A Parallel Graph Coloring Algorithm with Applications in Machine Learning

Maedeh Yahaghi and Saeed Bakhshan

Department of Computer Science, Wayne State University,
Detroit, USA

Abstract. In this paper, we present a parallel algorithm for coloring 3-colorable graphs, based on Wigderson's classical algorithm which guarantees a coloring using at most $O(\sqrt{n})$ colors. To parallelize the approach, we integrate Luby and Gebremedhin-Manne (GM) algorithm for parallel coloring. Experimental evaluations demonstrate that our parallel algorithm achieves speedup to 16 threads.

Keywords: Graph Coloring, Machine Learning, Parallel Algorithm

1 Introduction

Graph coloring is a fundamental problem in theoretical computer science and has numerous practical applications in areas such as scheduling, register allocation, machine learning, frequency assignment, and parallel task execution. The objective is to assign colors to the vertices of a graph such that no two adjacent vertices share the same color, with the aim of minimizing the total number of colors used. Despite its simple definition, graph coloring is computationally challenging. In particular, determining whether a graph can be colored with three colors is an NP-complete problem, and finding the minimum number of colors needed for general graphs is NP-hard.

Due to the intractability of the optimal solution, many heuristic and approximation algorithms have been proposed. Heuristic algorithms are often fast and practical but do not provide guarantees on the quality of the solution [1], i.e., the number of colors used. Approximation algorithms, on the other hand, offer provable upper bounds. For instance, Wigderson's algorithm [2] guarantees that any 3-colorable graph can be colored using at most $O(\sqrt{n})$ colors, where n is the number of vertices. However, this algorithm is inherently sequential and does not leverage modern multi-core architectures.

The increasing scale of graph-based data and the need for faster processing times highlight the importance of parallel graph coloring algorithms. While parallel heuristic methods exist, they lack guarantees on the number of colors used. The proposed parallel algorithm is designed to be efficient and suitable for execution on multi-core systems using OpenMP. The goal is focus on getting speedup by replacing two Luby and GM algorithms. Graph coloring also plays an important role in several machine learning workflows.

Many ML models operate on large computational graphs where tasks, operators, or data blocks exhibit dependencies. Coloring these graphs enables conflict-free scheduling on parallel hardware by assigning independent operations the same color so they can execute simultaneously. In feature selection and dimensionality reduction, graph coloring is used to cluster mutually exclusive or weakly correlated features, allowing models to reduce redundancy and improve generalization. Moreover, in graph-based semi-supervised learning and message-passing neural networks, coloring is applied to partition vertices into independent

David C. Wyld et al. (Eds): CSML, AISCA, DNLP, SOEA, NET, BDHI, SIPO – 2026
pp. 21-30, 2026. CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.160202

sets, enabling synchronous or color-wise parallel updates that accelerate training. These connections highlight the broader relevance of scalable parallel graph coloring algorithms in modern machine learning systems.

2 Related Work

A large amount of related work exists on graph coloring. However, we know of no other work combine Luby and Gebremedhin(GM) algorithm to increase the parallelism.

The classical sequential graph coloring algorithm is based on the greedy first-fit heuristic. Several other heuristics have been proposed that use relatively few colors and have good bounds on their computational complexity. In contrast, parallel algorithms have not been studied as extensively. Nevertheless, there are a few polynomial-time algorithms, some of which can solve the problem using as few colors as the sequential algorithms.

