**CENG577 Parallel Computing**

**Project Proposal: Scalable Parallel Graph Coloring Algorithms**

**İmre KOSDİK, Yağızcan PANÇAK**

**1 Abstract**

Graph coloring is the problem of assigning labels, known as colors, to the vertices of a graph so that no two adjacent vertices share the same label, using the minimum number of colors. In the context of parallel applications, graph coloring enables identification of independent tasks that can be performed simultaneously. The focus of this work is to implement the second parallel graph coloring algorithm described in [1], which is designed to improve the efficiency of the coloring process, particularly in parallel computing environments. Furthermore, this implementation will be evaluated by testing it on a variety of sparse graphs. Investigating the performance of the algorithm on these types of graphs provides valuable insights into its efficiency and scalability. This evaluation will help determine the practical applicability of the algorithm in solving real-life graph coloring problems.

**2 Introduction**

Graph coloring is a technique used in graph theory to assign labels, or "colors," to the vertices of a graph in such a way that no two adjacent vertices have the same color. The objective is to use the smallest number of colors possible. Graph coloring serves as a basis for solving scheduling and team building problems [3], identifying short circuits in printed circuit designs, a preprocessing step to speed up the computation of Jacobian and Hessian matrices.[2] The lowest number of colors that solves the graph coloring problem is called the chromatic number of a graph. Determining the chromatic number of a graph is an NP-Hard problem. As a consequence, one relies on heuristic methods to solve such a problem. The algorithms described in section 3 in [1] approach the problem with greedy methods and offer almost a linear speedup. The algorithms utilize block partitioning, dividing the vertex set of a graph into equal-sized blocks. The second algorithm is the modification of the first algorithm and uses fewer colors to color the graph. With this project, we aim to implement the second algorithm and investigate the performance of our implementation over different sets of sparse graphs.

**3 Motivation**

The motivation for this project stems from the unique challenges of graph coloring in parallel processing environments. Graph coloring is inherently complex due to its NP-hard nature, making it computationally expensive to find the optimal solution, particularly for large graphs that mirror real-world systems. Sparse graphs, commonly used in various applications such as social network analysis, circuit design, and scientific computations, pose additional challenges as they often have irregular structures that complicate partitioning and parallelization. Effective parallel graph coloring requires balancing the accuracy of the solution (in terms of minimizing colors) with the efficiency of the algorithm (in terms of time and computational resources).

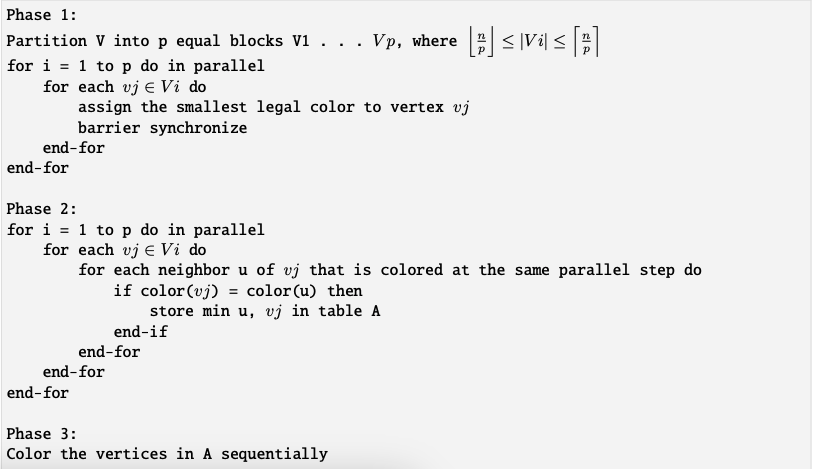
Gebremedhin and Manne’s work on scalable parallel graph coloring algorithms provides a promising approach to addressing these challenges. Their second algorithm builds upon the first by incorporating refinements that reduce the number of colors needed, potentially leading to more efficient use of computational resources. Implementing and testing this algorithm on a variety of sparse graphs will help reveal its practical benefits and limitations, as well as provide insights into how well it scales with the graph size and density. Additionally, the project aims to uncover how variations in graph structure impact the performance and effectiveness of the algorithm, contributing to the broader field of scalable graph algorithms.

**4 Background and Related Work**

In parallel computing, efficient graph coloring can help identify independent tasks, enabling them to execute concurrently without conflict. Various algorithms have been developed to achieve this in parallel environments, [1] is particularly notable. One primary reason for choosing Gebremedhin and Manne’s second algorithm over other existing methods is its efficient color minimization. Many traditional algorithms, like basic greedy methods, do not prioritize reducing color counts in parallel settings, which can lead to higher computational overhead and resource consumption. In contrast, the second algorithm refines the initial greedy approach, using fewer colors by reducing conflicts that arise from block partition boundaries. This makes it particularly suitable for sparse graphs, common in real-world scenarios like social networks and scientific computations, where minimizing colors directly impacts performance and computational resource demands. Their research proposes scalable parallel graph coloring algorithms with a focus on block partitioning, where the vertex set of the graph is divided into equal-sized blocks to facilitate parallel computation. Gebremedhin and Manne introduced two algorithms that utilize this partitioning technique, achieving significant speedups by processing these blocks concurrently. The first algorithm assigns colors to vertices in a greedy manner, while the second is a refinement that requires fewer colors, improving the efficiency of the coloring process. These methods demonstrate near-linear scalability, making them suitable for large and sparse graphs encountered in real-world applications. This project builds upon their work by implementing the second algorithm to explore its applicability and performance across various sparse graph structures.

