Parallel Graph Coloring for Social Network Clustering

Prajwal Manik Garje
Shayan Rahimi
Aida Sharbatdar
Anamay Brahme
Ugochukwu Vigilus Obioha
Department of Technology
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
email address or ORCID

Mugdha Kashyap
Department of Technology
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
hashim.ali@ue-germany.de

***Abstract*—Social networking connects millions of users through friendships and interactions, forming complex graphs. Analyzing these networks helps in understanding communities, testing features on groups, and minimizing opinion overlap. Graph coloring assigns each user (node) to a group (color) so that no two directly connected users share the same group. This project implements parallel graph coloring for large-scale social networks using OpenMP, ensuring efficient partitioning with reduced execution time.**

I. Introduction

Social networking connects millions of users through friendships and interactions, forming complex graphs. Analyzing these networks helps in understanding communities, testing features on groups, and minimizing opinion overlap.

Graph coloring assigns each user (node) to a group (color) so that no two directly connected users share the same group. This project implements parallel graph coloring for large-scale social networks using OpenMP, ensuring efficient partitioning with reduced execution time.

II. Problem Statement

The project groups Facebook users into separate communities based on their connections, ensuring:

- No two directly connected users are placed in the same group.
- Groups support real-world use cases like testing features, understanding social circles, and reducing opinion spread.
- Parallel algorithms achieve scalability while maintaining correctness.

III. Dataset

- Source: Facebook Social Circles dataset.
- Nodes: Users.
- Edges: Friendships (connections).
- Collected via a Facebook survey application.
- Includes circles (friend lists) and profile features.

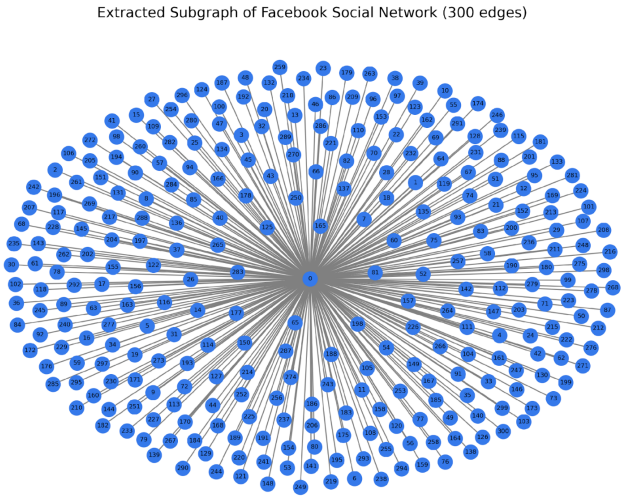

Fig. 1. Dataset Diagram

IV. Approach

A. Graph Traversal

- Ensures the graph is connected.
- Parallel Breadth-First Search (BFS) using OpenMP:
  - Explores graph level by level.
  - Marks nodes as visited in parallel.
  - Visualization: green = visited nodes.

B. Graph Coloring Algorithm

- Greedy algorithm assigns the lowest available color to each node.
- Parallel version:
  - Divides nodes among threads.
  - Uses locks and conflict detection to avoid race conditions.
  - Results in distinct groups (colors = groups).