Parallel algorithms improve the time complexity by distributing the vertices among multiple processors. These types of algorithms have two phases: the first phase colors vertices simultaneously, and the second phase resolves any color conflicts in sequential or parallel that require resolution. Luby's *Maximal Independent Set (MIS) algorithm* [3] assigns random weights to vertices and iteratively selects the vertex with the highest weight among its neighbors to form an independent set. Its time complexity is $O(\log n)$ due to hierarchical comparisons. The *Jones-Plassmann algorithm* [4] improves upon Luby's method by assigning unique initial weights, reducing the need for repeated weight generation and achieving a time complexity of $O\left(\frac{\log n}{\log \log n}\right)$. The *Largest-Degree-First (LDF)* [5] and *Smallest-Degree-Last (SDL)* [6] algorithms assign two weights to each vertex based on degree and randomness. Initially, they consider the weight based on degree, and when two vertices have the same degree, they consider a random weight. The LDF algorithm selects the vertex with the largest degree, while the SDL algorithm selects the vertex with the smallest degree. The *Jones-Plassmann Largest-Log-Degree-First (JP-LLF)* [7] and *Jones-Plassmann Smallest-Log-Degree-Last (JP-SLL)* [7] algorithms are similar to LDF and SDL, with the difference being that the degree is determined based on a logarithmic scale. The *Efficient Coloring with Local Information (ECL-GC) algorithm* [8] improves parallel execution by introducing shortcuts that increase independence among vertices during coloring. The *Distributed Graph Coloring (DistG) algorithm* [9] employs a distributed memory approach, where each vertex computes its color based on local information, with conflicts resolved through iterative updates. The *Asynchronous parallel graph coloring heuristics algorithm* [4] divide the graph into global and local vertices, resolving conflicts via asynchronous communication for scalable execution. *SeqSolve* [10], *IterSolve* [10–13], *IterSolveR* [11], *ColorTM* [14], *Color-Centric (CLU)* [15], *Vertex-Centric (VFF)* [16], and *Recoloring* [16] algorithms follow the two phases of parallel graph coloring using different approaches. Additionally, *hybrid* [17], *Gebremedhin and Manne (GM)* [13] and *Bal-ColorTM* algorithms, function similarly to *SeqSolve*, *IterSolve*, *IterSolveR*, *ColorTM*, *Color-Centric (CLU)*, *Vertex-Centric (VFF)*, and *Recoloring*, with one difference: they consider load balancing. In graph coloring, this means ensuring that each color class contains roughly the same number of vertices.

In this paper We propose a parallel graph coloring algorithm based on Wigderson's approach. The algorithm first computes the degree of all vertices in parallel. It then selects vertices whose degree is greater than or equal to $\sqrt{n}$, along with their neighbors, and places them into a set. On this set, we apply Luby's algorithm: each vertex is assigned a random value, and in each iteration, a vertex is colored if its value is the highest among its neighbors. This process continues until all vertices in the set are colored.

For the remaining vertices—those with degree less than $\sqrt{n}$ and not adjacent to the selected set—we apply the Gebremedhin–Manne (GM) algorithm. This algorithm works in two phases: in parallel, each vertex marks the colors used by its neighbors as forbidden, then selects the smallest available color not in the forbidden set. Since this step may result in adjacent vertices receiving the same color, we identify such conflicts and resolve them by recoloring the vertex with the smaller ID. The recoloring step is repeated until all conflicts are resolved and all vertices are properly colored.

3 Problem Statement

Graph coloring is a classical computational problem where the goal is to assign colors to the vertices of a graph such that no two adjacent vertices share the same color, while minimizing the total number of colors used. However, finding the optimal coloring is NP-hard, even for special cases such as 3-colorable graphs. As a result, heuristic and approximation algorithms are widely used. Heuristic methods, while efficient, do not offer guarantees on the number of colors used and often sacrifice optimality for speed. On the other hand, approximation algorithms provide theoretical guarantees. For instance, Wigderson's algorithm ensures that any 3-colorable graph can be colored using at most $O(\sqrt{n})$ colors, but it is sequential and not well-suited for parallel execution.

With the growing size of real-world graphs and the increasing availability of multicore computing platforms, there is a pressing need for parallel graph coloring algorithms. Most existing parallel approaches rely on greedy or heuristic techniques and lack provable bounds on the number of colors used.

This paper aims to address this gap by designing and analyzing a parallel graph coloring algorithm inspired by Wigderson's method that is approximation algorithm. The goal of this paper is to achieve better speedup that we mentioned.

4 Contribution

In this paper we design a new parallel algorithm based on Wigderson's method. The algorithm begins by identifying high-degree vertices and their neighbors in parallel and then applies a maximal independent set (MIS)-based strategy to color this subset using a small number of colors.

Next, we extend the parallelization to the remaining vertices using the Gebremedhin-Manne (GM) [13] algorithm, a parallel greedy coloring approach that minimizes conflicts while ensuring that the color usage remains within the desired bound. Our approach effectively mimics the behavior of the sequential Wigderson algorithm in a parallel setting.

Finally, we implement the parallel algorithm in C using OpenMP and evaluate its performance through experiments. Our results shows that we achieves speedup to 16 threads.

