SENG 442

Parallel and Distributed Computing

Eren Kaan Çakır

Berke Beyazbenli

Parallel Computing Performance Analysis

Introduction

This report analyzes the performance of a parallelized algorithm for detecting perfect squares in a given array. The program uses pthreads to distribute the workload among multiple threads. The goal is to measure execution time, speed-up, and efficiency for different thread counts and problem sizes.

Methodology

The algorithm checks whether numbers in a given array are perfect squares. The workload is divided evenly among available threads. Each thread processes a subset of the array and reports its execution time. The total execution time is recorded, and speed-up and efficiency are computed using the following formulas:

• Speed-Up (S) = Execution time with 1 thread / Execution time with N threads

• Efficiency (E) = Speed-Up / Number of Threads

Discussion

From the results, we observe that:
- For small arrays (100,000 elements), performance fluctuates due to thread overhead.
- Speed-up is not always linear; increasing thread count does not always reduce execution time due to scheduling and memory contention.
- Efficiency drops significantly as the number of threads increases, suggesting diminishing returns due to overhead.

Execution Time Table (Seconds)

Array Size	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
100000	0.002137	0.001380	0.001475	0.002426	0.001629	0.001994	0.006018
1000000	0.014362	0.016234	0.010395	0.022928	0.018728	0.021729	0.032185
5000000	0.072111	0.069652	0.078445	0.072121	0.072127	0.095959	0.075309

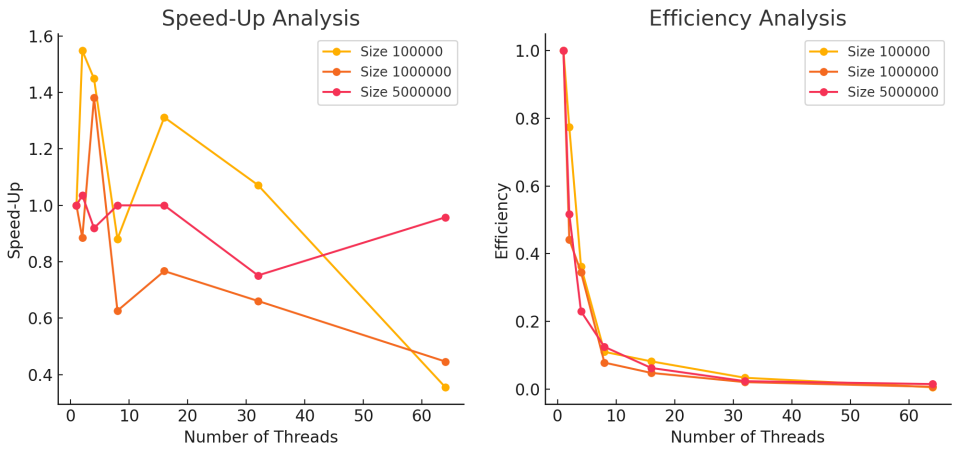
Speed-Up Analysis

Array Size	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
100000	1.00	1.55	1.45	0.88	1.31	1.07	0.36
1000000	1.00	0.88	1.38	0.63	0.77	0.66	0.45
5000000	1.00	1.04	0.92	1.00	1.00	0.75	0.96

Efficiency Analysis

Array Size	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
100000	1.00	0.77	0.36	0.11	0.08	0.03	0.01
1000000	1.00	0.44	0.35	0.08	0.05	0.02	0.01
5000000	1.00	0.52	0.23	0.12	0.06	0.02	0.01

Graphs for Performance Analysis



Conclusion

Parallelization improves performance to some extent, but increasing threads beyond a certain point leads to inefficiencies. Future improvements can involve optimizing memory access patterns and reducing synchronization overhead.