Parallel Computing Homework 1

Tyler Scotti

January 23rd, 2026

Part 1: Parallely Calculating PI with Shared-Memory via Threads

Number of Trials: 10^8

Question 1.1: Execution Time vs Number of Threads

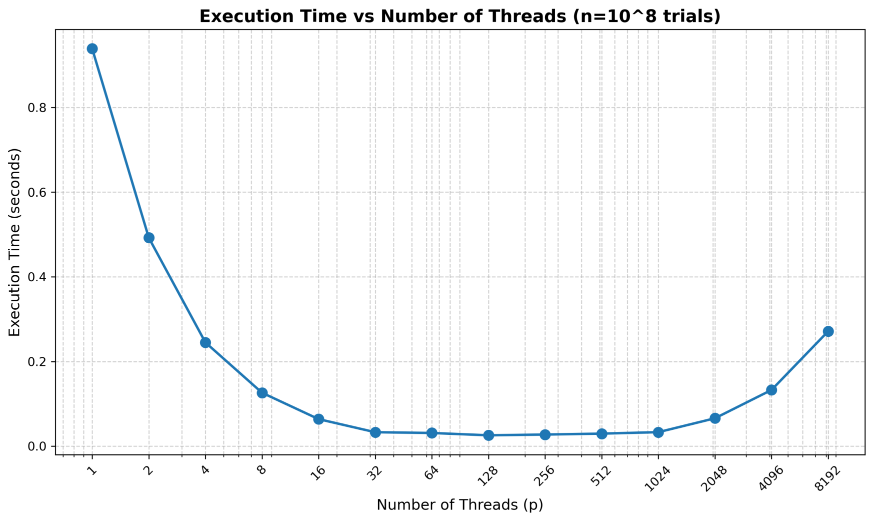

Figure 1: Execution Time vs Threads

The graph demonstrates how execution time varies with the number of threads for 10^8 Monte Carlo trials. Execution time decreases from 0.94 seconds with a single thread to a minimum of 0.026 seconds at 128 threads, achieving substantial performance gains. Beyond 128 threads, execution time increases due to thread management overhead and synchronization costs exceeding the benefits of parallelization.

Question 1.2: Speedup vs Number of Threads

The speedup curve shows near-linear scaling from 1 to 32 threads, reaching a maximum speedup of 36.52x at 128 threads. After this peak, speedup decreases as the overhead from thread creation, scheduling, and synchronization dominates. This behavior is consistent with Amdahl's Law, where the sequential portion and parallel overhead limit achievable speedup.