5 Sequential 3-Colorable Wigderson Algorithm

Wigderson's algorithm is a classical approximation algorithm for coloring 3-colorable graphs. It works by iteratively identifying vertices with high degree and coloring their neighborhoods using only two colors, then removing those neighborhoods and repeating the process. Once all high-degree vertices have been processed, the remaining graph has bounded degree and can be colored greedily [18]. The following pseudocode outlines the sequential version of this algorithm.

Algorithm 1: Wigderson's Algorithm [2]

Input: 3-chromatic graph G **Initialize:** Let $c = 1$
While $\Delta(G) \geq \lceil \sqrt{n} \rceil$ do:
  Choose a vertex $v \in G$ with $\deg(v) \geq \lceil \sqrt{n} \rceil$
  Use Bipartition Algorithm to 2-color $G[N(v)]$ with colors c and $c+1$
  Remove $N(v)$ from G
  $c \leftarrow c + \chi(G[N(v)])$
End While
If $\Delta(G) < \lceil \sqrt{n} \rceil$ then:
  Use Greedy Algorithm to color the remaining vertices with colors $c, c+1, \ldots, c+\lceil \sqrt{n} \rceil$

6 Parllel 3-Colorable Wigderson Algorithm

To overcome the sequential nature of Wigderson's original algorithm, we design a parallel version that follow sequential behaviour.
The parallel algorithm is structured in two main phases. In the first phase, we identify all high-degree vertices—those with degree at least $\sqrt{n}$—and include their neighbors to form a subgraph.
This subgraph is then colored using a parallel maximal independent set (MIS)-based strategy to ensure conflict-free coloring within a bounded number of colors.

Algorithm 2: Parallel Graph Coloring

Input: A 3-colorable graph $G = (V, E)$
Output: A proper coloring using at most $O(\sqrt{n})$ colors
1. procedure ParallelGraphColoring($G = (V, E)$)
2. **Compute Degrees in Parallel:**
3. for all $v \in V$ in parallel do
4. $\deg(v) \leftarrow |N(v)|$
5. end for
6. **Identify High-Degree Vertices and Their Neighbors:**
7. for all $v \in V$ in parallel do
8. $S \leftarrow \{v \in V \mid \deg(v) \geq \sqrt{n}\}$
9. $S \leftarrow S \cup \{u \mid \exists v \in S, u \in N(v)\}$ *Include neighbors of S*
10. end for
11. **Color S using Maximal Independent Set (MIS):**
12. $S' \leftarrow S$
13. while ($|S'| > 0$) do in parallel
14. Choose an independent set I from S'
15. Color all vertices in I with smallest available color
16. $S' \leftarrow S' - I$
17. end while
18. **Color Remaining Vertices using GM Algorithm:**
19. $W \leftarrow V - S$
20. while $W \neq \emptyset$ do
21. for each vertex $v \in W$ in parallel do
22. for each vertex $w \in \text{adj}(v)$ do
23. forbiddenColors[color[w]] $\leftarrow v$
24. end for
25. $c \leftarrow \min\{0 < i \leq 1 + \sqrt{n} \mid \text{forbiddenColors}[i] \neq v\}$
26. color[v] $\leftarrow c$
27. end for
28. $R \leftarrow \emptyset$ *Initialize the remaining worklist*
29. for each vertex $v \in V$ in parallel do
30. for each vertex $w \in \text{adj}(v)$ do
31. if color[v] = color[w] and $v < w$ then
32. $R \leftarrow R \cup \{v\}$

```
33.                end if
34.              end for
35.            end for
36.            W ← R                                   Update the worklist
37.          end while
38. end procedure
```

In the second phase, we use the Gebremedhin-Manne (GM) algorithm to color the remaining vertices. The GM algorithm allows vertices to choose colors in parallel based on their neighbors' current assignments and includes a conflict-resolution mechanism to ensure correctness.

7 Data Generation

To evaluate the performance of our parallel graph coloring algorithm, we generated synthetic graphs that are guaranteed to be 3-colorable. Specifically, we constructed tripartite graphs, which are a subclass of 3-colorable graphs. The vertex set is partitioned into three disjoint subsets: V_1, V_2, and V_3 that the graph size is $n \in \{5000, 10000, 15000\}$ that 5000,10000,15000 show the number of vertices in each partition and the edge density configurations be: configs = { (sparse, 0.3), (mid, 0.6), (dense, 0.8) }

Edges are only created between vertices from different subsets, and no edges exist within the same subset.We generated sparse and mid and dense graph with this configuration.

