Performance Analysis Report

May 06, 2025

1 Introduction

This report provides an in-depth analysis of performance metrics for a parallel computing task across varying configurations of MPI processes and OpenMP threads, with n values of 4, 5, 6, and 7. The focus is on total execution time, section timing breakdowns, and strong scaling speedup, evaluating general trends and deviations from expected results.

2 Methodology

The study assesses configurations with combinations of MPI processes (np) and OpenMP threads (t), ranging from $np = 1, t = 1$ to $np = 4, t = 4$. Data includes total execution time and sectional breakdowns (Graph, Partition, LocalVertices, IST, Gather, Output), with a speedup analysis for $n = 4$ and threads=1.

3 Results

3.1 General Trends

A consistent trend across all n values shows that total execution time increases with higher thread and process counts, particularly noticeable at $np = 4, t = 4$. For section timings, the Graph section dominates for $n = 5, 6,$ and 7, while Output and Gather times grow significantly with $n = 6$ and $n = 7$. The speedup for $n = 4$ approaches ideal scaling up to 2 MPI processes but declines thereafter.

3.2 Deviation from Expected Results

Expected results anticipated near-linear speedup with increasing processes and threads due to enhanced parallelism. However, the observed decline in speedup beyond 2 MPI processes and increased execution time with higher thread counts suggest sub-optimal scaling. This deviation indicates potential overhead or inefficiencies not fully offset by parallel gains.

3.3 Potential Causes

The discrepancy may arise from synchronization overhead between MPI processes and OpenMP threads, load imbalance across tasks, or memory contention in multi-threaded environments.

For higher n, the Graph and Gather sections' complexity might exacerbate these issues, reducing parallel efficiency.

3.4 Impact on Scalability and Performance

This behavior limits scalability, as adding more processes and threads beyond a certain point (e.g., $np = 2, t = 2$) does not yield proportional performance gains and may degrade it due to overhead. Performance peaks at moderate parallelism, suggesting a trade-off between resource utilization and coordination costs.

3.5 Effect of Increasing Threads and Processes

Increasing threads and processes improves parallel section execution (e.g., Graph, IST) by distributing workloads. However, the overall code performance does not improve proportionally due to sequential bottlenecks (e.g., Output) and overhead, indicating that benefits are confined to parallelizable sections rather than the entire application.

3.6 Performance on Different Computers

On systems with higher core counts and faster interconnects (e.g., multi-node clusters), scalability might improve due to reduced communication latency and better thread utilization. Conversely, on systems with limited memory or slower CPUs, the overhead could worsen, leading to poorer performance, especially for $n = 7$ configurations with high thread counts.

4 Conclusion

The analysis reveals a general trend of diminishing returns with increased parallelism, with optimal performance around $np = 2, t = 2$. Deviations from expected linear speedup highlight overhead and load imbalance issues, impacting scalability. While parallel sections benefit, overall code performance is constrained, and hardware specifications significantly influence outcomes.

5 Recommendations

Investigate dynamic load balancing and thread affinity to mitigate overhead. Test hybrid configurations (e.g., $np = 3, t = 2$) on diverse hardware to identify optimal setups.