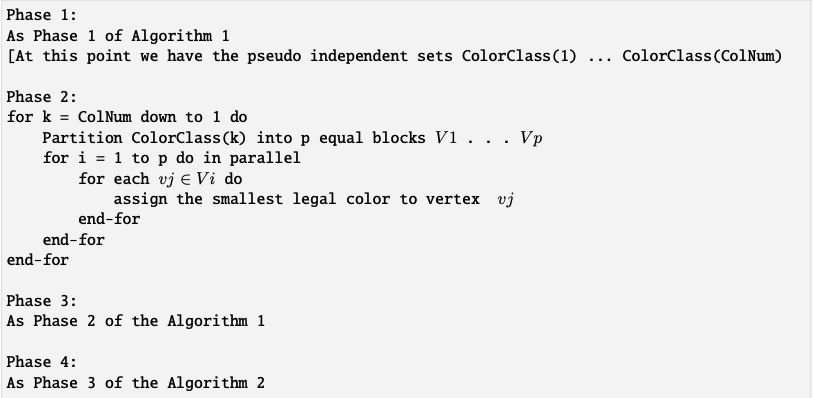
**4.1 Block Partition Based Graph Coloring Algorithm**

The algorithm consists of three phases. In the first phase, we partition the vertex set into p equal-sized blocks, where p represents the number of processors. The vertices within each block are colored in parallel, with each parallel step concluding with a barrier synchronization. The color assigned to a vertex is determined by the colors of its previously colored neighbors. These neighbors can belong either to the same block as the vertex or to a different block, which allows adjacent vertices in different blocks to be assigned the same color. This coloring approach is referred to as pseudo-coloring. In the second phase, we check in parallel whether each vertex has a valid coloring by comparing its color with those of its neighbors that were colored during the same parallel step in the first phase. If an invalid coloring is detected, we store one of the vertices of the conflicting edge in a table. In the final phase, we sequentially re-color each vertex listed in the table, resolving any conflicts. The pseudocode of the algorithm is below.

****

**4.2 Improved Block Partition Based Graph Coloring Algorithm**

The algorithm builds upon the method described in Section 4.1 and leverages the results of Culberson’s Iterated Greedy coloring heuristic. The first phase of this algorithm is identical to the first phase in Section 4.1. Additionally, we store the variable ColNum as the coloring used in this phase. At the end of the first phase, we have a ColNum number of pseudo-independent sets, meaning that any edge within a color class is a result of a "conflict" edge from Phase 1. In the second phase, we use the pseudo-colorings obtained in the first phase to reverse the color class ordering of the vertices. In pp parallel steps, we color the vertices in each color class in the same manner as in the first phase. The remaining third and fourth phases are identical to the second and third steps of the algorithm described in Section 4.1, respectively. The pseudocode of the algorithm is below.

****

**5 Methodology**

The primary focus of this project is to implement the second parallel graph coloring algorithm described by Gebremedhin and Manne and to investigate its performance on different sparse graphs. The motivation for this project stems from the need for efficient solutions in parallel processing environments, where identifying independent tasks is crucial. Graph coloring inherently represents a complex challenge due to the NP-hard nature of determining the chromatic number. Hence, heuristic and approximate approaches, like the one presented in Gebremedhin and Manne’s work, are necessary for practical applications. The project begins with implementing the algorithm in a parallelized environment. The selected algorithm employs block partitioning, a technique that divides the graph's vertices into equally sized blocks. Each block is processed in parallel, with an initial greedy coloring step. Subsequently, conflicts at the boundaries of these blocks are resolved to ensure no two adjacent vertices share the same color. This partitioned and parallel approach leverages the computational efficiency of concurrent processing while aiming to minimize the overall color count. To assess the effectiveness of the implementation, the algorithm will be tested on a range of sparse graph datasets, each representing different real-world scenarios. Performance metrics, such as execution time, speedup ratio, and the number of colors used, will be recorded. This analysis will provide insights into how well the algorithm scales with graph size and sparsity, and whether the refinement in color minimization (over the first algorithm) translates into measurable gains. By examining these factors, the project aims to evaluate the trade-offs between computational efficiency and color minimization, contributing to a deeper understanding of the algorithm's practical benefits and limitations in parallel graph coloring applications.

**5 References**

**[1]** Gebremedhin, Assefaw Hadish, and Fredrik Manne. "Scalable parallel graph coloring algorithms." Concurrency: Practice and Experience 12.12 (2000): 1131-1146

**[2]** Satish Thadani, Seema Bagora, Anand Sharma, “Applications of graph coloring in various fields”, Materials Today: Proceedings, Volume 66, Part 8, 2022, Pages 3498-3501, ISSN 2214-7853

**[3]** Ian Bogle, George M. Slota, Erik G. Boman, Karen D. Devine, Sivasankaran Rajamanickam, Parallel graph coloring algorithms for distributed GPU environments, Parallel Computing, Volume 110, 2022, 102896, ISSN 0167-8191