Deterministic Parallel Graph Coloring with Repartitioning by Auxiliary Graph Coloring

Johan Öfverstedt, Per Normann

December 2015

Abstract In this paper we propose a deterministic parallel graph coloring algorithm that enables Multi-Coloring in parallel for sparse undirected graphs by coarse-grained segmentation and auxiliary graph coloring. The proposed algorithm is implemented and tested on standard problem instances from engineering applications and benchmarked against various relevant deterministic graph coloring algorithms. Quantified results show that the proposed algorithm is competitive or better than the sequential Multi-Coloring algorithm with respect to execution time on multi-core architectures. The upper bound on the number of colors is guaranteed to be the same as for the Multi-Coloring algorithm.

1 Introduction

In computing and numerical science the vertices and edges of a graph are commonly denoted as V and E, the degree of a graph as Δ. Graph coloring is the problem of associating every vertex with a color such that each pair of adjacent vertices are assigned distinct colors. Colors are generally denoted by integers. In Figure 1.1 the vertices have been assigned integers according to the constraints of graph coloring.

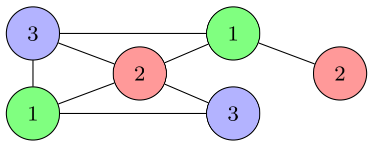

Figure 1.1: A graph with vertices colored such that no adjacent vertices share the same color. Colors are generally denoted by integers.

Finding an optimal coloring that minimizes the number of colors is a NP-hard problem and therefore intractable in general, requiring the use of heuristics [1]. Contemporary applications employing deterministic graph coloring predominantly implements some variation of the Multi-Coloring algorithm [2]. The Multi-Coloring algorithm constitutes a formidable benchmark for graph coloring both in execution time and the quality of the coloring [3].

The Multi-Coloring algorithm is sequential and does not allow for efficient use of contemporary multi-core hardware. Prior to this work we have proposed two different deterministic parallel graph coloring algorithms [4][5] for multi-core architectures. In this paper we propose a new deterministic parallel graph coloring algorithm for sparse undirected graphs.

2 New Graph Coloring Algorithm

In this paper we propose a deterministic parallel graph coloring algorithm with repartitioning by auxiliary coloring for undirected graphs. Shared memory architecture and a data structure supporting partitioning without an initial traversal are assumed. The algorithm consists of two main stages. In the first stage final colors are assigned to internal vertices and auxiliary colors to interface vertices. The distinction between internal and interface vertices are made in the same manner as in [5]. Internal vertices have all adjacent vertices in the same partition, interface vertices have at least one adjacent vertex outside the own partition. In the end of the first stage, sets of independent vertices are formed based on the auxiliary coloring, e.g. one set for each auxiliary color partition-wise. These sets are mutually independent since by the definition of graph coloring no vertex with an arbitrary color c is adjacent to any other vertex with the same color c. Producing the final color for a given vertex v only requires performing reads on its adjacent vertices and self mutation hence it is safe to coloring them in parallel. This is done in the second and final stage, where the sets of auxiliary colored vertices are processed one by one with synchronization. The vertices of these sets are distributed over processors and colored.

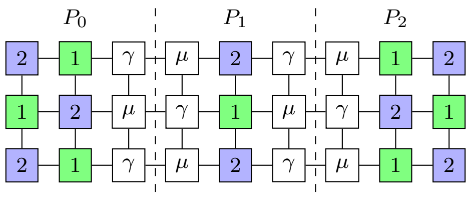
P_0
P_1
P_2
2 1 γ μ 2 γ μ 1 2
1 2 μ γ 1 μ γ 2 1
2 1 γ μ 2 γ μ 1 2

Figure 2.1: The initial stage of the proposed algorithm, internal vertices are assigned final colors and interface vertices are assigned colors from an auxiliary set of colors, here denoted by γ and μ.

