A Research on Distributed and Parallel Graph Coloring Algorithms

Yihang Shen

June 4, 2024

Abstract

Many challenges in solving the vertex graph coloring problem(VGCP) using a parallel strategy are waiting to be resolved. On the one hand, previous algorithms using Pregel-like frameworks, such as Apache Giraph based on Hadoop infrastructures, made some effective works in minimizing the execution time and used color number by cutting supersteps as much as possible, partly or integrally. But these algorithms still cannot achieve high parallelism in the entire coloring process due to the sequential execution of conflict check and correction stages. On the other hand, parallel graph coloring algorithms on GPU have made remarkable developments in reducing the number of iterations in recursive methods, achieving an impressive speedup over the older algorithms on GPU architecture. The shortcoming of the latter is that processing complex data structures like graphs and trees in parallel by moving graph applications to GPU architectures can be costly and time-consuming.There are great challenges in performing graph coloring on GPU in general. First, the long-tail problem exists in the recursion algorithm because the conflict (i.e., different threads assign the adjacent nodes to the same color) becomes more likely to occur as the number of iterations increases. Second, it is hard to parallelize the sequential spread algorithm because the color allocation depends on the adjoining iteration. Third, the atomic operation is widely used on GPU to maintain the color list, which can greatly reduce the efficiency of GPU threads.

1 Introduction

The Vertex Graph Coloring Problem (VGCP) is a fundamental issue in graph theory, requiring the assignment of colors to vertices such that no two adjacent vertices share the same color. This problem has numerous application fields across diverse areas, including computer science, telecommunications, traffic management, bioinformatics, and resource allocation in multiple scenarios. The primary objective in VGCP is to minimize the number of colors used, a value known as the chromatic number, and the program execution time as much as possible. However, determining the chromatic number is an NP-complete problem, necessitating near-optimal solutions due to the impracticality of exact solutions for large graphs. Meanwhile, limitation of parallelism and complexity of hardware environment cause the chromatic number and the execution time cannot be achieved simultaneously. The existing algorithms either sacrifice the execution time to obtain a better chromatic number, or sacrifice the chromatic number to obtain a less execution time.

Modern applications often generate extremely large graphs, such as those found in social networks, where the number of vertices and edges can be vast. These large-scale graphs pose significant challenges for coloring algorithms, particularly in terms of computational efficiency and scalability. Traditional sequential algorithms are inadequate for providing timely solutions for such large graphs, highlighting the need for parallel computing strategies.

Recent advancements in parallel graph coloring have explored various methodologies. Algorithms based on frameworks like Apache Giraph, which utilize Hadoop infrastructures, have made progress in reducing execution times and minimizing the number of colors by optimizing the amount of supersteps. However, these approaches often struggle to achieve high levels of parallelism throughout the entire coloring process, primarily due to the sequential execution required for conflict resolution stages. Parallel algorithms running on GPU architectures have shown promise by reducing the number of iterations and achieving significant speedups. Nonetheless, these methods face challenges related to the overhead and complexity of managing large graph structures on GPUs.

Utilizing big data frameworks such as Hadoop, Apache Spark, Apache Flink, and Google BigQuery offers significant advantages and suitability for addressing the Vertex Graph Coloring Problem (VGCP), particularly for large-scale graphs. By leveraging the parallel processing capabilities and distributed storage mechanisms of these big data platforms, algorithms for graph coloring can achieve significant performance improvements. They enable the decomposition of large graphs into manageable subgraphs, which can be processed concurrently, thus reducing overall computation time and resource usage. Although recent research achievements have been made based on these frameworks, releasing the full potential of graph processing on big data frameworks is still a big challenge.

2 Motivation

In this section, we demonstrate the main problem when running the graph coloring algorithms on GPU. According to previous studies, the graph coloring algorithms can be divided into several categories, in which sequential spread and recursion are most often used.

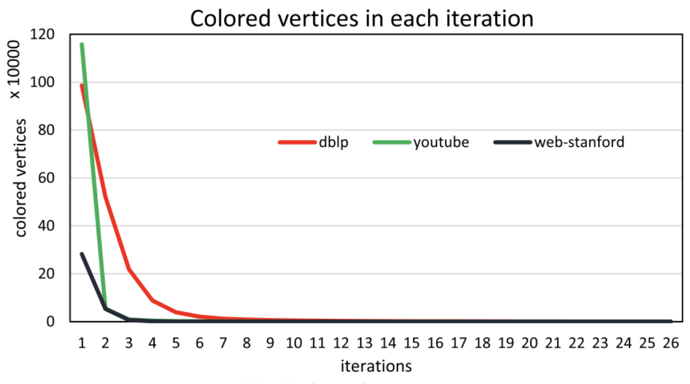
The basic idea of the sequential spread algorithm is to traverse the entire graph using the algorithms such as greedy coloring, and check the vertex's color one by one. These algorithms proceed in synchronized steps and use the threads to work on the active vertices. A crucial attribute of a synchronous computing model is the number of synchronized steps. An algorithm that has a less number of synchronized steps delivers the better performance.

