Parallel Graph Coloring Milestone Report

Link to Project Website: https://jerryliang49.github.io/15418-Final-Project/

Work Done So Far:

We have made solid progress so far. To start, we set up the foundational infrastructure for our parallel graph coloring project. For the graph representation we chose Compressed Sparse Row (CSR) format, which stores each vertex's neighbors contiguously to enable cache-efficient sequential access and good hardware prefetcher behavior. We wrote loaders for both edge-list (SNAP) and METIS formats with automatic detection by file extension. To build a diverse benchmark suite, we wrote a Python graph generator supporting three models, namely Erdos-Renyi (uniform random), R-MAT/Kronecker (power-law degree distributions mimicking real social and web networks), and 2D grids (perfectly regular structure). We also converted three real-world datasets, including the Enron email network (36K vertices), Amazon co-purchasing graph (262K vertices), and Pennsylvania road network (1.09M vertices), giving us a benchmark suite that spans five structural classes and four sizes. In terms of implementation, we first finished a sequential greedy coloring routine that serves as both our correctness reference and quality baseline. The key optimization is sizing the used[] array to max_degree + 2 rather than the vertex count and only resetting the first degree(v) + 2 entries per vertex, which keeps the array in L1 on sparse graphs and avoids wasted clears on power-law graphs where most vertices have low degree.

For our parallel implementation we adopted the Gebremedhin-Manne speculative coloring algorithm, which works in rounds. Phase 1 tentatively colors all worklist vertices in parallel using per-thread used[] arrays, accepting that some adjacent vertices may receive the same color due to concurrent writes. Phase 2 then detects conflicts by scanning edges, and when two adjacent vertices share a color the higher-indexed vertex loses and is added back to the worklist. This rule is deterministic, requires no synchronization, and guarantees forward progress. Our initial implementation used schedule(dynamic, 64) to handle load imbalance from varying vertex degrees and collected conflicts via thread-local vectors merged under #pragma omp critical. We instrumented three timing components (init, compute, total) and tracked conflicts, rounds, and colors used. On GHC's 8-core machines this baseline achieved 4-5x compute-time speedup on our medium and large graphs (4.5x on er_100k, 4.7x on amazon0302, 4.0x on roadnet-pa), with conflict rates below 0.25% and convergence in at most 2 rounds. We identified several remaining bottlenecks, including fork/join overhead on small graphs, the critical section serializing conflict merging, load imbalance on power-law graphs from hub vertices, and a fully serial initialization phase.

Some Preliminary Results

Parallel Speedup (compute time, 1-thread parallel as baseline)

Graph	1T comp (ms)	2T speedup	4T speedup	8T speedup
er_10k	0.44	1.3x	1.6x	1.2x
rmat_10k	0.41	1.4x	1.7x	1.1x
grid_100	0.14	0.7x	0.6x	0.4x
email-enron	1.50	1.6x	2.5x	2.8x
er_100k	5.16	1.8x	3.2x	4.5x
rmat_100k	4.06	1.8x	2.9x	3.9x
grid_100k	1.34	1.4x	2.2x	2.0x
amazon0302	11.39	1.8x	3.3x	4.7x
roadnet-pa	27.32	1.6x	2.6x	4.0x

Conflict Rate

Graph	Vertices	Conflicts	Conflict Rate	Rounds
er_10k	10,000	10	0.10%	2
rmat_10k	10,000	0	0.00%	1
email-enron	36,692	56	0.15%	2
er_100k	100,000	14	0.01%	2
rmat_100k	100,000	13	0.01%	2
amazon0302	262,111	98	0.04%	2
roadnet-pa	1,088,092	2,234	0.21%	2

Initialization Time

Graph	Init Time
er_1m	40.1ms
roadnet-pa	34.2ms
amazon0302	9.8ms
er_100k	3.7ms
rmat_1m	66.1ms

Our parallel speculative implementation scales well on medium and large graphs, reaching 4-5× compute-time speedup at 8 threads on amazon0302, er_100k, and roadnet-pa, but stays flat or regresses on the smallest inputs where the parallel work is too small to amortize thread-management overhead. Conflict rates stay below 0.25% on every graph and convergence happens in at most two rounds, confirming that the speculative approach is not bottlenecked by wasted work. The main pattern to address next is the initialization phase, which takes 40-66 ms on the 1M-vertex graphs and meaningfully caps end-to-end speedup. Reducing this overhead, along with improving load balance on power-law graphs, will be the focus of our next optimization iteration.

Original Goals and Deliverables

We are on track to complete all of our original deliverables and goals. The sequential greedy baseline is done and serves as our correctness and quality reference. We have completed one parallel implementation version with decent results and are currently analyzing the bottlenecks to see what we need to optimize in the next iterations. All of the testing scaffold and benchmarking/analysis scripts are in place, which will help speedup the iteration process. In terms of "**nice to haves**", we have already completed or are in process of completing a few of them, including 1) comparing the behavior of the algorithm on very different graph classes, and 2) Exploring whether graph preprocessing or vertex ordering can reduce conflict rates and improve scaling and 3) Implementing several parallel coloring strategies (currently exploring a speculative + Jones-Plassmann hybrid approach). If time allows, which we believe it will, we plan to implement and optimize a GPU version as well.

Revised Goals

- Implement a hybrid speculative + Jones-Plassmann coloring algorithm as a second parallel variant.
- Further optimize our existing speculative implementation based on bottleneck analysis (scheduling strategy, graph representation, preprocessing, init-phase overhead).
- Run a full evaluation across all graph classes and thread counts, with enough analysis depth to explain the results rather than just report them.
- Implement and optimize a GPU version of speculative coloring if time allows.

Revised Schedule

Due to our early presentation time, our schedule is quite condensed.

Time Frame	Goals
April 15 - April 16	● Implement hybrid speculative + Jones-Plassmann algorithm (Jerry) ● Optimize existing speculative implementation based on bottleneck analysis (Harry)
April 17 - 18	● Run large-scale experiments across all graphs and thread counts, finalize plots (Harry). ● Begin drafting the final report (Jerry).
April 19 - April 21	● Keep working on report (Both) ● Implement GPU Version (if time allows) (Harry) ● Optimize GPU Version (if time allows) (Jerry)
April 22	● Finalize report (Both) ● Begin working on Poster (Both)
April 23	● Finish poster (Both) ● Finalize website (Harry) ● Submit Report (Jerry)

Poster Session

We will not have a live demo during the presentation. We will have plots of different types (line, bar, scatter) showing speedup, conflict rate, init-versus-compute time breakdowns, and comparisons between different approaches (speculative, hybrid, GPU) and different graph families. We will also include a summary table comparing colors used and total runtime across algorithms and a short methodology note explaining our two speedup baselines (relative to parallel at 1 thread and relative to pure sequential).

Concerns and Issues

One concern we have is the initialization overhead. On graphs smaller than roughly 1M vertices, initialization (parallel degree computation, hub partitioning) accounts for most of our total runtime at 8 threads, which caps the total speedup. Computation speedup is quite good. We are not yet sure how much of this can be reduced without compromising algorithm quality.

Another concern is the GPU stretch goal. Implementing, debugging, and benchmarking a CUDA port on top of the remaining CPU-side work within a few days is tight, and we may end up discarding it if the CPU-side analysis takes longer than expected.