In Figure 2.1 the initial stage of the proposed algorithm is outlined. All internal vertices are assigned final colors and interface vertices are assigned auxiliary colors. The algorithm then proceeds by coloring all vertices in partition P_0 with the auxiliary color γ in parallel. The algorithm is deterministic in the sense that the coloring is always the same for a given graph and initial partitioning. The complexity is $O(|V| + |E|)$ for both the initial stage presented as pseudo code in Algorithm 1 and the finalizing stage in Algorithm 2. In order to maximize the number of finalized vertices in the initial stage, symmetry breaking can be implemented [5].

Algorithm 1 Auxiliary Coloring Algorithm: Initial Stage

```
 1: for $\forall v_i$ do                          ▷ In parallel over initial partitions $P_n$.
 2:     $C \leftarrow \varnothing$                ▷ Reset auxiliary and final color set $A$ and $C$.
 3:     $A \leftarrow \varnothing$
 4:     for $\forall v_{i,j}$ adjacent to $v_i$ do
 5:         if $v_{i,j} \notin P(v_i)$ then       ▷ Form sets of forbidden colors.
 6:             $A \leftarrow a_{i,j} \cup A$
 7:             Flag $v_i$ as unsafe.
 8:         else
 9:             $C \leftarrow c_{i,j} \cup C$
10:         end if
11:     end for
12:     if $v_i$ is unsafe then                   ▷ Assign auxiliary or final color to $v_i$.
13:         $a_i \leftarrow \min(a \notin A)$     ▷ Form sets of independent vertices.
14:         $A_{P_n,a_i} \leftarrow v_i \cup A_{P_n,a_i}$
15:     else
16:         $c_i \leftarrow \min(c \notin C)$
17:     end if
18: end for
19: Synchronization.
```

Algorithm 2 Auxiliary Coloring Algorithm: Finalizing Stage

```
 1: for $\forall P_n$ do                          ▷ $P_n$ are the initial partitions.
 2:     for $\forall A_{P_n,k}$ do                ▷ $A_{P_n,k}$ are sets of auxiliary colored vertices.
 3:         for all $\forall v_i \in A_{P_n,k}$ do                        ▷ In parallel.
 4:             for $\forall v_{i,j}$ adjacent to $v_i$ do
 5:                 $C \leftarrow c_{i,j} \cup C$ ▷ Form sets of forbidden colors.
 6:             end for
 7:             $c_i \leftarrow \min(c \notin C)$
 8:         end for
 9:         Synchronization.
10:     end for
11: end for
```

3 Implementation and Test Configuration

For testing and benchmarking the algorithms in Table 3.1 are implemented in plain C/C++ using the low-level pthreads-library for multi-threading within a single testing framework and compiled with g++ version 4.4.7. The framework and implementations are crafted for efficiency and performance to reduce noise and systematic error from the measured run times and resulting speed-up calculations.

Table 3.1: Algorithms considered in the test.

Sequential greedy Multi-Coloring
Symmetry breaking algorithm
Proposed auxiliary coloring algorithm

Among the algorithms listed in Table 3.1 the sequential greedy Multi-Coloring algorithm is chosen because it is a standard benchmark in graph coloring [3]. The symmetry breaking algorithm is chosen due to the promising results presented in [5].

The test matrices used in this work are found on the matrix server[1] of the Department of Computer and Information and Engineering at the University of Florida. A summary of the features of the test matrices is presented in Table 3.2.

Table 3.2: Specifications of test graphs. The number of edges E and vertices V are scaled 10^{-6}, Δ is the degree of the graph.

	Graph specifications.			
Matrix	V	E	Δ	$avg\Delta$
apache2	0.7	4.8	7	5.7
bone010	1.0	71.7	80	71.6
ecology2	1.0	5.0	4	4.0
Flan_1565	15.6	117.4	80	74.0
G3_circuit	1.6	7.7	5	3.8
Geo_1438	1.4	63.2	56	41.9
Hook_1498	1.5	60.9	92	39.7
ldoor	1.0	46.5	76	46.9
msdoor	0.4	20.2	76	46.7
offshore	0.3	4.2	30	15.3
parabolic_fem	0.5	3.7	6	6.0
pwtk	0.2	11.6	179	52.4
Serena	1.4	64.5	248	45.4
StocF-1465	1.5	21.0	188	13.3
thermal2	1.2	8.6	10	6.0

All test matrices in Table 3.2 are colored using the algorithms in Table 3.1 and the results are compared with respect to execution time and color quality.

