Evaluating Parallel Graph Coloring for Conflict-Free Graph Processing

Amin Hong(minhong)   Jinrae Kim (jinraek)

December 1, 2025

URL: https://aminh0.github.io/15-618-project/

1 Project Overview

This project accelerates Laplacian smoothing on large graphs using graph-coloring-based batching. We evaluate three coloring strategies—sequential greedy, speculative parallel greedy, and Luby MIS—and study how color count and scheduling influence parallel performance.

2 Updated Schedule

2.1 Completed Work

Table 1 summarizes the work completed so far, organized by week and team member.

Week	Completed Tasks (Member)
Week 1	• Designed and implemented overall graph data structures — **Amin** • Implemented sequential greedy coloring baseline — **Jinrae** • Implemented speculative parallel greedy coloring — **Amin**
Week 2	• Implemented Luby MIS-based parallel coloring — **Amin** • Integrated color-class batching for Laplacian smoothing — **Jinrae** • Implemented Burke refinement heuristic — **Jinrae** • Performed evaluation and analysis of all coloring modes (greedy, speculative, MIS) — **Amin**

Table 1: Completed work by week and member.

2.2 Updated Plan for Remaining Work

We provide an updated, more detailed plan for the remaining project period.

Week 3 (Mon–Wed)

- Add and evaluate an additional coloring algorithm (e.g., Kempe-based refinement)
- Prototype Kempe-chain recoloring and test feasibility on medium-size graphs
- Integrate new algorithm into the unified benchmarking framework
- Collect preliminary results comparing all algorithms

Week 3 (Thu–Fri)

- Run full-scale experiments on GHC and PSC machines
- Collect performance data across multiple densities ($p = 0.001$–0.5)

- Analyze runtime breakdown, color counts, and parallel scaling
- Generate plots and organize evaluation figures for the write-up
- Final code cleanup and prepare submission materials

3 Work Completed So Far

Over the first two weeks, we implemented the full pipeline needed for graph-coloring–based batching. Our implementation includes three different coloring strategies. Sequential greedy coloring processes vertices in order and assigns the lowest available color. Speculative parallel greedy coloring extends this by tentatively coloring all vertices in parallel, detecting conflicts, and iteratively recoloring only the conflicting vertices—mimicking Jones–Plassmann style speculative [2]. Luby MIS–based coloring constructs color classes using repeated rounds of Luby's parallel Maximal Independent Set algorithm: in each round, vertices with locally maximal random priorities enter the MIS and receive the next color[3]. These three methods provide different trade-offs between parallelism and color count.

We additionally implemented two kernels that operate on the coloring output. The Laplacian smoothing kernel uses color classes as conflict-free batches, updating all vertices of the same color in parallel while respecting graph dependencies[1]. This is implemented as a nested loop over color classes with OpenMP per-class parallelism. The Burke refinement kernel is a local-search post-processing pass: each vertex considers whether switching to a different color would reduce conflicts and updates its color accordingly, using a parallel reduction across vertices. All components—including the random graph generator, conflict detector, color-class builder, and unified command-line interface (-g, -gp, -lb, -B, -L)—are fully implemented and tested for correctness on small graphs.

4 Updated Goals

- **50% Completion:** Implemented a scalable graph generator supporting arbitrary node counts and densities. Added three initial coloring strategies: sequential greedy, speculative parallel greedy, and Luby MIS. Integrated a command-line interface for selecting coloring modes and graph parameters.
- **75% Completion:** Implemented Burke refinement for post-processing color assignments. Integrated color-class batching into the Laplacian smoothing kernel. Completed correctness checks on small graphs and validated that smoothing runs end-to-end on all coloring outputs.
- **100% Completion:** Conduct full performance evaluation across multiple graph densities and sizes. Measure parallel speedup and runtime breakdown for each coloring algorithm. Analyze how color count, load balance, and graph structure affect smoothing throughput and parallel scaling
- **125% Completion (Stretch Goal):** Introduce "fixed palette size" experiments: Force color counts (e.g., 100, 80, 60, 40, 20 colors) and compare runtime differences. Analyze how reducing available colors affects parallelism and workload balance.
- **150% Completion (Stretch Goal):** Estimate or search for the "optimal" number of colors that balances parallelism and batch size. Implement a simple heuristic or adaptive coloring refinement to target near-optimal color counts. Conduct expanded evaluations on larger datasets or more smoothing iterations.

5 Preliminary Results

Using a fixed input size of n = 10,000 and edge probability $p = 0.01$, we measured the strong-scaling behavior of the different coloring algorithms under 1, 2, 4, and 8 OpenMP threads. Speculative greedy coloring showed limited scalability, achieving at most a 1.92× speedup at 8 threads. This is because, for sparse graphs with very few conflicts, the speculative phases offer little parallel work and the algorithm becomes dominated by memory-bound adjacency scans, leading to substantial synchronization and cache contention overhead.

In contrast, Luby MIS exhibited nearly ideal scaling, reaching up to 5.92× speedup at 8 threads. Since each MIS round is inherently parallel and does not require conflict resolution or recoloring, the algorithm exposes significantly more parallelism and amortizes per-thread overhead more effectively. These preliminary results indicate that MIS-based methods are more suitable for high-parallelism environments, while speculative greedy coloring is constrained by structural dependencies and memory access bottlenecks.