In the sequential spread model, each synchronized step can be divided into three phase. In the first phase, the colored vertices need to send their colors to their neighbouring vertices. In the second phase, the neighbours receive the message about the colors and then in the third phase, the neighbours execute the local computation. However, we observed from our benchmarking experiments that although the number of synchronized steps is large, there are only a small number of active vertices in each step. Moreover, it is difficult to run this execution model in parallel, because the active vertices in cur- rent iteration can be colored only when the results of previous iterations are known.

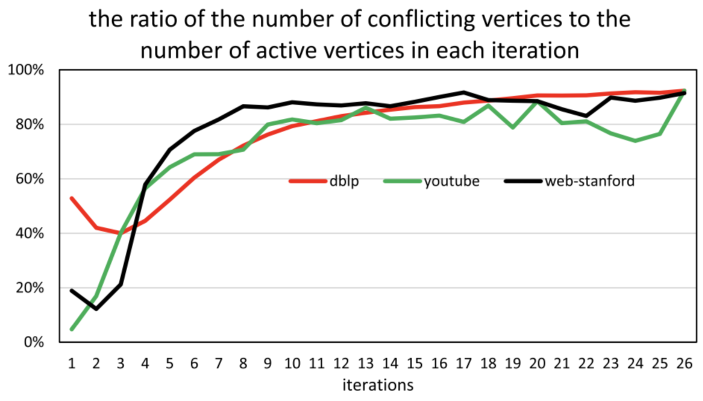
The other type of graph coloring algorithms employs the recursive execution, which works in a similar way as PageRank. High performance recursion-based graph coloring algorithms have been developed in recent works. In this coloring model, every vertex is assigned a color at the beginning. Then every vertex sends its color value to its neighbours. Next, the neighbouring vertex updates its color if the color conflicting occurs (i.e., the neighbouring vertices have the same color).

The colors of most vertices converge in the first few iterations. The remaining small fraction of vertices take a long time to converge to the final color due to color conflicts. This long-tail phenomenon can be demonstrated from the following benchmark experiments. We ran the experiments on a NVIDIA Tesla P100 GPU, which is equipped with 16 GB on-board memory, 3,584 CUDA cores, and Red Hat Enterprise Linux Server release 6.2 (Linux version 3.10.0-514.el7.x86 64). As shown in Fig. 1a, the number of vertices that are colored in each iteration drops dramatically in the first few iterations (i.e., these vertices converge to their final colors) and only a very small fraction of vertices are active in a large number of remaining iterations. Take the Youtube dataset as an example. There are 1,157,828 vertices in total. 1,148,571 vertices converge in the first 2 iterations, while the remaining 9,257 vertices take more than 30 iterations to con- verge. We also plot the ratio of the number of conflicting vertices to the number of active vertices in each iteration, shown in Fig. 1b. The ratio is only 4 percent in the first iteration, and increases to 92.3 percent in the 26th iteration. These results suggest the recursion algorithm is very effective in the first few iterations, but does not conduct much useful work in the remaining iterations.

We have shown that a single coloring strategy in the entire execution may be problematic because the conflict always exists and can be dramatically different in each iteration. This motivates us to develop an adaptive and hybrid algorithm for graph coloring, where the coloring mode changes adaptively and automatically at different stages, aiming to achieve the best overall performance.



(a) Colored vertices



(b) Vertices conflict rate

Figure 1: The number of colored vertices (Fig. 1a) and the ratio of the number of conflicting vertices to the number of active vertices (Fig. 1b) as the graph coloring process progress through each iteration.

3 Evaluation

In this section, we present the results of experimental evaluation for different design choices and compare the state-of-the-art graph coloring techniques each other on both power-law and random graphs.