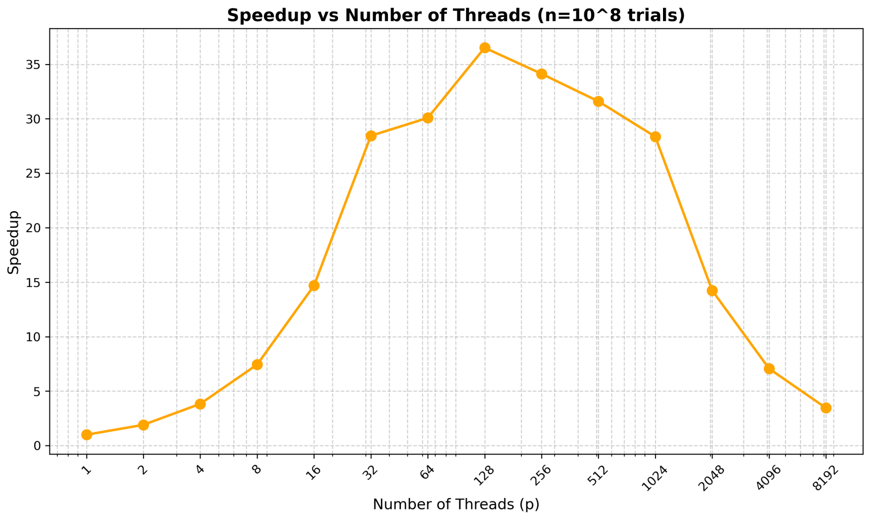

Figure 2: Speedup vs Threads

Question 1.3: Efficiency vs Number of Threads

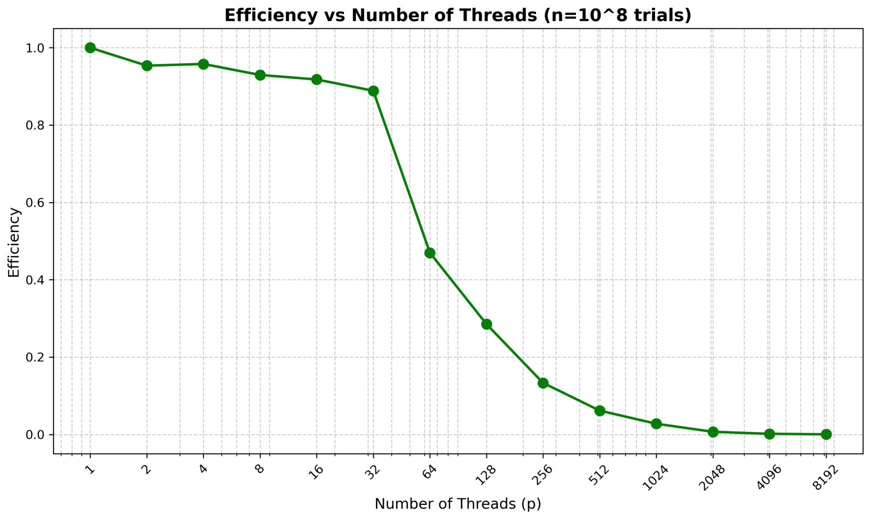

Figure 3: Efficiency vs Threads

Parallel efficiency (defined as Efficiency $= \frac{\text{Speedup}}{p}$) remains high (>90%) for up to 32 threads, indicating effective utilization of parallel resources. Beyond this point, efficiency drops sharply, reaching approximately 28.5% at 128 threads and continuing to decline. This trend reflects diminishing returns as thread overhead becomes more significant relative to computational work per thread.

Question 1.4

Q: In your experiments, what value of p minimizes the parallel runtime?

A: 128 threads minimizes the parallel runtime at 0.0257 seconds, representing a 36.52x speedup over sequential execution. While higher thread counts are available, they introduce more overhead than performance benefit.

Question 2

Number of Trials: 10^{10}

Question 2.1:

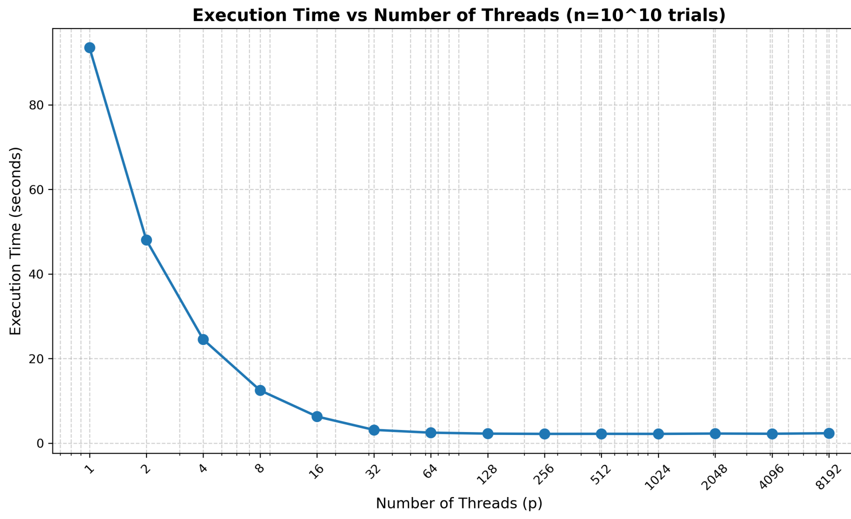

Figure 4: Execution Time vs Threads

Q: In this case, what value of p minimizes the parallel runtime?

A: 256 threads for 10^{10} trials resulted in the minimum execution time.

Question 2.2:

Q: Do you expect the runtime to increase as p is increased beyond a certain value? If so, why? And is this observed in your experiments.

A: Yes, although we increased the problem size, eventually, the thread overhead will outgrow the benefit gained from parallelization.

Question 3

Q: Do you expect that there would be a difference in the number of threads needed to obtain the minimum execution time for two values of n? Is this observed in your experiments.

