國立臺灣大學重點科技研究學院積體電路設計與自動化學位學程

碩士論文

Program in Integrated Circuit Design and Automation

Graduate School of Advanced Technology

National Taiwan University

Master's Thesis

高品質分散式圖著色演算法

Distributed High-Quality Graph Coloring Algorithm on GPUs

吳建賢

Chien-Hsien Wu

指導教授: 郭斯彥 博士

Advisor: Sy-Yen Kuo Ph.D.

中華民國 114 年 7 月

July, 2025

doi:10.6342/NTU202502064

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

高品質分散式圖著色演算法

Distributed High-Quality Graph Coloring Algorithm on GPUs

本論文係吳建賢 R12K41022 在國立臺灣大學 重點科技研究學院積體電路設計與自動化碩士學位學程完成之碩士學位論文，於民國 114 年 7 月 21 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Program for 7/21/2025 on have examined a Master's thesis entitled above presented by CHIEN-HSIEN,WU R12K41022 candidate and hereby certify that it is worthy of acceptance..

口試委員 Oral examination committee:

郭斯彥 （指導教授 Advisor）　　張耀文

陳俊良　　[illegible]　　雷欽隆

主任 Director: [illegible]

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

Acknowledgements

感謝郭斯彥教授在這兩年間的悉心指導，無論在學術研究上，或是做人處世方面，都給予我莫大的啟發與幫助。亦感謝宙穎學長在論文上的指導，讓我能順利完成研究工作。同時，也謝謝 HPC team 的所有成員，感謝大家一路以來的支持與協助。

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

摘要

圖著色（Graph Coloring）是一種基礎的演算法，目的是將顏色分配給圖形的節點，且相鄰節點不能擁有相同的顏色。在平行圖著色演算法中，主要評估兩個參數：顏色的品質與執行速度。我們提出的優化包含兩個部分：

首先，我們重新設計 JP-SL 的優先級分配流程，以減少全域頂點掃描的次數並在執行過程中更有效地利用平行化。其次，我們將大規模圖劃分為多個子圖，並將其分配至多個 GPU 上同時運行，以進一步提升整體執行效率。

關鍵字：圖著色、圖演算法、平行運算、平行應用、演算法設計與分析

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

Abstract

Graph coloring is a fundamental problem involving the assignment of colors to the vertices of a graph, ensuring that no two adjacent vertices possess the same color. In parallel graph coloring, two primary parameters are evaluated: color quality and execution speed. Our optimization methodology has two parts. We first improve priority allocation by re-engineering JP-SL to minimize global vertex scans and optimize parallelism during execution. Secondly, we enhance the overall runtime by splitting the extensive graph into subgraphs and allocating them across many GPUs for concurrent execution.

Keywords: graph coloring, graph algorithms, parallel computing, parallel applications, algorithm design and analysis.

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

Contents

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

List of Figures

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

List of Tables

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

Chapter 1 INTRODUCTION

Graph coloring is a fundamental issue in graph theory that entails giving colors to the vertices of a graph $G(V, E)$ in a manner that ensures no two adjacent vertices possess the same color. The main goal is to reduce the quantity of colors employed while adhering to this constraint. Graph coloring has broad applications in both theoretical and practical research, significantly impacting areas such as resource allocation, scheduling, network optimization, clustering, data mining, image processing, and image segmentation, where effective partitioning of data or images is essential for performance. Moreover, graph coloring methods are essential for network resource allocation and process scheduling, where it is necessary to reduce conflicts by assigning unique resources to conflicting activities or users. Moreover, these techniques are utilized in sparse Jacobian matrix optimization [1] and LU decomposition [2], which are crucial for numerical analysis and solving extensive systems of linear equations in scientific computing.

Presently, prevalent GPU techniques for graph coloring can be generally classified into two methodologies:

1. Speculative Greedy (SGR) Scheme-Based Methods [3]. Picasso [4] is a significant algorithm in this area, distinguished by its allowance for conflicts between vertices during the coloring process. The benefit of such algorithms is that they typ-

doi:10.6342/NTU202502064

ically necessitate reduced computational time. Nonetheless, this incurs the expense of employing a marginally greater quantity of colors in comparison to MIS-based methods.

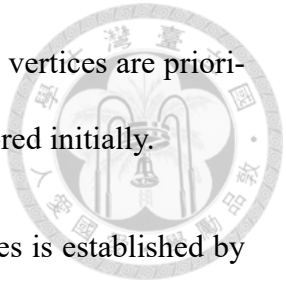
2. Maximal Independent Set-Based Methods (MIS)[5]. MIS methodologies are essential in the advancement of graph coloring algorithms. The Jones and Plassmann (JP) algorithm [6] is a notable algorithm in this field, forming the foundation for several future optimizations. Recently, ECL-GC [7] has been established as a data-driven application of the Jones-Plassmann Algorithm. This method markedly improves execution speed, exceeding that of Speculative Greedy (SGR) techniques, while preserving the benefit of utilizing fewer colors characteristic of MIS-Based Algorithms.

This study examines the comparison of MIS-based methodologies, with particular emphasis on the well-known Jones and Plassmann (JP) algorithm [6]. The principal rationale for focusing on MIS Algorithms is their ability to attain superior coloring quality with a reduced number of colors, resulting in their extensive utilization. Furthermore, recent improvements have improved the efficiency of MIS Algorithms, rendering their execution speed equivalent to, or even exceeding, that of Speculative Greedy (SGR) approaches. MIS-based systems effectively utilize parallelism to identify independent sets, yielding solutions that require fewer colors and enhance the overall efficiency of graph coloring.

The paper by [8] presents various prominent JP Algorithms, emphasizing the differences in vertex prioritization methodologies. The following is an introduction to various JP Algorithms:

- **Largest Degree First (LF) [9]**: Vertices with a greater degree are accorded higher

doi:10.6342/NTU202502064



priority, indicating that vertices with a larger number of nearby vertices are prioritized. In the coloring step, vertices with higher priority are colored initially.

- **Smallest Degree Last (SL)[10][11]**: The precedence of vertices is established by the repetitive elimination of the vertex with the minimal degree from the graph. The residual subgraph is colored utilizing the Largest Degree First heuristic, followed by the incremental reinsertion and coloring of the deleted vertices. This method prioritizes the vertices eliminated last, and during the coloring step, those with higher priority are colored first.

- **First-Fit (FF)[9]**: Colors the vertices in the order they appear in the vertex set, without any additional prioritization.

- **Random Ordering (R)[6]**: Randomly assigns an order to the vertices and colors them according to this random sequence.

- **Saturation Degree (SD)[11]**: Prioritizes vertices that have the most neighbors already assigned unique colors, using vertex degree as a tiebreaker.