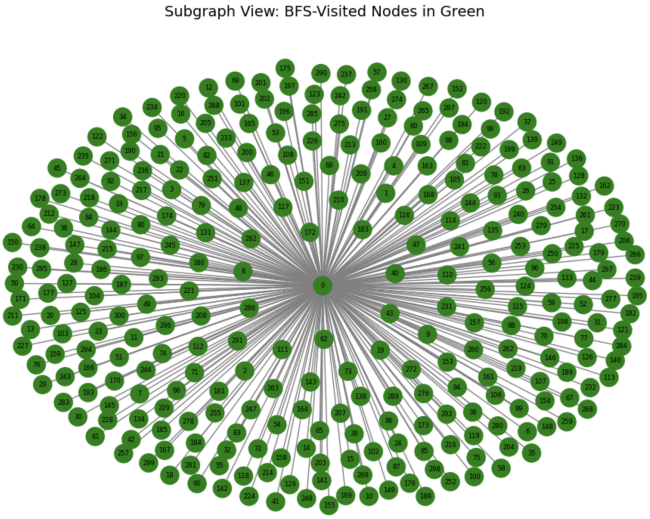

Fig. 2. Parallel BFS Visualization

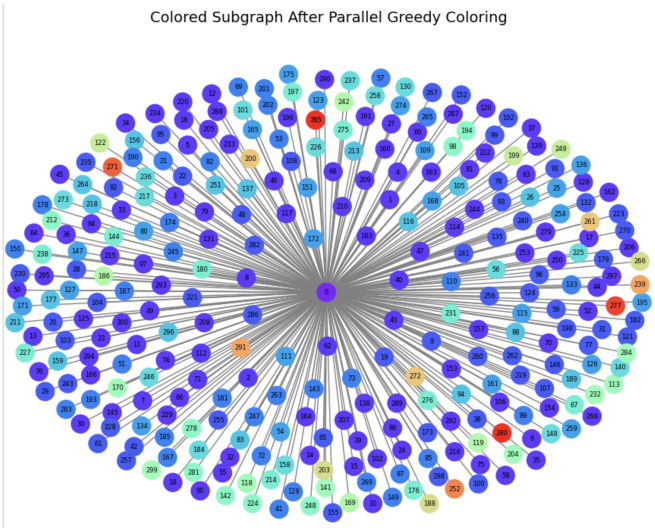

Fig. 3. Parallel Coloring Algorithm

V. Results

- Coloring Output: 86 colors used, indicating strong connectivity and few large independent clusters.
- Visualization: Subgraphs reveal intertwined patterns, confirm algorithm correctness, and provide qualitative understanding.

VI. Technical Implementation

A. Sequential Graph Coloring

- Algorithm: Greedy coloring using adjacency list.
- Implementation:
  - Boolean array tracks unavailable colors.
  - Metrics: total colors, mean color, variance, CPU time, execution time.

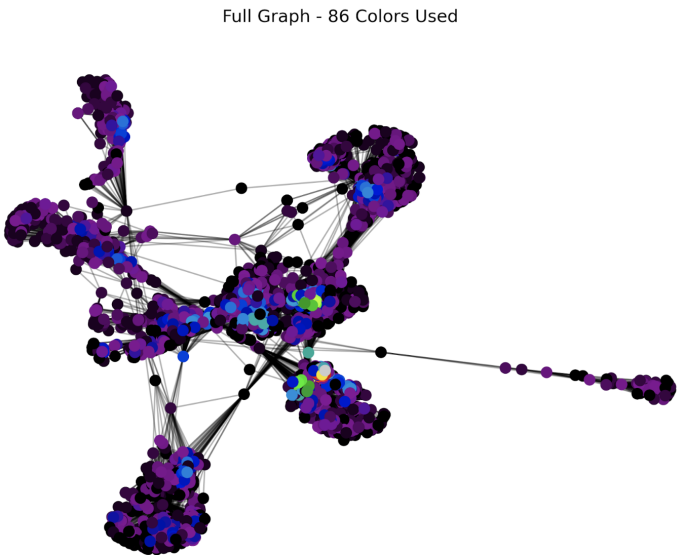

Fig. 4. Coloring Visualization of Subgraph

B. Parallel Graph Coloring (OpenMP)

- Enhancements:
  - Scales efficiently with available cores.
  - Measures speedup and CPU utilization.
  - Logs results to `results_parallel.csv`.
- Algorithm:
  - Phase 1: Tentative parallel coloring.
  - Phase 2: Conflict detection and uncoloring conflicting nodes.
- Techniques:
  - `#pragma omp parallel for` for coloring and conflict detection.
  - Dynamic scheduling for load balancing.
  - Thread control via `OMP_NUM_THREADS`.