The test graphs are converted to the CSR-format, which is described in [2], prior to the coloring procedure. In our implementation this ensures good access and locality of rows in test matrices, i.e. the sets of neighbouring vertices of each vertex. This choice of graph data format is also backed up by its ubiquity in numerical applications, its compact representation and its good memory locality [2].

The heuristic selected to color vertices is the greedy Multi-Coloring algorithm. One of the major implementation decisions for this heuristic is the data structure used to maintain the set of neighbouring colors while searching a vertex's neighbourhood to compute its color. We implement a `BITSET` structure for this, thereby taking advantage of the target hardware platform's ability to efficiently store and operate on 64 colors as a single 64-bit integer due to it being a 64-bit architecture. This yields worst case time complexity of $O(1)$ for the `Insert`-operation and $O(\Delta)$ for the `FindFirst`-operation. `FindFirst`, although linear in complexity, has a small constant given the previously discussed implementation which is important for the practical purpose of being used in this algorithm. The internal bookkeeping memory required to operate this data structure is $O(\Delta)$.

[1] http://faculty.cse.tamu.edu/davis/matrices.html - Accessed: 2015-11-07

The auxiliary coloring is implemented by storing vertices in dynamic arrays, one dynamic array for each initial partition and auxiliary color. By implementing symmetry breaking [5] for the first coloring round, some vertices do not require auxiliary coloring but can be assigned their final color. In particular all vertices in the first segment can be finalized.

Without imposing constraints on the number of auxiliary colors and the processing of their related sets of vertices the number of synchronization rounds can be very large, hence hurt the performance. We have implemented two techniques to obtain a reasonable upper bound on the number of synchronization rounds.

- A fixed upper bound is placed on the number of auxiliary colors. Above this threshold vertices are sorted into a special set for sequential processing.
- A parallel threshold is also introduced on the number of vertices in each auxiliary colored set, below which the whole set will be sequentially colored without synchronization. A threshold of 32 is used in this implementation.

In the final coloring stage, consecutive ranges of vertices from the auxiliary colored sets are distributed evenly over processors. This type of slicing is employed to maximize memory locality and prevent false sharing.

The tests are preformed on the machine Tussilago at the Division of Scientific Computing at Uppsala University, CPU: AMD Opteron (Bulldozer) 6282, 2.6 GHz, 16-cores, dual socket. The RAM of the hardware is 128GB, the CPU cache hierarchy levels are listed in Table 3.3.

Table 3.3: The size of the hardware's cache in kilobytes.

Cache level	Size
L1	64
L2	2048
L3	6144

The selected test problems are large enough to heavily reduce significance of timer precision noise in the measured execution times. Time measurements are implemented by enclosing the algorithm with calls to `gettimeofday`.

Empirical correctness testing consists of a conflict and invalid color detection procedure which is executed after every run. Determinism and consistency between runs are verified by generating a hash-code from the resulting color sequence.

4 Results

In Table 4.1 a comparison between the proposed algorithm, a sequential Multi-Coloring scheme and a symmetry breaking deterministic parallel graph coloring scheme is presented. The benchmarks presented regarding execution times are the best result out of 3 runs. The minimal overhead for both parallel algorithms while executed on one thread is achieved by falling back on sequential greedy multi-coloring.

Table 4.1: Performance of the proposed algorithm compared to implementations of sequential Multi-Coloring and symmetry breaking algorithm. Time is in milliseconds.