The generated graphs are written to a file in the following format:

- The first line contains the number of vertices and the number of edges.
- Each subsequent line contains two integers u and v, representing an undirected edge between vertex u and vertex v.

This approach ensures the generated graphs are 3-colorable by construction, making them suitable for testing both correctness and performance of our algorithm.

8 Solution and Implementation

The main goal of this paper is to getting speedup for graph coloring. So, we mimic the sequential approximation algorithm (Wigderson) behavior.

The sequential algorithm consists of two steps:

1. **Step 1:** find vertices with degree bigger and equal $\lceil\sqrt{n}\rceil$ and then color their neighbors with colors c and $c+1$. This step guarantees that the subgraph of a 3-colorable graph can be colored using at most two colors. To parallelize this approach, all vertices first compute their degree in parallel. Then, those vertices and their neighbors form a set. For coloring the set we used Luby algorithm that assigned the random number to vertices and color vertices that their value is bigger than their neighbor with same color and then apply this approach for remaining vertices in the set until all vertices in set be colored. **Step 2:** Color the remaining vertices, which have a degree less than $\lceil\sqrt{n}\rceil$, using a greedy algorithm. Since we aim to mimic the sequential algorithm, we propose using the Gebremedhin-Manne (GM) algorithm. GM algorithm, all vertices first check their neighbors' colors in parallel and then put them as forbidden color for that vertex then assign the minimum color to vertices while are not in the forbidden color. It is possible that we have color conflict so we select vertices with minimum ID that are in color conflic for recoloring until we do not have color conflict. The GM algorithm is shown in Algorithm 2.

9 Experimental Result

To evaluate the performance and scalability of the proposed parallel algorithm, we conducted experiments on synthetic tripartite graphs of sizes 15000, 30000, and 45000 vertices under three edge-density configurations: sparse (0.3), mid-density (0.6), and dense (0.8). All experiments were executed using OpenMP, and speedup was measured relative to the sequential baseline implementation that follows Wigderson's original structure.

The results across all graph classes demonstrate a consistent upward trend in parallel scalability as the number of threads increases, confirming the effectiveness of replacing Wigderson's sequential coloring phases with Luby's MIS-based parallel phase and the parallel GM greedy recoloring phase.

For dense graphs (Figure 1), the algorithm exhibits the highest speedup among all graph categories. Dense instances contain a large number of high-degree vertices, causing a significant portion of the workload to be shifted into the Luby MIS stage, which is highly parallelizable. As a result, the speedup grows nearly linearly up to 8 threads and continues to improve, though with diminishing returns, up to 16 threads. For the 45000-vertex dense graph, the speedup approaches the upper bound predicted by Amdahl's Law given the proportion of parallel work.

For mid-density graphs (Figure 2), the parallel performance also shows strong improvement, but with slightly lower speedup than in the dense case. This behavior aligns with the algorithmic design: mid-density graphs have fewer high-degree vertices meeting the $\sqrt{n}$ threshold, which reduces the size of the Luby-colored set. Consequently, a larger portion of the work transitions to the GM phase, which, although parallel, involves repeated conflict-resolution rounds that limit ideal scalability. Nonetheless, the algorithm consistently achieves meaningful speedup across all graph sizes, and the 45000-vertex instance reaches near-linear scalability up to 8 threads.

In sparse graphs (Figure 3), speedup is still present but more moderate. These graphs contain the fewest high-degree vertices, causing most of the workload to fall into the GM phase. The GM algorithm relies heavily on per-vertex color checking and conflict resolution, which introduces overhead due to the lower level of parallel independence across vertices. As expected, this reduces opportunities for massive parallelism compared to dense graphs. However, even in the sparse case, the algorithm achieves speedup that increases steadily with both graph size and thread count, confirming that parallelization remains beneficial even when the structure is less conducive to MIS-based coloring.

