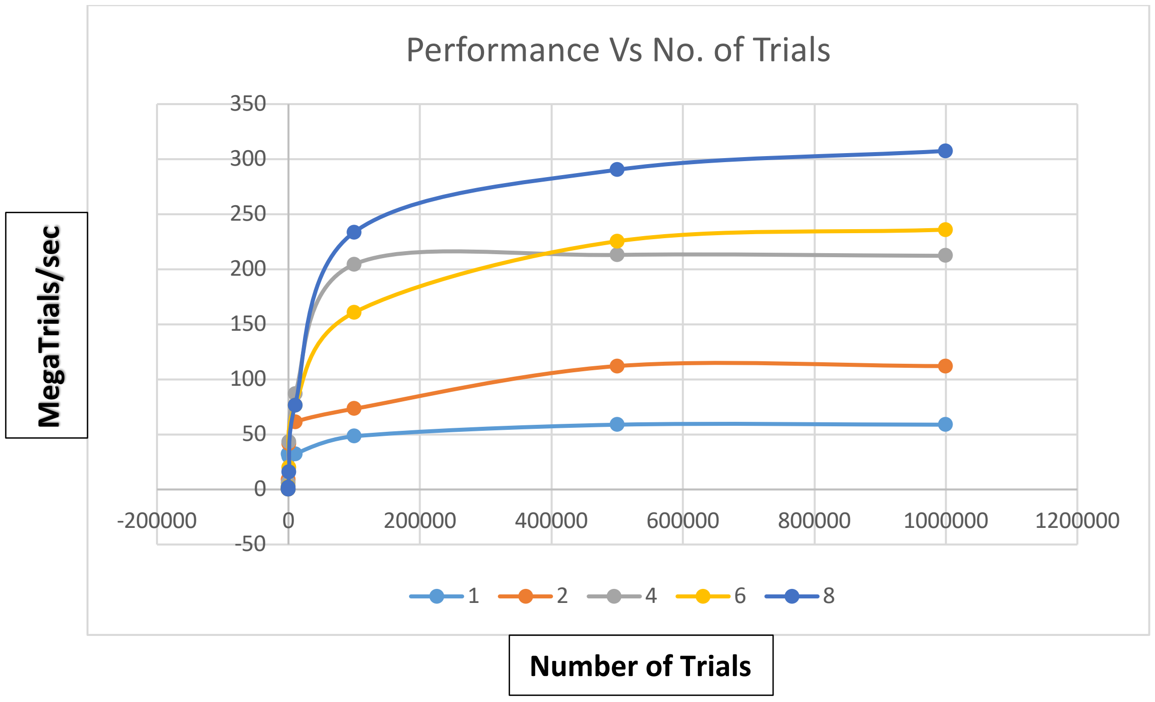
2. A rectangular data table (Table.2) of the performance numbers as a function of NUMT and NUMTRIALS.

	1	**10**	**100**	**1000**	**10000**	**100000**	**500000**	**1000000**
1	0	0	32.26	29.54	32.26	48.3	58.85	58.85
2	0.13	1.23	9.12	41.53	60.96	73.3	112.08	112.08
4	0.08	0.63	5.91	43.46	87.02	204.5	213.04	212.22
6	0.03	0.26	2.62	19.97	76.26	160.76	225.43	236.01
8	0.02	0.18	1.78	16.13	76.26	233.67	290.34	307.52

1. Graph 1 : Performance versus the number of Monte Carlo trials, with the colored lines being the number of OpenMP threads.



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



3. **Estimate of the Probability.**

From Table 1, we can calculate the probability = (57.01 + 57.01 + 57.01 + 57.01 + 57.01) / 5

= 57.01%

4. Compute Fp, the Parallel Fraction, for this computation.

SpeedUp(Sp) = Performance with Thread 8 / Performance with Thread 1

= 307.52 / 58.85 = 5.2255

$$F_p(Parallel\ Fraction) = \frac{n}{n-1}\left(1 - \frac{1}{Speedup}\right) = \frac{8}{8-1}\left(1 - \frac{1}{5.2255}\right) = 0.9241$$

Where *Sp* is the speedup, and *n* is the number of threads. The calculated *Fp* suggests that the task is highly parallelizable, and as seen in the performance graphs, the increase in threads initially leads to significant performance gains, demonstrating effective scaling and utilization of the parallel architecture provided by OpenMP.

Commentary

In Graph 1: Performance vs. Number of Trials. The performance, measured as Mega-Trials Per Second, increases with the number of trials. This trend suggests that the OpenMP implementation effectively utilizes the parallel architecture when handling larger datasets**.** With an increase in the number of trials, the overhead of thread management and synchronization becomes relatively insignificant compared to the computational work, leading to better performance scaling. Additionally, the graph serves as a practical guide for choosing an appropriate number of threads for different sizes of the problem: use fewer threads for smaller problems and more threads for larger ones, but we should always be wary of the point where adding more threads does not result in proportional performance gains.

From the graph 2, we observe that performance increases with the number of trials, indicative of efficient parallel execution. Notably, the performance gains from increasing the number of threads are significant, with the highest performance achieved using eight threads, followed closely by six threads, which suggests a well-optimized parallel processing system. Each thread count shows a rising trend, but the four-thread and two-thread counts level off more quickly than the six and eight, implying a potential saturation point where additional trials do not yield a proportional increase in performance due to possible limitations such as resource constraints or thread management overhead.