A: Yes, I would expect the optimal number of threads to increase with the problem size, and this is confirmed by my results. For 10^8 trials, the optimal thread count was 128, but with 10^{10} trials, it increased to 256 threads.

Question 4

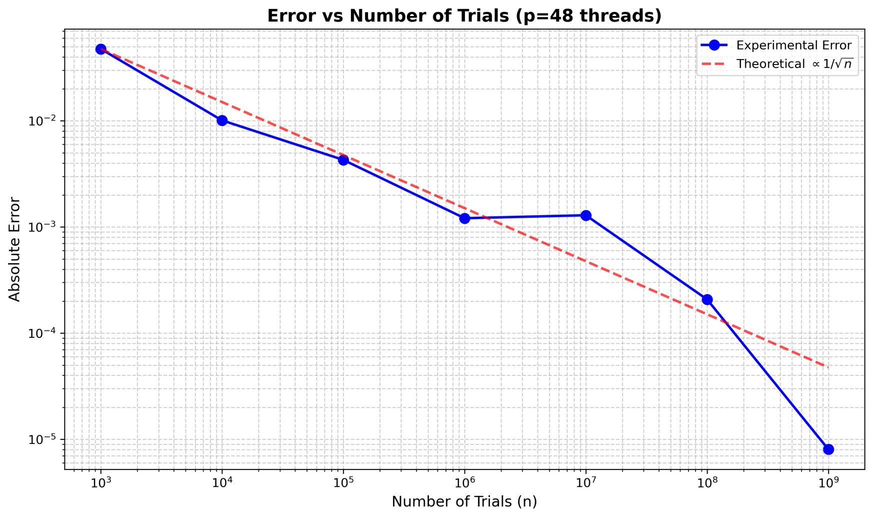

Figure 5: Error vs Trials

Part 2: Parallely Calculating PI via MPI

Number of Trials: 10^{10}

Question 5.1: Execution Time vs Number of Processes

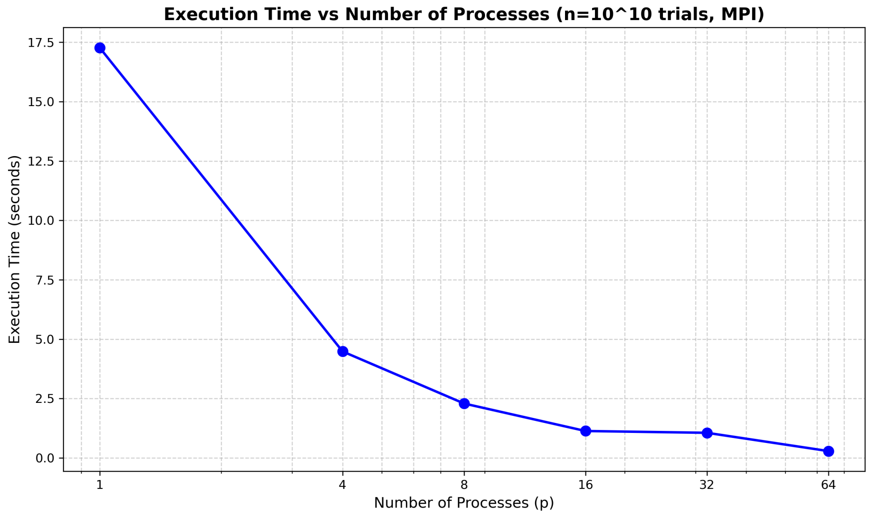

Figure 6: Execution Time vs Processes

The graph illustrates how execution time varies with the number of MPI processes for 10^{10} Monte Carlo trials. Execution time decreases dramatically from 17.27 seconds with a single process to just 0.29 seconds with 64 processes. Unlike the shared-memory thread implementation, the MPI approach shows consistent performance improvement across all tested process counts, with no performance degradation at higher process counts.