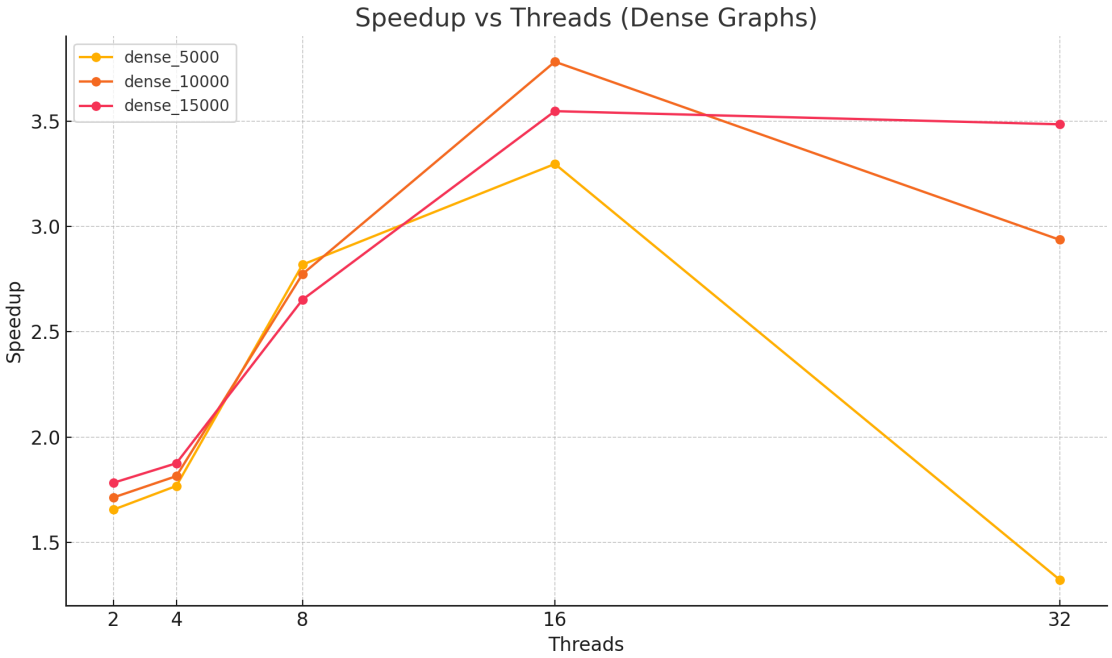

Fig. 1: speedup for Dense Graph with 15000,30000,45000 Vertices

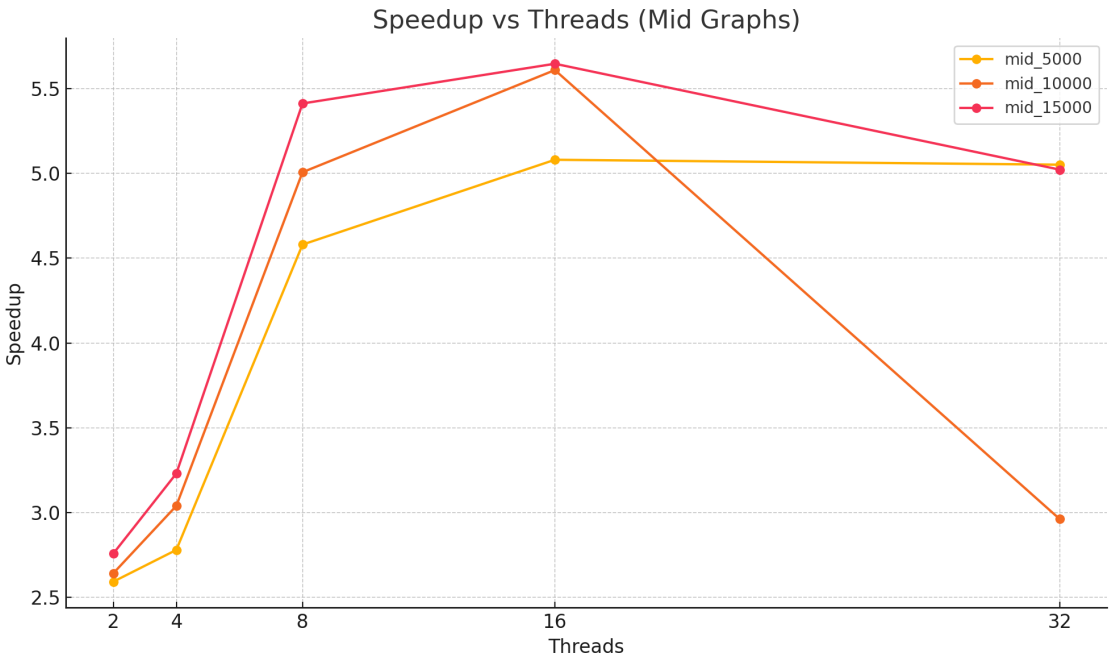

Fig. 2: speedup for Mid Graph with 15000,30000,45000 Vertices

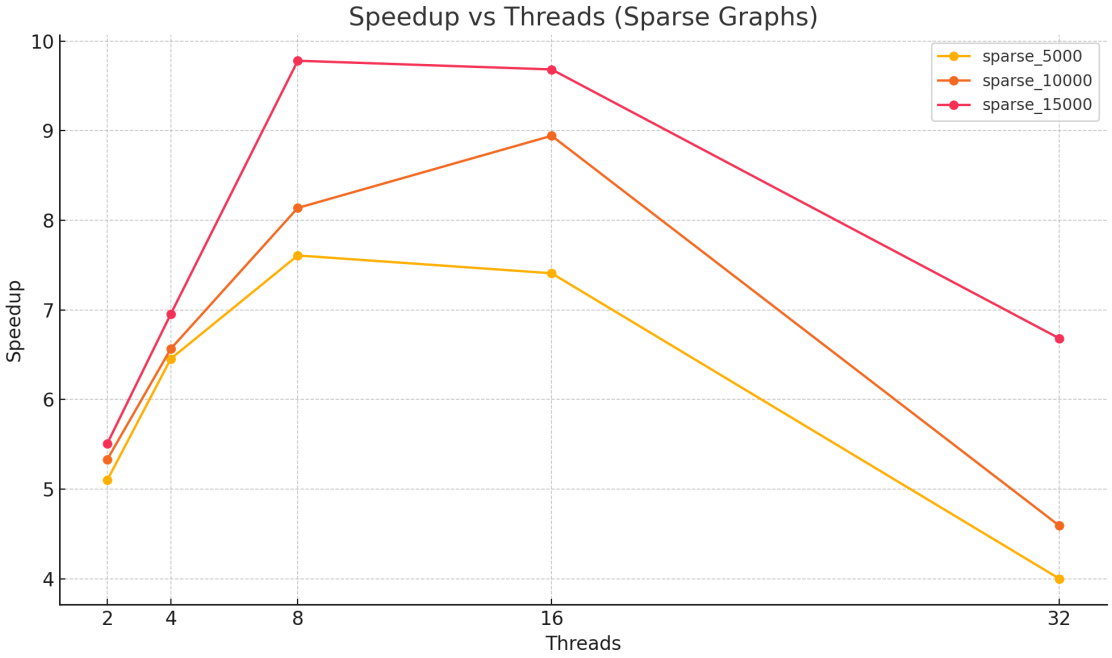

Fig. 3: speedup for Sparse Graph with 15000,30000,45000 Vertices

10 Limitations of the Research

Although the proposed parallel algorithm provides substantial parallel speedup, but some limitations remain.

First, the algorithm assumes that the input graph is 3-colorable, which restricts its applicability in settings where no such structural guarantee exists. Real-world graphs often do not conform to this constraint, and the performance or correctness of the method may vary when applied outside this class.

Second, the efficiency of the MIS-based parallel phase depends heavily on the presence of high-degree vertices. In graphs where few vertices satisfy the $\sqrt{n}$ degree threshold—especially sparse or irregular graphs—the amount of exploitable parallelism decreases, limiting speedup.

Third, the GM recoloring phase may require multiple rounds of conflict resolution. This iterative process can introduce significant overhead in graphs that generate many color conflicts, reducing theoretical scalability.

Finally, our evaluation is based on OpenMP (shared-memory method). The behavior of the algorithm on GPUs, many-core accelerators, or distributed-memory environments has not yet been explored. Investigating these architectures represents an important direction for future work.

11 Conclusion

In this work, we developed a parallel algorithm by extending Wigderson's classical approach for coloring 3-colorable graphs. While the original algorithm is sequential, our parallel version achieves substantial speedup by integrating Luby's MIS procedure for high-degree neighborhoods and the Gebremedhin–Manne (GM) algorithm for parallel greedy coloring. The combined design preserves the structural behavior of the sequential approximation algorithm while enabling scalable execution on modern multi-core architectures.

Through implementation in C with OpenMP and experimental evaluation on multiple graph sizes and densities, we showed that the algorithm scales effectively up to 16 threads, with dense graphs exhibiting the highest parallel efficiency. These results confirm that parallelizing the core components of Wigderson's method can yield significant performance gains without compromising the approximation properties of the original algorithm.