	Multi-Coloring		Symmetry Breaking		Proposed algorithm	
Matrix	*Time*	*Colors*	*Time**	*Colors**	*Time**	*Colors**
apache2	22	3	21/16/9	3/4/4	20/15/10	3/4/4
bone010	173	39	178/120/67	39/45/48	180/99/55	39/45/48
ecology2	28	2	27/19/10	2/4/4	25/19/13	2/4/4
Flan_1565	278	42	278/213/112	42/42/45	295/164/88	42/45/48
G3_circuit	44	4	45/41/24	4/4/5	47/34/21	4/4/4
Geo_1438	175	29	180/106/58	29/33/33	187/100/58	29/33/33
Hook_1498	165	30	169/103/55	30/33/33	176/94/55	30/33/33
ldoor	122	42	130/140/127	42/42/42	132/112/65	42/42/42
msdoor	54	42	54/56/59	42/42/42	58/43/30	42/42/42
offshore	16	12	17/15/15	12/13/13	18/13/11	12/13/13
parabolic_fem	17	5	17/30/30	5/5/6	19/24/18	5/5/5
pwtk	30	48	31/21/13	48/48/48	32/19/11	48/48/48
Serena	171	36	178/122/73	36/39/36	182/170/67	36/38/36
StocF-1465	79	11	80/55/31	11/11/11	83/47/29	11/11/12
thermal2	49	7	52/43/30	7/7/7	55/34/21	7/7/7

* Figures presented as: 1/2/4-threads.

Figure 4.1 displays the speed-up achieved by the proposed algorithm for undirected sparse graphs. On the left side graphs that showed good speed-up is presented and on the right hand side graphs with minor speed-up.

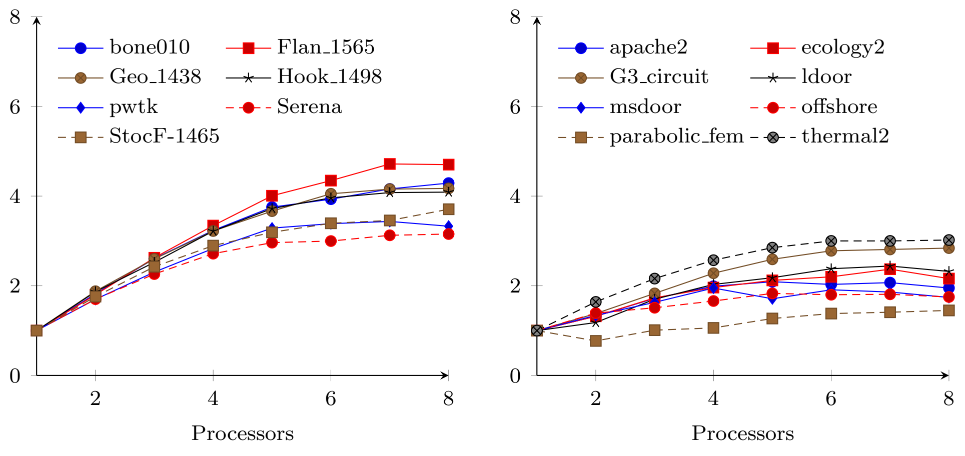

Figure 4.1: Parallel speed-up achieved by the proposed algorithm plotted with trend lines.

Compared with other schemes such as [4][5] the proposed algorithm obtains speedups for the test graphs greater than 1 with a smaller variance.

5 Conclusion

A new high performance deterministic parallel graph coloring algorithm for sparse undirected graphs is proposed in this paper. The proposed algorithm enables greedy multi-coloring in parallel using repartitioning by auxiliary coloring. The results show that the proposed algorithm is a plausible replacement for the sequential multi-coloring algorithm in practical applications executed on contemporary mainstream hardware.

Bibliography

[1] Garey, M.R. Johnson, D.S., “Computers and intractability,” *Freeman W.H. New York*, 1979.

[2] Saad, Y., “Iterative methods for sparse linear systems,” *Society for Industrial and AppliedMathematics*, 2003.

[3] Allwright, R. Bordawekar, P.D. Coddington, K. Dincer, C. Martin, A, “A comparison of parellel graph coloring algorithms,” *Cite Seer*, 1979.

[4] Normann, P. Öfverstedt, J., “Deterministic parallel graph coloring with hashing,” *Technical report series at Division of Scientific Computing, Department of Information Technology, Uppsala University*, 2015.

[5] Normann, P. Öfverstedt, J., “Deterministic parallel graph coloring with symmetry breaking,” *Technical report series at Division of Scientific Computing, Department of Information Technology, Uppsala University*, 2015.