Question 5.2: Speedup vs Number of Processes

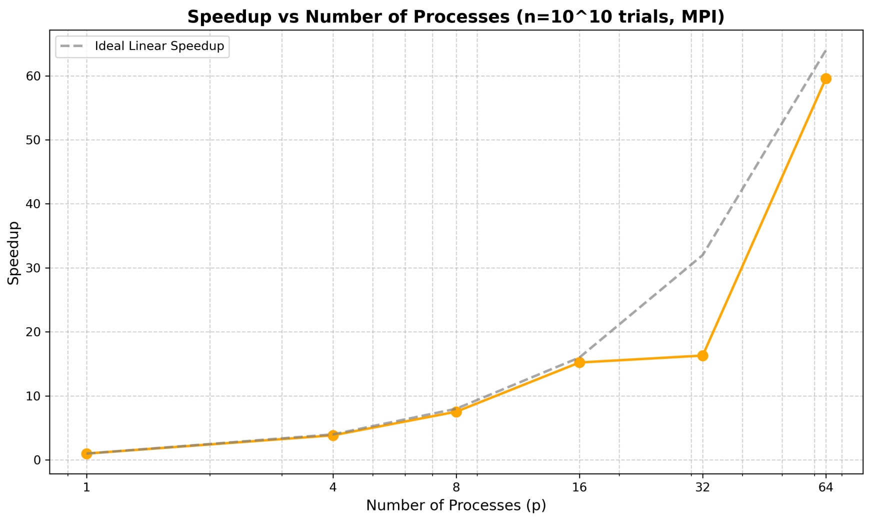

Figure 7: Speedup vs Processes

The speedup curve demonstrates excellent scaling characteristics of the MPI implementation. The algorithm achieves near-linear speedup up to 16 processes, reaching 15.22x speedup (95% efficiency). At 64 processes, the speedup reaches 59.58x, which represents 93% parallel efficiency. The gray dashed line shows ideal linear speedup for comparison. The MPI implementation significantly outperforms the shared-memory approach.

Question 5.3: Efficiency vs Number of Processes

Parallel efficiency remains remarkably high across all process counts, staying above 93% for most configurations. There is a notable dip to approximately 51% at 32 processes, which may indicate suboptimal load balancing at that specific configuration. However, efficiency recovers to 93% at 64 processes.

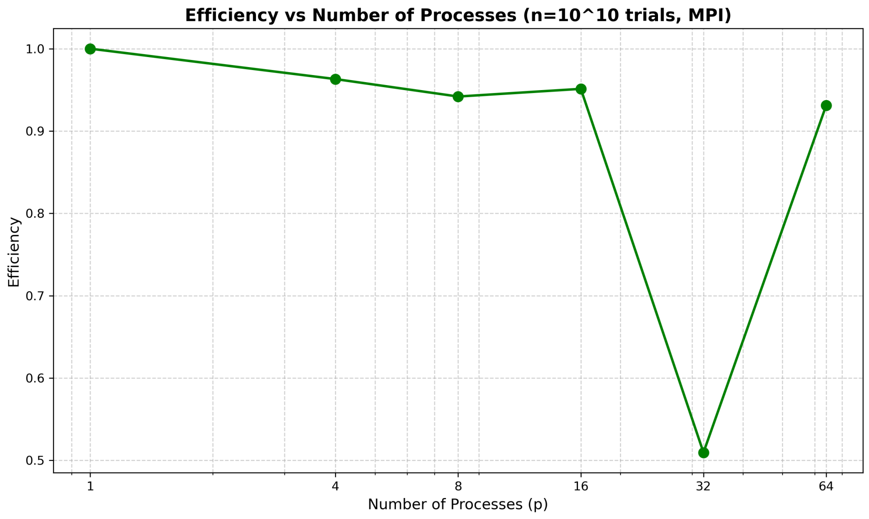

Figure 8: Efficiency vs Processes

Question 5.4

Q: What value of p minimizes the parallel runtime?

A: 64 processes minimizes the parallel runtime at 0.2899 seconds, achieving a 59.58x speedup over sequential execution with 93% parallel efficiency.