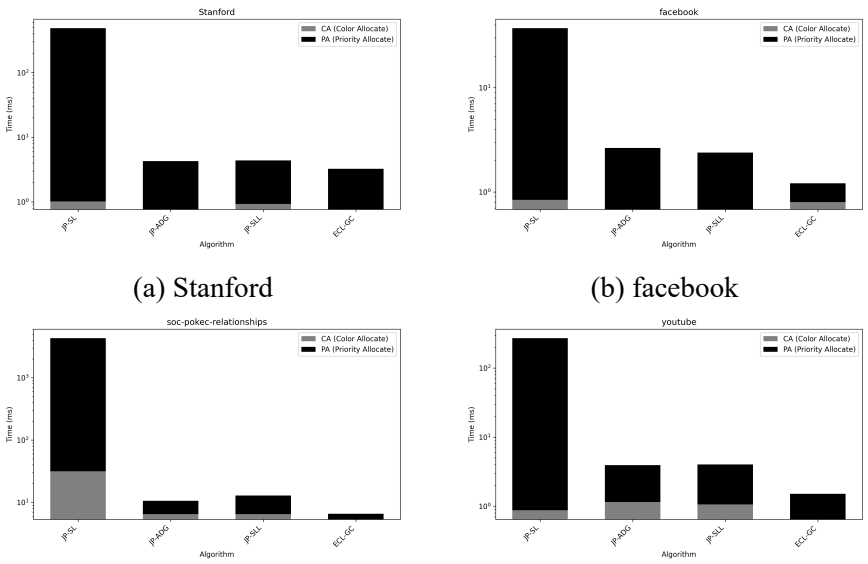
Motivation

Fig. 1.1 illustrates that JP-SL demonstrates considerable execution time during the Priority Allocation (PA) phase across all datasets, with this phase constituting a significant fraction of the overall coloring process. This unequivocally demonstrates that PA has emerged as the principal performance bottleneck of the algorithm in real-world applications.

To address this issue, the present study aims to accelerate the computation in this

doi:10.6342/NTU202502064

phase while simultaneously exploring feasible strategies to enhance the final coloring quality.



(a) Stanford (b) facebook

(c) soc-pokec-relationships (d) youtube

Fig. 1.1: Runtime of CA & PA in four datasets

Contribution

This study's primary contributions are classified into two domains: the optimization and design of Priority Allocation (PA), and large-scale Graph Coloring (GC) in distributed systems.

1.0.1 Priority Allocation Optimization

1.0.1.1 Resilient Smallest Degree Last (R-SL) Architecture

In traditional JP-SL, the Priority Allocation (PA) phase requires multiple global scans over all vertices. In contrast, our proposed modification completes the PA phase with only a single scan, achieving approximately a six-fold improvement in execution speed.

doi:10.6342/NTU202502064

1.0.1.2 Worklist-Based Optimization

To improve GPU utilization during priority assignment, we propose an efficient *worklist-based parallelization* technique that enables efficient warp-level removal of low-degree vertices.

1.0.1.3 Fuzzy Parameter to Reduce Iteration Count

We present a fuzzy-parameter-based technique utilizing θ, which dynamically modifies the vertex removal range in each iteration. This method decreases the overall iterations, hence expediting the priority assignment procedure in JP-SL while preserving superior coloring quality.

1.0.2 Distributed Graph Coloring Framework

We develop a distributed graph coloring system utilizing graph partitioning for large-scale graphs that exceed 800MB in memory use. Utilizing METIS [12], the graph is partitioned into several subgraphs, each individually colored concurrently across different GPUs. The suggested approach significantly decreases execution time and enhances scalability for big graph processing through the integration of boundary preprocessing and ghost vertex [13] optimization.

The rest of the paper is organized as follows. Chapter II provides background information on the basic graph coloring algorithms, the data representations used, and the priority sorting techniques of Largest Degree First (LF) and Smallest Degree Last (SL). Chapter III presents our proposed optimizations in PA. Chapter IV demonstrates the experimental results, comparing the performance of our algorithm with other existing ap-

doi:10.6342/NTU202502064

proaches. Finally, Chapter V concludes the paper.

doi:10.6342/NTU202502064

Chapter 2 BACKGROUND

This chapter shows the Jones and Plassmann (JP) Algorithm and offers comprehensive descriptions of its variations, JP-SL and JP-LF, along with their implementations. The fundamental distinction between JP-SL and JP-LF resides in the methodology employed for priority ordering, as each variant utilizes a unique approach to ascertain vertex priorities.

The JP Algorithm is categorically divided into two phases: the Priority Allocation Phase and the Coloring Phase. Most versions of the JP Algorithm concentrate on enhancing the priority allocation step. Subsequent sections will offer a comprehensive introduction to each of these two phases.

2.1 Priority Allocation

During the Priority Allocation step, each vertex is allocated a priority, which may differ based on the particular algorithm employed. I will illustrate the priority ordering methods of JP-LF and JP-SL below. This section will utilize Fig. 2.1 as the foundational figure to demonstrate the sorting process.

doi:10.6342/NTU202502064

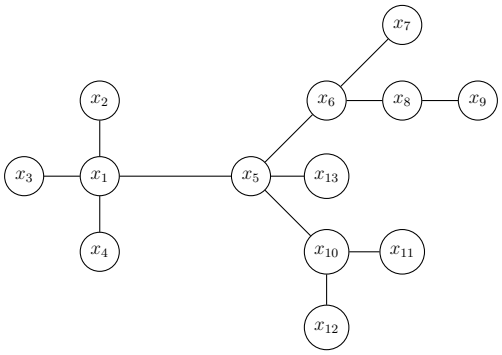


Fig. 2.1: Original Graph

2.1.1 JP-LF

In JP-LF, priority allocation is determined by the total degree of each node, with nodes arranged according to their number of adjacent nodes. As illustrated in Fig. 2.2, node x_1 possesses four nearby nodes, resulting in a priority of 4, but node x_2 has one adjacent node, conferring a priority of 1.

The benefit of JP-LF is its capacity for rapid priority sorting and its superior coloring quality relative to JP-FF and JP-R. Nonetheless, a limitation of this approach is its failure to represent the relative significance of nodes. Preferably, more significant nodes should be assigned greater priorities (i.e., be colored first). In Fig. 2.2, nodes x_1 and x_5 possess equivalent degrees, leading to uniform priority. In such instances, the JP Algorithm resolves the tie through random selection, which may inadequately distinguish the significance of these nodes.

2.1.2 JP-SL

In JP-SL, priority allocation is executed by sequentially eliminating nodes having a degree below a defined threshold, referred to as δ. This threshold is generally established at the minimal degree of the existing graph. This procedure is reiterated until every node

doi:10.6342/NTU202502064

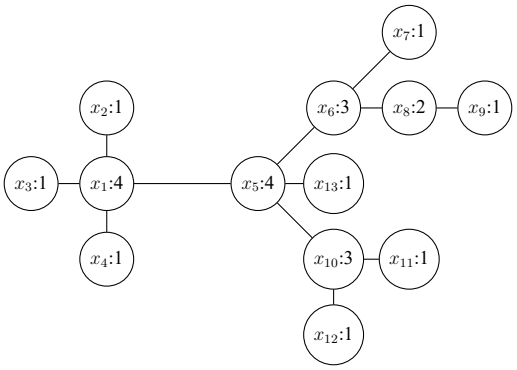


Fig. 2.2: JP-LF Priority Allocation

has been allocated a priority. As illustrated in Fig. 2.3, nodes with a degree below δ are eliminated in each iteration and designated a priority. The iterative approach culminates in the final output illustrated in Fig. 2.6.