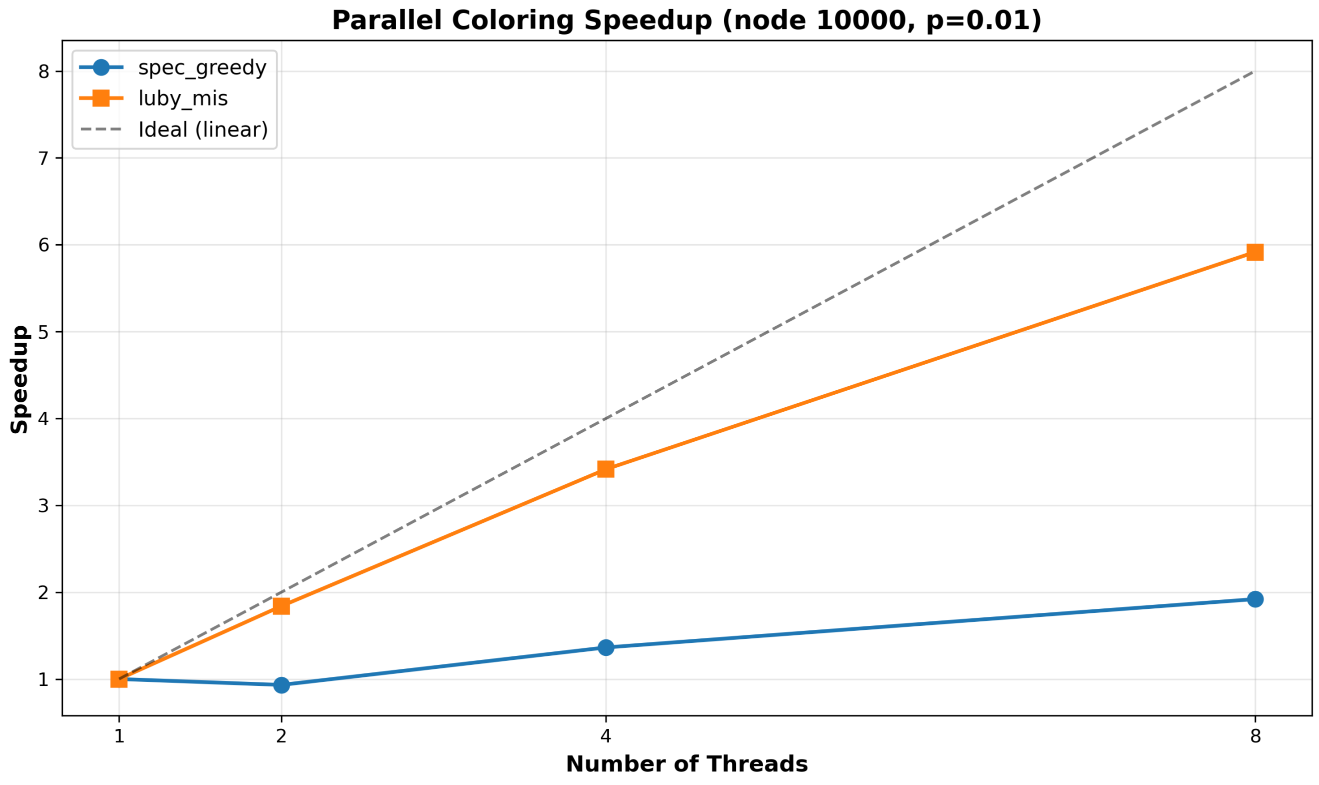

Figure 1: coloring algorithm speedup comparison

6 Issue/Concerns

One immediate concern is the instability of the Kempe-chain–based refinement strategy[4] we explored. Although Kempe swaps are theoretically capable of resolving local conflicts in fixed- K colorings, our initial attempts show that they do not reliably converge to a conflict-free assignment on random graphs. In dense regions of the graph, the Kempe chain often grows large or cyclic, causing recolorings that simply reintroduce conflicts elsewhere. This makes it unclear whether Kempe refinement can be used as a practical parallel conflict-resolution mechanism in our pipeline.

A second concern is the general difficulty of maintaining both parallelism and low color count. The speculative greedy method produces near-optimal colorings but can incur significant conflict-driven restarts, while Luby MIS is highly parallel but tends to generate a large number of colors. Balancing these two competing properties—parallel scalability and color efficiency—may require additional heuristics or hybrid methods that we have not yet tested.

7 Remaining Unknowns

Several important aspects of our evaluation are still unresolved at this stage. First, we have not yet run our implementation on the PSC machines, so we do not know how well our current code and OpenMP configuration will scale on the target hardware. Performance characteristics such as cache behavior, NUMA effects, and thread scheduling may differ significantly from our GHC test environment, and it is unclear whether any of the coloring algorithms will exhibit unexpected bottlenecks or instability at larger core counts.

Second, we have not yet established a clear guideline for the graph configurations used in our evaluation. Up to this point, we have only tested whether the coloring algorithms run correctly and in parallel for arbitrary choices of node count and connectivity. We have not examined how performance or color quality vary as a function of graph structure, nor have we identified a principled range of graph sizes or densities that should represent our core workloads. As a result, our current experiments do not yet capture how graph characteristics influence the behavior of each algorithm, and we still need to define a consistent and meaningful set of evaluation cases for the final report.

References

[1] Field, D. A. Laplacian smoothing and delaunay triangulations. *Communications in Applied Numerical Methods 4*, 6 (1988), 709–712.

[2] Gebremedhin, A., and Manne, F. Scalable parallel graph coloring algorithms. In *International Conference on High Performance Computing* (2000), Springer, pp. 241–256.

[3] Luby, M. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1985), STOC '85, Association for Computing Machinery, p. 1–10.

[4] Whitney, H., and Tutte, W. Kempe chains and the four colour problem. In *Hassler Whitney Collected Papers*. Springer, 1992, pp. 185–225.