Question 6: Time vs N-TasksPerNode

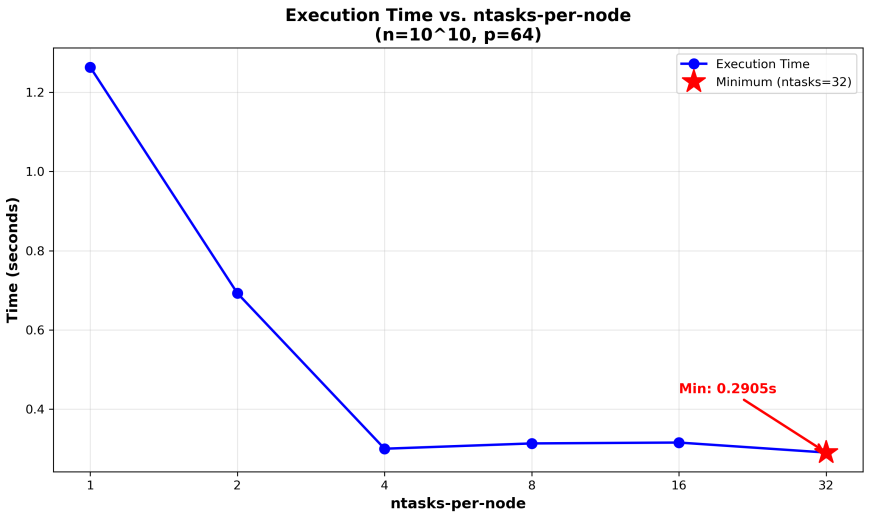

Figure 9: Time vs TasksPerNode

Question 7

Question 7.1: Speedup Observed

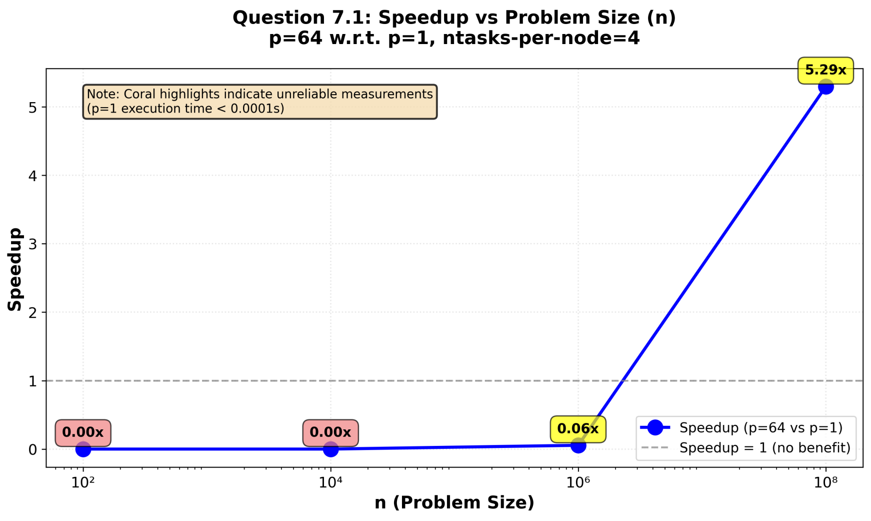

Figure 10: Speedup

Question 7.2: Error vs Problem Size

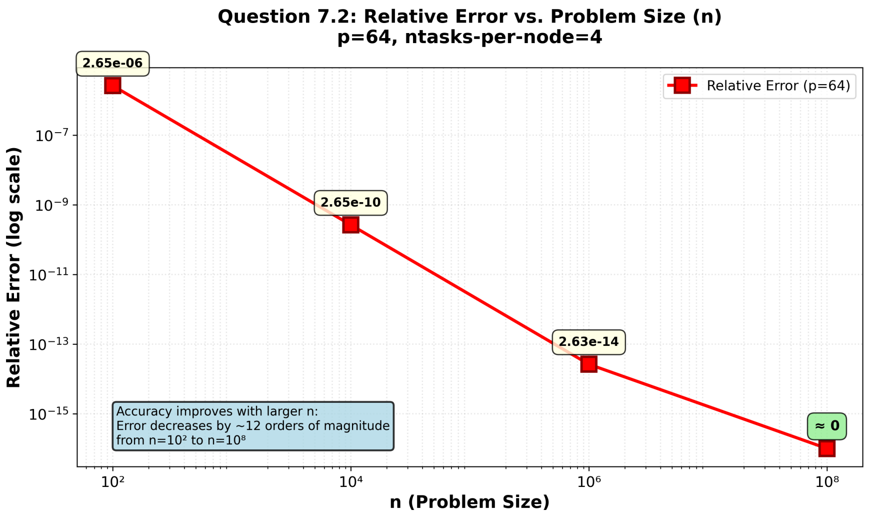

Figure 11: Error vs Problem Size