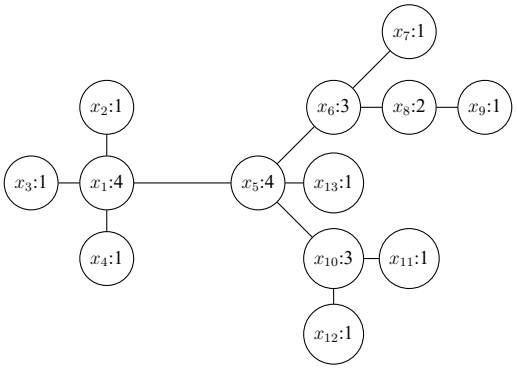


Fig. 2.3: JP-SL Priority Allocation:Original

Fig. 2.3 to Fig. 2.4 illustrate that the threshold is 1 ($\delta = 1$), resulting in the elimination of nodes $x_2, x_3, x_4, x_7, x_9, x_{11}, x_{12}, x_{13}$. The nodes are subsequently allocated a priority of 1.

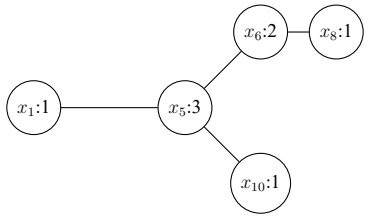


Fig. 2.4: JP-SL Priority Allocation:Iteration 1

As indicated by Fig. 2.4 to Fig. 2.5, the threshold is maintained at $\delta = 1$, leading to the elimination of nodes x_1, x_8, x_{10}. The nodes are subsequently allocated a priority of 2.

doi:10.6342/NTU202502064

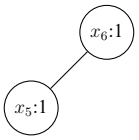


Fig. 2.5: JP-SL Priority Allocation:Iteration 2

Ultimately, Fig. 2.6 illustrates the resultant priority allocation. Nodes x_1 and x_5 exhibited identical priority in JP-LF; however, in JP-SL, x_5 possesses a superior priority. The nearby nodes of x_5 possess larger significance (i.e., higher degrees) than those of x_1, indicating that the local centrality of x_5 is superior. JP-SL typically attains superior coloring quality compared to JP-LF because it considers both the degree of each vertex and local centrality in establishing the priority order.

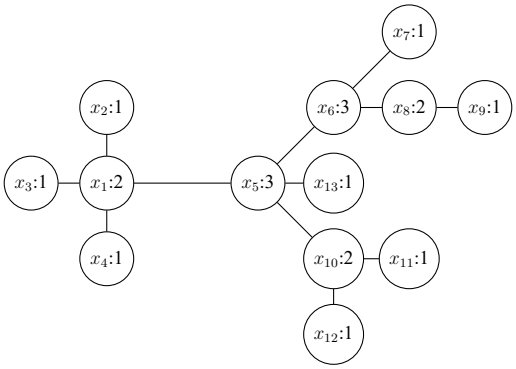


Fig. 2.6: JP-SL Priority Allocation:Result

2.2 Color Allocation

During the coloring step, we utilize the priority order established in the Priority Allocation Phase to transform the undirected graph into a directed graph. The conversion entails orienting edges between neighboring nodes so that the node with a higher priority directs towards the node with a lower priority. If two nodes possess identical priority, the direction is decided arbitrarily.

The whole implementation is detailed in Algorithm 1, which operates as a CUDA

doi:10.6342/NTU202502064

Algorithm 1: Directed Graph Generation

Data: Undirected Graph $G = (V, E)$
Result: Directed Acyclic Graph $DAGIn_v$ for each $v \in V$
/* First level parallelism */
1 **foreach** $v \in V$ **do**
  /* Compare with all neighbors */
2 **foreach** $n \in N(v)$ **do**
3 **if** $priority(v) < priority(n)$ **then**
4 $DAGIn_v \leftarrow DAGIn_v \cup \{n\}$;

kernel. In line 3, the algorithm compares the neighbor n of the current node v. If the priority of v is inferior to that of n, n is appended to v's input degree list, denoted as $DAGIn_v$ (line 5). Thus, each node records which of its neighboring nodes have input degrees. This method tries to improve the coloring phase: instead of assessing all neighboring nodes, it is sufficient to confirm that all nodes in the input degree list are colored. If they have, the current node is thereafter colored, significantly reducing the number of verifications.

Algorithm 2 illustrates the process of graph coloring, utilizing the directed graph produced by Algorithm 1 as input. Each vertex continues to iterate until it is colored. At the outset, `again` is assigned the value `false` (line 2), whereas `done` is assigned the value `true` (line 3). If all indegree vertices have been colored, it indicates that the subsequent vertex to be colored is the current vertex (line 16). Upon the completion of the coloring (line 17), the `while` loop terminates.

The variable `done` is utilized to ascertain whether all adjacent vertices in the indegree list have been colored. If any vertex remains uncolored, `done` is assigned `false` (line 12), signifying that the current vertex must await the coloring of all vertices with indegree before it can advance. At this juncture, `again` is assigned the value `true` (line 20) to persist in examining the indegree list.

doi:10.6342/NTU202502064

Algorithm 2: JP Algorithm Color Process

Data: Undirected Graph $G(V, E)$ with DAGIn information
Result: Color assignment $color_v$ for each $v \in V$

```
1  repeat
2      again ← false;
3      done ← true;
4      foreach v ∈ V do
5          if not colored(v) then
6              foreach n ∈ DAGIn_v do
7                  if colored(n) then
                       /* Remove neighbor's possible color */
8                      posscol_v ← posscol_v \ color_n;
9                  else
10                     done ← false;
11             bestcol_v ← best(posscol_v);
12             if done then
13                 color_v ← bestcol_v;
14             else
15                 again ← true;
16 until again == false;
```


doi:10.6342/NTU202502064

Chapter 3 METHODOLOGIES

The overall architecture of our algorithm is illustrated in Fig. 3.1. As shown, the system incorporates a Distributed Optimization Component (Section 3.2) and includes three major enhancements in the Priority Allocation (PA) Phase: (1) **Resilient SL** (Section 3.1.1), (2) **Worklist-Based Removal** (Section 3.1.2), and (3) **Fuzzy Parameter** (Section 3.1.3).

Once the priority assignment is completed, the resulting ordering is passed to the Color Assignment (CA) Phase, where we apply the ECL-GC algorithm to perform graph coloring. Our framework places strong emphasis on PA (Priority Allocation) optimization, particularly focusing on accelerating the priority ordering process of the SL Strategy. The following sections describe the implementation details of each optimization in depth.

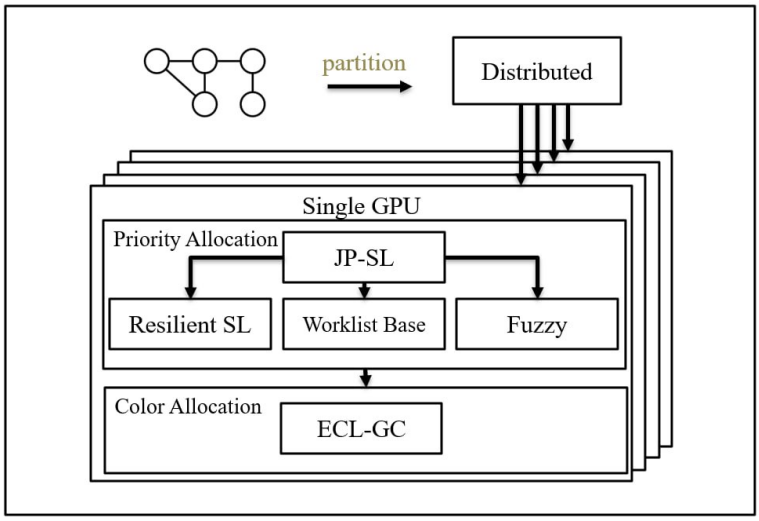


Fig. 3.1: Architecture of R-SL

doi:10.6342/NTU202502064

3.1 Priority Allocation

3.1.1 Resilient SL

Traditional JP-SL Algorithms require first identifying the Minimum Degree δ among all vertices, and then removing vertices v satisfying $Degree(v) \leq \delta$. This approach necessitates scanning all active vertices twice per iteration, resulting in significant runtime overhead and requiring frequent global synchronizations. Consequently, the PA phase in JP-SL becomes highly time-consuming.