Datasets	Vertices	Edges	Direction
Stanford	281,903	2,312,497	Directed
dblp	986,207	6,707,236	Undirected
youtube	1,157,828	2,987,624	Undirected
RoadNet	1,971,282	5,533,214	Undirected
Wiki	2,394,385	5,021,410	Undirected
soc-lj	4,847,571	68,993,773	Directed
RMAT	9,999,993	160,000,000	Undirected
random	19,999,888	100,000,000	Undirected
twitter	41,652,230	1,468,365,182	Directed
webbase	118,142,155	1,019,903,190	Directed

Table 1: Datasets Used in the Experiments

3.1 Experimental Setup

We have conducted the experiments with both directed and undirected graphs. (u,v) represents the undirected edge between vertices u and v while ¡ u,v ¿ represents a directed edge from u to v.

We have carried out the experiments on total of 10 different graphs, 8 real-world graphs and 2 synthetic graphs as detailed in Table 1. The vertex degree among all graphs ranges from 2 to 106. Synthetic graphs RMAT and random are generated using PaRMAT and have the random degree distribution. The twitter and webbase are shared in the Laboratory for Web Algorithmics (LAW) and the remaining graphs are obtained from Stanford Network Analysis Project (SNAP).

We conduct the experiments on a NVIDIA Tesla P100 GPU, which is a Pascal architecture-based GPU equipped with 16 GB on board memory and 3,584 CUDA cores. The GPU is coupled with host machine equipped with 2 Intel(R) Xeon(R) E5-2670 CPUs, each at 2.60 GHz, and 8 GB memory. The host machine is running RedHat OS version 4.4.5-6. The algorithm is implemented with C++ and CUDA 9.0 using the "-arch=-sm35" compute compatibility flag.

3.2 The Algorithms for Comparison

We compared three state-of-the-art methods in the experiments, which are Kokkos, JPL and cuSPARSE. All these three methods are implemented on NVIDIA GPU. We describe some operation details of these three methods as follows.

- Kokkos. Kokkos uses a first-fit policy to assign color for the vertices. In the first-fit algorithm, a large FORBID array is used to store the colors of the neighbors of the current coloring vertex. Namely, the current coloring vertex cannot choose the colors from the FORBID array. This method can achieve good processing speed. But the FORBID arrays of the large degree vertices consume large memory space. On the other hand, if a small FORBID array is used for the large degree vertices, multiple memory accesses are then needed, which slows down the coloring process.
- Jones-Plassmann-Luby (JPL) Graph Coloring Algorithm. The JPL coloring algorithm uses an approximate maximal independent set policy to partition the vertices into several sets and assign the colors to each set. This coloring policy uses the iteration approach, which can be trapped in the long-tail problem easily.
- cuSPARSE. The cuSPARSE library is developed by NVIDIA. The coloring implementation of cuSPARSE follows a csrcolor routine by using a multi-hash method to find the independent sets. Similar as the JPL graph coloring algorithm, it is easy to be trapped in the long-tail problem.

3.3 Recursion Versus Sequential Spread

Table 2 shows that recursion only coloring method can achieve better runtime performance as compared to sequential spread only coloring method, but it tend to use more colors on some datasets. As we explained this problem clearly in Section 2, this phenomenon occurs because that only active vertices updated in sequential spread only processing model, while all the vertices are updated in the recursion only coloring method. While the vertices are colored permanently once the color is choosed in recursion only coloring model, but the colors may changed in later iterations in recursion only coloring method. We can also conclude that random graph can be colored in fewer iterations under recursion only coloring method, because all the degrees of the vertices are change in a narrow space which can better suitable for GPU SIMD processing model.

Table 2: Execution Time for Different Coloring Algorithms (in Milliseconds)

Datasets	**Recursion Only**		**Sequential Spread Only**	
	Time	**Color**	**Time**	**Color**
Stanford	8.696	115	98.152	113
dblp	49.876	255	339.137	120
youtube	27.792	167	172.035	45
RoadNet	30.438	110	183.646	6
Wiki	129.233	112	272.397	97
soc-lj	493.387	490	5002.41	329
RMAT	1892.932	82	7989.48	78
random	3431.787	89	12496.272	84
twitter	15319.55	1189	51823.1	910
webbase	10438.999	1650	186029.255	1507

3.4 Experiment Results

We designed a set of experiments for different data- sets with different value of fractions. The execution time of these two stages with different fraction values are shown in Fig. 5. The left side y-axis in Fig. 5 is the coloring time in milliseconds, while the y-axis at the right side is the number of colors. The red line with gray diamond dots shows the total coloring time of Feluca, while the black line and the green line show the coloring times of the recursion and the sequential spread stage, respectively. The number of colors is shown by the yellow line with triangle dots.