Beyond classical graph-theoretic applications, the relevance of scalable parallel graph coloring extends directly to modern machine learning systems. Many ML workloads rely on large computational graphs, message-passing neural networks, or synchronization-sensitive update rules, all of which benefit from partitioning vertices into independent sets. Efficient colorings enable conflict-free parallel execution of operators, accelerate mini-batch propagation in graph neural networks, and support structure-aware feature grouping for dimensionality reduction and model sparsification. As machine learning models continue to increase in structural complexity, fast parallel graph coloring becomes an increasingly important primitive.

Overall, this work contributes both a practical parallel algorithm and a pathway toward enabling high-performance graph coloring in machine-learning pipelines and other large-scale data-driven systems. Future work may explore integrating the algorithm into ML runtimes, extending the parallelization strategies to k-colorable graphs, or adapting the method for GPU and distributed-memory environments.

References

1. M. R. Garey and D. S. Johnson, *Computers and Intractability.* New York: W. H. Freeman and Company, 2002, vol. 29.
2. A. Wigderson, "Improving the performance guarantee for approximate graph coloring," *J. ACM*, vol. 30, no. 4, pp. 729–735, 1983. [Online]. Available: https://doi.org/10.1145/2157.2158
3. M. Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM J. Comput.*, vol. 15, pp. 1036–1053, 1985. [Online]. Available: https://api.semanticscholar.org/CorpusID:1126858
4. M. T. Jones and P. E. Plassmann, "A parallel graph coloring heuristic," *SIAM J. Sci. Comput.*, vol. 14, no. 3, p. 654–669, May 1993. [Online]. Available: https://doi.org/10.1137/0914041
5. D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems." *The Computer Journal*, vol. 10, pp. 85–86, 1967.
6. D. W. Matula, G. Marble, and J. D. Isaacson, "Graph coloring algorithms," in *Graph Theory and Computing*, R. C. Read, Ed. New York: Academic Press, 1972, pp. 109–122.
7. W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring," *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pp. 50–59, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:2139284

8. G. Alabandi, E. Powers, and M. Burtscher, "Increasing the parallelism of graph coloring via shortcutting," *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 262–275, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:209896447
9. A. Brighen, H. Slimani, A. Rezgui, and H. Kheddouci, "A new distributed graph coloring algorithm for large graphs," *Cluster Computing*, vol. 27, pp. 875–891, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257727769
10. A. H. Gebremedhin and F. Manne, "scalable parallel graph coloring algorithms," *Euro-Par 2005 Parallel Processing*, vol. 12, p. 1131–1146, 2000.
11. G. Rokos, G. Gorman, and P. H. J. Kelly, "A fast and scalable graph coloring algorithm for multi-core and many-core architectures," in *European Conference on Parallel Processing*, 2015, pp. 414–425. [Online]. Available: https://api.semanticscholar.org/CorpusID:17288257
12. E. G. Boman, D. Bozdağ, U. Catalyurek, A. H. Gebremedhin, and F. Manne, "A scalable parallel graph coloring algorithm for distributed memory computers," in *Euro-Par 2005 Parallel Processing*, J. C. Cunha and P. D. Medeiros, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 241–251.
13. U. V. Çatalyürek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen, "Graph coloring algorithms for multi-core and massively multithreaded architectures," *Parallel Computing*, vol. 38, no. 10-11, pp. 576–594, October 2012.
14. C. Giannoula, G. I. Goumas, and N. Koziris, "Combining htm with rcu to speed up graph coloring on multicore platforms," in *Information Security Conference*, 2018, pp. 350–365. [Online]. Available: https://api.semanticscholar.org/CorpusID:46977632
15. C. Giannoula, A. Peppas, G. Goumas, and et al., "High-performance and balanced parallel graph coloring on multicore platforms." *J Supercomput*, vol. 79, p. 6373–6421, 2023.
16. H. Lu, M. Halappanavar, D. Chavarría-Miranda, A. Gebremedhin, and A. Kalyanaraman, "Balanced coloring for parallel computing applications," *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 7–16, 2015.
17. S. Che, G. Rodgers, B. Beckmann, and S. Reinhardt, "Graph coloring on the gpu and some techniques to improve load imbalance," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, 2015, pp. 610–617.
18. R. M. R. Lewis, *Guide to Graph Colouring: Algorithms and Applications*, second edition ed. Springer, 2023.