Algorithm 3 presents our proposed method, R-SL. The key advantage of R-SL is that it completes the JP-SL process with only a single scan per iteration. Specifically, while removing vertices, the algorithm concurrently updates the next iteration's Minimum Degree δ (lines 8 and 20).

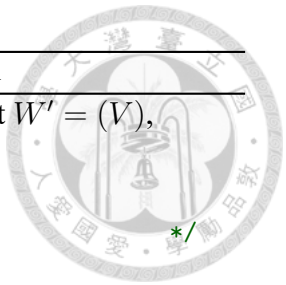
To augment efficiency, we present a **Worklist-Based Parallelization** technique, which will be further discussed in Section 3.1.2. This method enhances vertex removal efficiency by enabling threads within a warp to process removable vertices jointly.

3.1.2 Worklist-Based Parallelization

We observed that in numerous iterations, the quantity of vertices eliminated is rather minimal. This imbalance results in diminished GPU use, since the majority of threads must remain inactive while awaiting the completion of the removal phase before advancing to the subsequent iteration.

To resolve this issue, we maintain a *Remove Worklist* W, which aggregates all vertices

doi:10.6342/NTU202502064



Algorithm 3: R-SL Algorithm with Fuzzy Parameter Optimization

Data: Graph $G = (V, E)$, Work list $W = (V)$, Acyclic Work list $W' = (V)$, threshold = δ, Priority list P
Result: Priority list P

```
   /* Init Parameters                                                    */
1  worker ← 0;
2  W ← ∅;
3  δ_n, δ_c ← 1;
   /* thread-base parallelism                                            */
4  repeat
5      foreach v ∈ G do
6          atomicMin(δ_n, Degree(v));
7          if Degree(v) ≤ θ + δ_c then
8              atomicAdd(worker,1);
9              W ← W ∪ v;
10     grid.syn();
       /* warp-base parallelism                                          */
11     foreach wv ∈ W do
12         P(v) ← iteration + (Degree(wv) − δ_c) + 1;
           /* use warp scan neighbors                                    */
13         foreach n ∈ N(v) do
14             atomicSub(Degree(n), 1);
15             atomicMin(δ_n, Degree(n));
16     grid.syn();
17     if thread_id == 0 then
18         δ_c ← δ_n;
19         δ_n ← INT_MAX;
20         W ← ∅;
21     grid.syn();
22 until worker == nodes;
```

that satisfy the removal criterion (i.e., $Degree(v) \leq \delta + \theta$), as indicated in line 9 of the method. Thereafter, we employ Warp-Level Parallelism to navigate and process each vertex in W (line: 11), subsequently updating its adjacent vertices as needed. This design ensures enhanced GPU utilization and markedly increases the efficiency of the removal phase relative to simplistic thread-per-vertex solutions.

doi:10.6342/NTU202502064

3.1.3 Fuzzy Parameter

We propose an optimization strategy based on SL to reduce the iterations needed during the **PA** Phase. The fundamental concept is to incorporate a user-defined Fuzzy Parameter θ, so modifying the original removal criterion from vertices with $Degree(v) \leq \delta$ (Minimum Degree of graph) to those with $Degree(v) \leq \delta + \theta$. To avert significant deterioration in color quality, we implement a localized **LF** Priority Ordering for the additionally eliminated vertices, assigning their priorities as $Degree(v) - \delta$. This maintains a degree-based hierarchy within the fuzzy spectrum, improving the stability and efficiency of the Graph Coloring (**GC**), while offering users the freedom to balance performance and quality. The implementation details of this strategy are outlined in Algorithm 3(line: 12), which demonstrates how vertex priorities and iteration counts are updated accordingly.

3.2 Distributed

Considering that parallel graph coloring can still require substantial execution time on extremely large graphs, we developed a Distributed Graph Coloring Algorithm. The detailed workflow is illustrated in Fig. 3.2. Initially, we partition the large graph into multiple subgraphs. Subsequently, boundary vertices are pre-colored using our proposed R-SL algorithm. These pre-colored boundary vertices are then converted into ghost vertices [13] and their coloring information is propagated to each subgraph. This procedure effectively avoids color conflicts during the subsequent coloring phase within each subgraph.

However, directly partitioning the graph can inadvertently lower the priority of boundary vertices during the JP-SL Priority Allocation (PA) Phase, due to the artificial reduction

doi:10.6342/NTU202502064

in their degrees. To mitigate this effect, we further leverage ghost vertices by including these pre-colored boundary vertices into the Priority Allocation Computations. This ensures the boundary vertices' priorities closely reflect their original, unpartitioned degrees, thereby preventing an undesired reduction in priority and maintaining coloring quality and consistency.

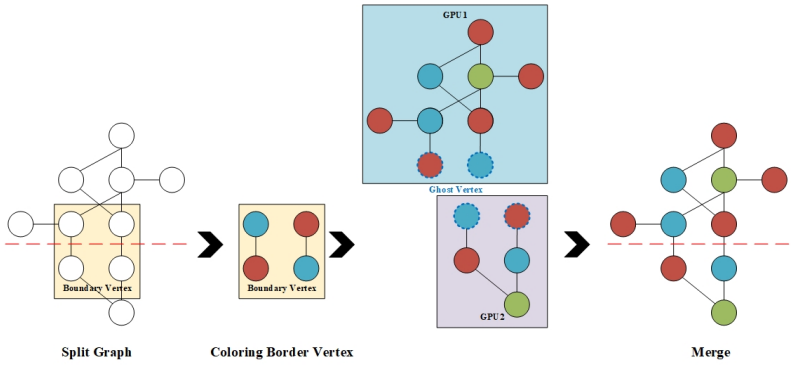


Fig. 3.2: R-SL Graph Partition Flow

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

Chapter 4 EXPERIMENTAL

4.1 Environment

Our experimental evaluation comprises two parts. The first part evaluates the performance of R-SL against other parallel algorithms and investigates the influence of the Fuzzy Parameter θ on runtime and coloring quality. The second part concentrates on assessing the scalability and efficacy of our distributed graph coloring technology.

In the single-GPU experiments, we utilized a system featuring an Intel Core i9-14900 CPU and a single NVIDIA RTX 4090 GPU, operating on CUDA version 12.6 with driver version 560. For the distributed experiments, we utilized a system featuring an Intel Xeon Gold 6154 CPU paired with eight NVIDIA V100 GPUs.