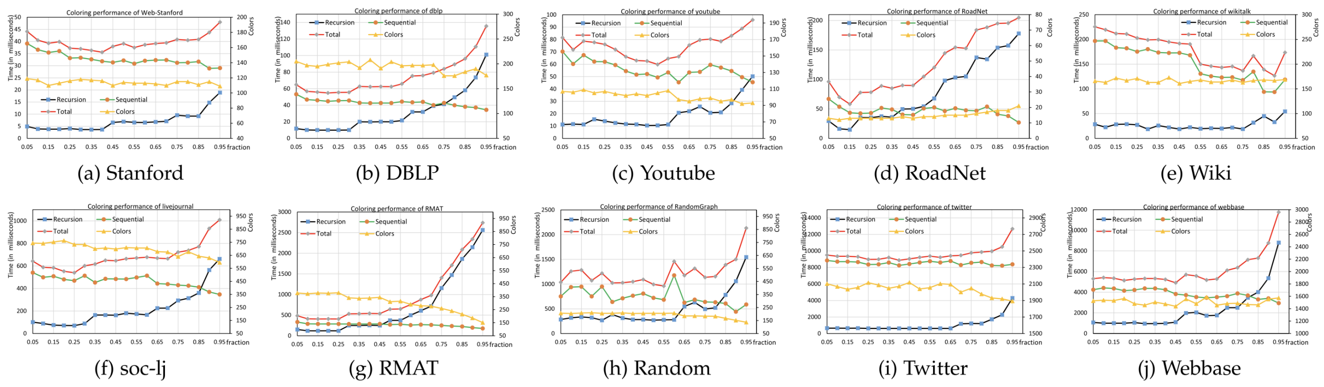


Figure 2: Coloring time with different fraction. X-axis is the value of fraction, the Y-axis on the left is the coloring time in milliseconds, while the Y-axis on the right is the number of colors. "Sequential" means the time spent by the sequential spread stage of the coloring algorithm while "recursion" means the time by the recursion stage. The parameter fraction indicates the ratio of the number of colored vertices in the recursion stage to the number of total vertices in the graph.

The sequential spread stage is most time consuming with a small fraction value, which means there are very few vertices colored in the recursion stage. Fig. 5 shows that for all the power-law graphs, the execution time of the recursion stage is much smaller than that of the sequential spread stage with a small fraction value, which means the recursion algorithm is much faster than the sequential spread algorithm. On the contrary, the recursion method needs more time with a big fraction value, which means there are more conflicts occurred at the end of the recursion stage.

However, after a majority of vertices find the suitable colors, these colored vertices will have impact on the colors of the remaining ver- tices. This causes a small number of remaining vertices to change their colors repeatedly in later iterations and there- fore slows down the progress.

4 References

[1] D. Marx, "Graph colouring problems and their applications in scheduling," Periodica Polytechnica Elect. Eng., vol. 48, no. 1/2, pp. 11–16, 2004.

[2] G. J. Chaitin, "Register allocation and spilling via graph coloring," U.S. Patent 4 571 678, Feb. 1986.

[3] A. M. Herzberg and M. R. Murty, "Sudoku squares and chromaticpolynomials,"NoticesAmer.Math.Soc.,vol.54, pp.708–717,Jun./ Jul. 2007. [Online]. Available: http://www.ams.org/notices/ 200706/

[4] G. Chartrand and P. Zhang, Introduction to Graph Theory. New York, NY, USA: McGraw Hill Education, 2017.

[5] M. R. Garey and D. S. Johnson, "The complexity of near-optimal graph coloring," J. ACM, vol. 23, no. 1, pp. 43–49, Jan. 1976. [Online]. Available: http://doi.acm.org/10.1145/321921.321926

[6] X. Zhang, G. Tan, S. Xue, J. Li, K. Zhou, and M. Chen, "Understanding the GPU microarchitecture to achieve bare-metal performance tuning," in Proc. 22nd ACM SIGPLAN Symp. Princ. Practice Parallel Program., 2017, pp. 31–43. [Online]. Available: https://doi.org/10.1145/3018743.3018755

[7] K. Meng, J. Li, G. Tan, and N. Sun, "A pattern based algorithmic autotuner for graph processing on GPUs," in Proc. 24th Symp. Princ. Practice Parallel Program., 2019, pp. 201–213. [Online]. Avail- able: https://doi.org/10.1145/3293883.3295716