TABLE I
Performance Metrics of Sequential and Parallel Graph Coloring

Algorithm	Threads	Total Colors	Execution Time (s)	CPU Time (s)	Mean Color	Color Variance
Sequential	1	276	3.5725	3.5707	44.78	2167.31
Parallel	1	276	3.6137	3.6125	44.78	2167.31
Parallel	2	281	2.0125	4.0210	45.54	2390.02
Parallel	4	283	1.0148	4.0592	45.85	2445.96
Parallel	6	288	0.6867	4.1204	45.83	2468.27
Parallel	8	287	0.5723	4.5780	46.00	2489.67
Parallel	10	290	0.4991	4.9911	45.85	2481.31

VII. Performance Analysis

- Initial linear speedup: Near-linear scaling from 1 to 4 threads.
- Optimal number of threads: 6–8, beyond which overhead dominates.
- Limitations: Synchronization, contention, and load imbalance reduce efficiency beyond 8 threads.

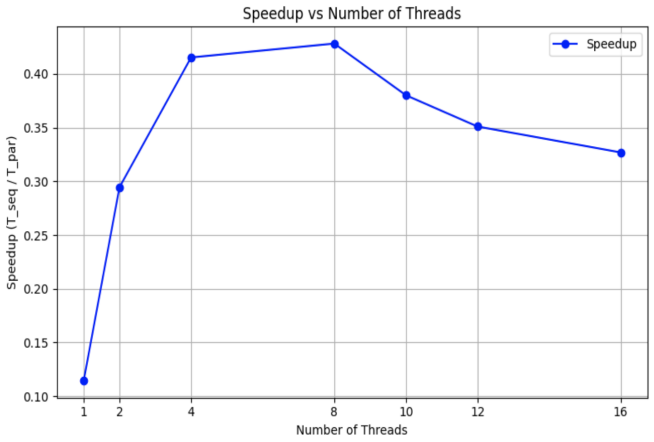

Fig. 5. Speedup vs Threads

VIII. Technical Discussion

A. Scalability

Execution time decreases significantly with thread count (3.61s to 0.49s), but speedup is non-linear due to synchronization overhead and CPU limits.

B. CPU Usage

CPU time increases with threads, confirming effective parallelism.

C. Color Metrics

Sequential and 1-thread parallel coloring produce the same number of colors (276), ensuring baseline correctness. With more threads, color count increases slightly (up to 290), due to independent local decisions and additional conflict resolution.

D. Variance

Mean color rises slightly (44.78 $\rightarrow$ 46.0), and color variance increases with threads, but stabilizes beyond 8 threads.

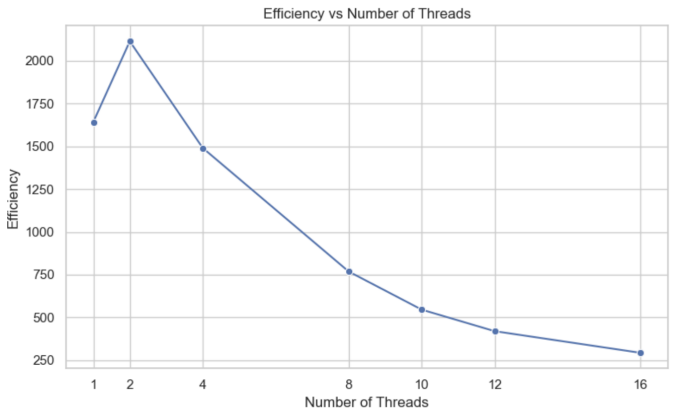

Fig. 6. Color Count vs Threads

IX. Conclusion

- Significant runtime reduction achieved with OpenMP parallel coloring, especially up to 8 threads.
- Trade-off: More colors and higher variance as parallelism increases.
- Scalability: Speedup plateaus due to thread contention and hardware limits.
- Correctness preserved: 1-thread parallel matches sequential results.
- Practical value: Suitable for large social graphs where faster coloring outweighs minimal color use.