4.2 Comparison with Other Algorithms

Table. 4.2 lists the existing graph coloring techniques employed for comparative analysis. These are classified into two primary categories: speculative greedy (SGR) and maximal independent set (MIS). SGR comprises csrcolor, data_pq, data_wlc, and kokkos-VB_BIT, whereas MIS encompasses ECL-GC and various iterations of the ones and Plassmann (JP) series, including JP-SL, JP-ADG, and JP-SLL.

doi:10.6342/NTU202502064

Table. 4.1 enumerates the graph datasets utilized in the assessment. The datasets are sourced from reputable organizations including Stanford Network Analysis Platform (SNAP) [14], SuiteSparse Matrix Collection (SSMC) [15], Carnegie Mellon University (CMU) [16], Discrete Mathematics and Theoretical Computer Science at the University of Rome (DIMACS) [17], and the Galois framework (Galois) [18]. They range in size from small-scale graphs (under 1MB) to large-scale graphs (over 1GB), encompassing various properties and topologies.

Table 4.1: Graph datasets used for evaluation (|V|: number of vertices, |E|: number of edges)

Graph name	**Type**	$\|V\|$	$\|E\|$
as-skitter	Internet topology	1,696,415	22,190,596
cit-Patents	Patent citations	3,774,768	33,037,894
delaunay_n24	Triangulation	16,777,216	100,663,202
Email-Enron	Communication	36,692	183,831
europe_osm	Road map	50,912,018	108,109,320
facebook	Social network	4,039	88,234
le450_25d	Leighton graph	450	50,000
r4-2e23	Random	8,388,608	67,108,846
rmat22	RMAT	4,194,304	65,660,814
school1	Class scheduling	385	19,095
Slashdot0811	Social network	77,360	905,468
Slashdot0902	Social network	82,168	948,464
soc-Epinions1	Social network	75,879	508,837
soc-pokec	Social network	1,632,803	30,622,564
Stanford	Web links	281,903	2,312,497
twitter	Social network	81,306	1,768,149
wiki-Talk	Collaboration	2,394,385	5,021,410
wiki-Vote	Vote network	7,115	103,689
youtube	Social network	1,134,890	2,987,624
kron_g500	Kronecker	2,097,152	182,081,864
uk-2002	Web links	18,520,486	523,574,516

4.2.1 Single-GPU Performance Comparison

The results of the study are encapsulated in Table. 4.2. Given the intrinsic randomness of coloring algorithms, we conducted five rounds for each dataset and chose the optimal outcome (minimum colors) for final evaluation.

doi:10.6342/NTU202502064

The results demonstrate that the proposed method attains comparable color quality to JP-SL while markedly exceeding JP-SL in execution speed. R-SL attains about a six-fold acceleration even in the absence of the Fuzzy Parameter θ. With the introduction of $\theta = 10$, the speedup significantly escalates, approaching 43 times. Despite the significant improvements in execution time, the overall color count remains unchanged, indicating that our methodology successfully preserves superior coloring quality while substantially enhancing runtime efficiency.

Furthermore, with $\theta = 10$, the execution time of R-SL is slightly higher than that of existing algorithms like JP-SLL; nonetheless, it exhibits considerable advantages in coloring quality, showcasing enhanced stability and superiority in the quantity of colors utilized.

Table 4.2: Performance comparison of graph-coloring algorithms (entries are Color / Time(ms))

	SGR-based algorithms								**MIS-based algorithms**											
Dataset	csrcolor [19]		data_pq [20]		data_wlc [20]		VB_BIT [21, 22]		ECL-GC [7]		JP-ADG [23]		JP-SLL (r=1) [8]		JP-SL [10, 11]		R-SL		R-SL(θ=10)	
as-skitter	335 /	66.1	83 /	47.8	98 /	58.8	109 /	292.3	71 /	7.5	81 /	40	71 /	23.7	70 /	451.9	**68** /	75.9	**68** /	16.1
cit-Patents	143 /	27.3	15 /	3.1	16 /	1.7	24 /	9.1	14 /	2.2	18 /	4.9	15 /	4.2	14 /	453.5	14 /	143.3	14 /	7.7
delaunay_n24	48 /	82.1	7 /	7.5	8 /	6.8	9 /	12.1	6 /	5.6	8 /	14	7 /	10.1	7 /	1413.8	**5** /	474.8	6 /	9.7
Email-Enron	153 /	11.3	33 /	3.0	48 /	3.6	50 /	11.6	29 /	1.1	39 /	2.7	31 /	2.3	**27** /	12.3	28 /	3.7	28 /	2
europe_osm	32 /	229.8	5 /	26.8	5 /	33.7	7 /	12.5	5 /	7.9	5 /	25.0	5 /	115.7	**4** /	2580.5	**4** /	72	5 /	19.2
facebook	208 /	13.0	76 /	3.7	87 /	6.1	88 /	12.5	76 /	1.4	87 /	2.6	87 /	2.5	**74** /	10.4	**74** /	4.4	75 /	2.2
le450_25d	126 /	9.8	30 /	1.0	37 /	3.9	37 /	7.2	31 /	1.0	36 /	2.0	32 /	1.7	31 /	3.8	31 /	2.2	30 /	1.7
r4-2e23.sym	48 /	43.9	8 /	6.6	9 /	6.5	10 /	9.9	7 /	4.4	8 /	9.4	8 /	7.7	7 /	25.6	**6** /	40.5	7 /	14.1
rmat22.sym	341 /	43.8	69 /	8.4	87 /	10.8	94 /	36.1	38 /	10.2	63 /	15.0	45 /	12.9	37 /	265.2	38 /	44.6	**36** /	16.5
school1	159 /	11.0	33 /	2.4	38 /	2.8	41 /	7.7	34 /	1.0	41 /	2.2	38 /	1.9	**14** /	3.5	**14** /	2.1	14 /	1.5
Slashdot0811	175 /	12.4	44 /	2.5	54 /	3.2	59 /	14.9	40 /	1.2	47 /	3.1	39 /	2.1	**32** /	23.1	33 /	4.5	**32** /	2.1
Slashdot0902	181 /	12.8	46 /	2.8	55 /	4.2	60 /	15.1	40 /	1.1	47 /	3.0	39 /	2.3	**33** /	24.5	**33** /	4.6	**33** /	2.1
soc-Epinions1	221 /	14.1	46 /	3.0	65 /	4.3	69 /	23.3	41 /	1.5	54 /	4.3	43 /	2.9	34 /	26.4	34 /	5.1	34 /	2.4
soc-pokec-rel.	192 /	22.2	44 /	7.0	53 /	4.5	57 /	21.9	32 /	7.5	44 /	12.0	33 /	10.1	**29** /	575.8	**29** /	112.8	**29** /	26.9
Stanford	238 /	31.8	64 /	19.4	62 /	7.6	64 /	71.3	64 /	4.5	65 /	14.0	61 /	13.1	**61** /	98.5	**61** /	22.9	62 /	4.9
twitter	233 /	15.0	72 /	6.2	82 /	6.4	81 /	24.6	73 /	2.8	77 /	4.7	74 /	3.0	**71** /	90.1	**71** /	22.3	72 /	4.3
wiki-Talk	394 /	64.1	82 /	35.5	103 /	27.6	112 /	149.9	72 /	4.8	84 /	24.0	67 /	16.8	**56** /	197.4	57 /	16.8	**56** /	7.7
wiki-Vote	175 /	11.6	29 /	1.4	41 /	2.6	44 /	8.5	28 /	1.1	38 /	2.5	32 /	2.0	27 /	9.5	27 /	3.1	**27** /	1.9
youtube	203 /	27.1	42 /	9.2	52 /	7.8	60 /	44.6	32 /	1.6	43 /	9.2	33 /	7.6	28 /	101.6	28 /	12.4	**27** /	4
TOTAL	3605 /	749.2	828 /	197.2	1000 /	202.7	1075 /	785.1	733 /	68.4	885 /	197.3	760 /	142.5	656 /	6367.4	**655** /	1,067.8	**655** /	147

doi:10.6342/NTU202502064

4.2.2 GPU vs CPU: Algorithm Comparison

To thoroughly evaluate the effectiveness of R-SL, we compared its color quality and performance with classical CPU-based algorithms, both sequential and parallel. All parallel CPU algorithms presented in this comparison were executed using 32 threads to ensure consistency and fair benchmarking. Specifically, we include the Smallest-Last (SL^M) algorithm by Matula and Beck [11] and the JP-SL^W heuristic introduced by Hasenplaugh et al. [8] for comprehensive context.

The SL^M algorithm iteratively removes one vertex at a time—the vertex with the smallest current degree—in the Priority Allocation (PA) phase, followed by a Greedy coloring approach in the Color Assignment (CA) phase. Conversely, JP-SL^W simultaneously removes all vertices with the minimum global degree during the PA phase and applies the JP for color assignment.

Furthermore, while the Dstur algorithm is recognized for its superior coloring quality, it was found to be impractical for many of our test datasets, frequently exceeding the specified 30-minute runtime limit (indicated by "-").

The results in Table 4.3 demonstrate R-SL's robust performance. It achieves a total color count only slightly higher than SL^M, while running at nearly **6× the speed** of the GPU-based JP-SL^W. This illustrates R-SL's exceptional balance between coloring quality and computational efficiency.

doi:10.6342/NTU202502064

Table 4.3: GPU vs CPU: Algorithm Comparison (entries are Color / Time, CPU in seconds, GPU in milliseconds)

Dataset	CPU Sequential (s)			CPU parallel (s)			GPU (ms)			
	Dstur	Greedy	SL^M	JP-SL^W	ADG	SLL	JP-SL^W	ADG	SLL	R-SL
as-skitter	-/ -	71/ 1.93	67/ 10.63	67/ 2.70	74/0.06	73/0.05	70/ 451.9	80/ 4.67	73/ 4.77	68/ 75.9
cit-Patents	-/ -	14/ 3.97	13/ 48.96	13/ 14.96	15/0.16	14/0.09	14/ 453.5	18/ 2.47	16/ 3.09	14/143.3
delaunay_n24	-/ -	7/ 5.98	5/ 23.92	6/ 42.73	7/0.53	7/0.22	7/1413.8	7/ 9.06	7/ 8.95	5/474.8
Email-Enron	25/ 1.58	29/ 0.00	27/ 0.08	28/ 0.02	31/0.01	29/0.00	27/ 12.3	33/ 0.35	31/ 0.30	28/ 3.7
europe_osm	-/ -	5/ 5.63	4/ 73.36	4/ 41.41	5/1.32	5/0.40	4/2580.5	5/11.37	5/11.37	4/ 72
facebook	71/ 0.25	76/ 0.00	74/ 0.04	76/ 0.02	76/0.01	75/0.00	74/ 10.4	88/ 0.71	87/ 0.63	74/ 4.4
le450_25d	29/ 0.05	30/ 0.00	31/ 0.00	31/ 0.01	30/0.01	30/0.00	31/ 3.8	32/ 0.47	32/ 0.45	31/ 2.2
r4-2e23	-/ -	7/ 8.06	6/ 45.19	6/ 1.59	8/0.34	7/0.17	7/ 25.6	8/ 4.78	8/ 4.85	6/ 40.5
rmat22	-/ -	37/ 7.66	37/ 69.55	37/ 3.60	38/0.30	37/0.26	37/ 265.2	60/ 8.45	44/10.38	38/ 44.6
school1	27/ 0.07	32/ 0.00	15/ 0.00	14/ 0.00	36/0.00	35/0.00	14/ 3.5	34/ 0.46	38/ 0.52	14/ 2.1
Slashdot0811	32/ 8.71	39/ 0.04	34/ 0.20	33/ 0.04	41/0.01	39/0.01	32/ 23.1	43/ 0.42	39/ 0.39	33/ 4.5
Slashdot0902	32/ 9.91	41/ 0.04	32/ 0.23	34/ 0.05	42/0.04	41/0.01	33/ 24.5	43/ 0.48	39/ 0.38	33/ 4.6
soc-Epinions1	30/ 7.58	44/ 0.01	34/ 0.16	34/ 0.03	44/0.03	43/0.02	34/ 26.4	46/ 0.86	43/ 0.75	34/ 5.1
soc-pokec	-/ -	33/ 3.60	29/ 31.42	38/ 6.29	34/0.17	31/0.13	29/ 575.8	35/ 6.37	34/ 6.70	29/112.8
Stanford	63/ 76.79	64/ 0.16	61/ 1.33	61/ 0.45	64/0.01	64/0.01	61/ 98.5	65/ 0.79	61/ 0.92	61/ 22.9
twitter_combined	71/ 11.63	73/ 0.10	71/ 0.89	71/ 0.18	74/0.02	73/0.01	71/ 90.1	77/ 1.25	74/ 0.93	71/ 22.3
wiki-Talk	-/ -	72/ 0.51	56/ 2.46	55/ 0.51	74/0.07	70/0.04	56/ 197.4	83/ 5.53	67/ 4.09	57/ 16.8
wiki-Vote	23/ 0.32	27/ 0.00	27/ 0.01	28/ 0.02	29/0.00	28/0.00	27/ 9.5	32/ 0.43	32/ 0.45	27/ 3.1
youtube	26/1663.93	33/ 0.48	28/ 3.18	28/ 0.46	35/0.04	32/0.02	28/ 101.6	43/ 1.16	33/ 1.04	28/ 12.4
TOTAL	-/ -	734/38.17	651/311.61	664/115.05	757/3.15	733/1.45	656/6367.4	832/60.08	763/60.96	655/ 1068

4.3 Impact of Worklist-Based Parallelization

Table 4.4 compares the coloring results and runtime between two implementations: the baseline version (**w/o wl**), which processes one vertex per thread sequentially, and the optimized version (**wl**), which incorporates a worklist-based design along with warp-level parallelism. Both implementations yield identical color counts across all datasets—except for *europe_osm* and *rmat22*, which show minor differences—demonstrating that worklist-based parallelization preserves coloring quality.

In terms of runtime, the **wl** version significantly reduces computation time across most datasets. Notable speedups were observed in *wiki-Talk* (10×), *youtube* (4.5×), *soc-pokec* (2.6×), and *Slashdot0811/0902* (approximately 3×).

Overall, the total runtime decreased from 1738.72 ms in the baseline version to 1024.73 ms in the optimized version, yielding an average speedup of approximately **1.7**×. This result demonstrates that the combination of worklist scheduling and warp-level opti-

doi:10.6342/NTU202502064

mization enables more efficient GPU thread utilization, particularly for graphs with high parallelism and sparsely active vertices in each iteration.

Although a few datasets (e.g., *delaunay_n24* and *europe_osm*) exhibit slight runtime increases due to the overhead of maintaining worklists, the trade-off is justified by the substantial performance gains observed across the majority of real-world graphs.

Table 4.4: Impact of Worklist-based Parallelization on Coloring Quality and Runtime

Dataset	w/o wl		wl	
	Color	Runtime (ms)	Color	Runtime (ms)
as-skitter	68	324.4	68	83.25
cit-Patents	14	183.67	14	176.48
delaunay_n24	7	330.68	7	400.41
Email-Enron	28	8.03	28	4.72
europe_osm	5	43.53	4	63.18
facebook	74	8.22	74	5.32
le450_25d	31	3.15	31	2.31
r4-2e23	7	18.73	7	19.95
rmat22	37	131.36	38	45.74
school1	14	3.5	14	2.32
Slashdot0811	33	14.22	33	5.06
Slashdot0902	33	14.69	33	5.2
soc-Epinions1	34	16.92	34	6.11
soc-pokec	29	300.52	29	117.31
Stanford	61	68.04	61	31.47
twitter_combined	71	61.6	71	24.71
wiki-Talk	57	139.46	57	13.88
wiki-Vote	27	6.28	27	3.56
youtube	28	61.72	28	13.75
TOTAL	658	1738.72	658	1024.73

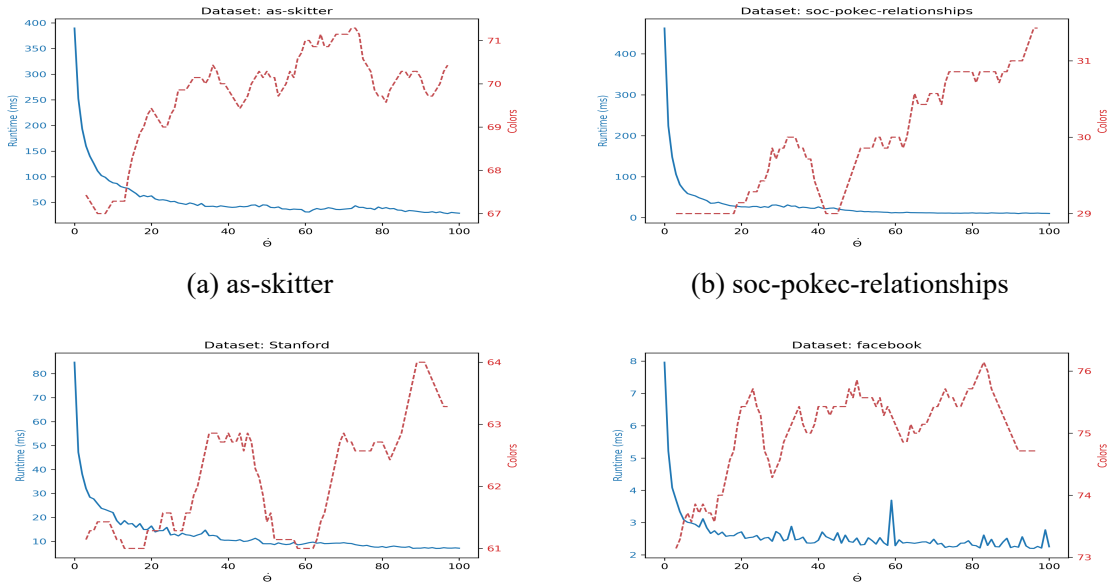
4.4 Effect of the Fuzzy Parameter

This study performed statistical analysis on four datasets, with findings illustrated in the image. In Fig. 4.1, the x-axis signifies the Fuzzy Parameter θ, the left y-axis (in blue) indicates the execution time, and the right y-axis (in red) shows the amount of colors utilized.

The use of the Fuzzy Parameter results in an exponential reduction in the algorithm's execution time, markedly improving efficiency. However, when θ becomes very big, the

doi:10.6342/NTU202502064

quality of coloring deteriorates, resulting in an increase in the number of colors employed. This phenomena distinctly exemplifies a conventional trade-off between execution performance and coloring quality. Thus, users can modify the θ parameter to meet specific application needs, thereby optimizing efficiency and color quality.



(a) as-skitter (b) soc-pokec-relationships

(c) Stanford (d) Facebook

Fig. 4.1: Runtime and Coloring Behavior Under Varying θ

4.5 Ordering Consistency

To confirm that our proposed algorithm attains effectiveness comparable to the JP-SL, we conducted a consistency analysis. The main objective of this investigation is to identify whether the priority ordering produced by our proposed R-SL algorithm roughly corresponds with that of JP-SL. In the graph coloring problem, the relative sequence of vertex priorities is important, rather than their absolute values. Hence, our objective is to ensure that the relative priority ordering established by JP-SL is preserved by R-SL. The formula for measuring consistency is defined as follows:

Given two priority lists P_{SL} (JP-SL) and P_S (another JP-Series algorithm), each con-

doi:10.6342/NTU202502064

taining n vertices, the total number of pairwise comparisons is defined as:

$$T = n(n-1) \tag{4.1}$$

Consistency C is computed by summing the number of vertex pairs that preserve the same relative ordering across both lists:

$$C = \sum_{i \neq j} \left[(P_{SL,i} > P_{SL,j} \wedge P_{S,i} > P_{S,j}) \vee (P_{SL,i} < P_{SL,j} \wedge P_{S,i} < P_{S,j}) \right] \tag{4.2}$$

The final consistency ratio is given by:

$$\text{Consistency Ratio} = \frac{C}{T} \tag{4.3}$$

This ratio quantifies the degree of agreement between the priority rankings of JP-SL and the compared algorithm.

Fig. 4.2 illustrates our experimental results, with the x-axis denoting the different datasets and the y-axis reflecting the consistency ratio between the proposed approaches and JP-SL. Algorithms with priority rankings akin to JP-SL will attain a ratio approaching 100%. R-SL exhibits the highest consistency ratio among the evaluated algorithms, underscoring its efficacy in accurately mirroring JP-SL's priority ordering. Furthermore, despite the introduction of the Fuzzy Parameter ($\theta = 10$), the consistency ratio remains comparatively stable without substantial degradation, demonstrating that the fuzzy optimization effectively reconciles improved execution efficiency with the preservation of priority consistency.

doi:10.6342/NTU202502064

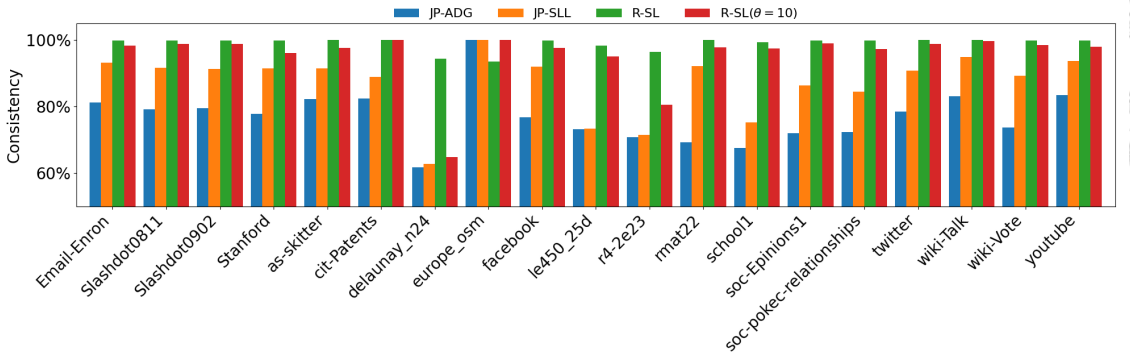
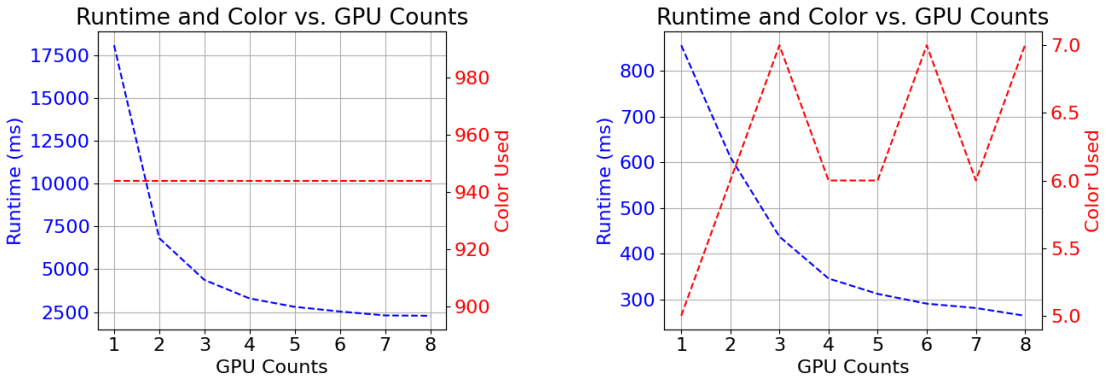


Fig. 4.2: Ordering consistency of approximate ordering heuristics and R-SL with respect to JP-SL across various datasets.



(a) uk-2002 (b) delaunay_n24

Fig. 4.3: Runtime Scaling of R-SL on a multiple-GPU machine.

4.6 Scalability

We utilize `OpenMP` [24] to enable multi-GPU execution. Fig. 4.3 shows the runtime scalability across up to eight V100 GPUs, with the x-axis representing the number of GPUs and the y-axis indicating runtime.

For both the *uk_2002* and *delaunay_n24* datasets, Fig. 4.3 demonstrates a consistent reduction in runtime as the number of GPUs increases, highlighting the scalability across different types of graphs.

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

Chapter 5 CONCLUSION

This paper rigorously examines the performance limitations in the Priority Allocation (PA) phase of graph coloring algorithms, proposing various novel optimization methodologies and architectural designs.

During the PA phase, we provide the Resilient Smallest Degree Last (R-SL) algorithm, which substitutes the recurrent kernel launches characteristic of conventional JP-SL with an efficient synchronization method utilizing atomic operations and active workers. This architecture dramatically decreases data transmission and computational duration. Additionally, we include a Fuzzy Parameter (θ) to flexibly adjust the vertex removal range, therefore decreasing the number of iterations and enhancing execution performance.

In response to the increasing demand for extensive graph processing, we propose a distributed multi-GPU graph coloring system utilizing METIS partitioning. The system facilitates effective parallel processing of huge datasets with boundary node preprocessing and a ghost vertex method. Experimental findings indicate that this strategy considerably decreases execution time while exerting just a negligible effect on coloring quality.

doi:10.6342/NTU202502064

doi:10.6342/NTU202502064

References

[1] Assefaw Hadish Gebremedhin, Fredrik Manne, and Alex Pothen. What color is your jacobian? graph coloring for computing derivatives. *SIAM Rev.*, 47:629–705, 2005.

[2] Maxim Naumov, Patrice Castonguay, and Jonathan Cohen. Parallel graph coloring with applications to the incomplete-lu factorization on the gpu. *Nvidia White Paper*, 2015.

[3] Umit Catalyurek, John Feo, Assefaw Gebremedhin, Mahantesh Halappanavar, and Alex Pothen. Graph coloring algorithms for muti-core and massively multithreaded architectures, 2012.

[4] S. M. Ferdous, Reece Neff, Bo Peng, Salman Shuvo, Marco Minutoli, Sayak Mukherjee, Karol Kowalski, Michela Becchi, and Mahantesh Halappanavar. Picasso: Memory-Efficient Graph Coloring Using Palettes With Applications in Quantum Computing. 1 2024.

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

doi:10.6342/NTU202502064

[6] Mark T. Jones and Paul E. Plassmann. A parallel graph coloring heuristic. *SIAM Journal on Scientific Computing*, 14(3):654–669, 1993.

[7] Ghadeer Alabandi, Evan Powers, and Martin Burtscher. Increasing the parallelism of graph coloring via shortcutting. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '20, page 262–275, New York, NY, USA, 2020. Association for Computing Machinery.

[8] William Hasenplaugh, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. Ordering heuristics for parallel graph coloring. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, page 166–177, New York, NY, USA, 2014. Association for Computing Machinery.

[9] Dominic J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *Comput. J.*, 10:85–86, 1967.

[10] James R. A. Allwright, Rajesh R. Bordawekar, Paul D. Coddington, Kivanç Dinçer, and Christine Martin. A comparison of parallel graph coloring algorithms. 1995.

[11] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, jul 1983.

[12] George Karypis and Vipin Kumar. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, September 1998.

[13] Ian Bogle, Erik G. Boman, Karen Devine, Sivasankaran Rajamanickam, and George M. Slota. Distributed memory graph coloring algorithms for multiple gpus. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 54–62, 2020.

doi:10.6342/NTU202502064

[14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[15] Tim Davis and Yifan Hu. Suitesparse matrix collection. https://sparse.tamu.edu/. Accessed: 2021-05-06.

[16] Michael Trick. COLOR graph coloring instances. https://mat.tepper.cmu.edu/COLOR/instances/, 2018. Carnegie Mellon University, accessed on 2025-03-31.

[17] DIMACS, Center for Discrete Mathematics and Theoretical Computer Science. Dimacs challenge graph instances. http://www.dis.uniroma1.it/challenge9/download.shtml. Accessed: 2021-05-06.

[18] Galois Project, ISS - The University of Texas at Austin. Galois graph benchmark suite. https://iss.oden.utexas.edu/?p=projects/galois. Accessed: 2021-05-06.

[19] Xuhao Chen and Li. Csrcolor. https://github.com/chenxuhao/csrcolor, 2021. Last accessed on May 6, 2021.

[20] Xuhao Chen, Pingfan Li, and Canqun Yang. Efficient and high-quality sparse graph coloring on the GPU. *CoRR*, abs/1606.06025, 2016.

[21] Mehmet Deveci, Erik G Boman, Karen D. Devine, and Sivasankaran Rajamanickam. Parallel graph coloring for manycore architectures. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 892–901, 2016.

[22] Kokkos-Kernels Developers. Kokkos-kernels. https://github.com/kokkos/kokkos-kernels, 2021. Last accessed on May 6, 2021.

doi:10.6342/NTU202502064

[23] Maciej Besta, Armon Carigiet, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, and Torsten Hoefler. High-performance parallel graph coloring with strong guarantees on work, depth, and quality. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020.

[24] Chao-Tung Yang, Chih-Lin Huang, and Cheng-Fang Lin. Hybrid cuda, openmp, and mpi parallel programming on multicore gpu clusters. *Computer Physics Communications*, 182(1):266–269, 2011.

doi:10.6342/NTU202502064