Deterministic Parallel Graph Coloring with Hashing

Per Normann, Johan Öfverstedt

June 2015

Abstract In this paper we propose a new deterministic parallel graph coloring algorithm. Parallelism is achieved by distribution of vertices to processors by hashing. The hashing is based on markers assigned to each conflict prone vertex.

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.

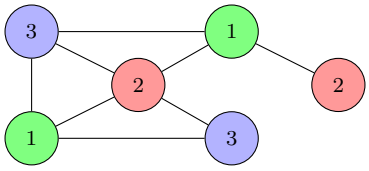


Figure 1.1: A graph with vertices colored such that no adjacent vertices share the same color. Colors are generally denoted by integers.

Finding an optimal coloring that minimizes the number of colors is a NP-hard problem and therefore intractable in general, requiring the use of heuristic methods [1]. Contemporary applications utilizing graph coloring predominantly implements some variation of the Greedy Multi-Coloring algorithm [2]. The reason to opt for the Greedy Multi-Coloring algorithm is that it is currently known as the best performing graph coloring algorithm and it creates a formidable benchmark, both in execution time and the quality of the coloring. Saad presents the Greedy Multi-Coloring [3] algorithm in detail.

The Greedy Multi-Coloring algorithm is sequential and does not allow for efficient use of modern multi-core hardware. Among previous attempts to color graphs in parallel there are two main categories. First there are Luby methods [4] where independent sets of vertices are formed in various ways to be safely colored in parallel. The second category is algorithms where a graph is tentatively colored in parallel followed by conflict resolution, this approach seems to have been proposed for the first time in [5]. The overall performance of these algorithm categories does not put the Greedy Multi-Coloring algorithm into obsolescence. Recently a new graph coloring scheme was presented in [6], which outperforms the sequential Greedy Multi-Coloring algorithm on multi-core processors. This recent attempt to parallelize graph coloring is however stochastic, hence not suitable for the graph analysis step in computational pipelines where reproducible results are required by design.

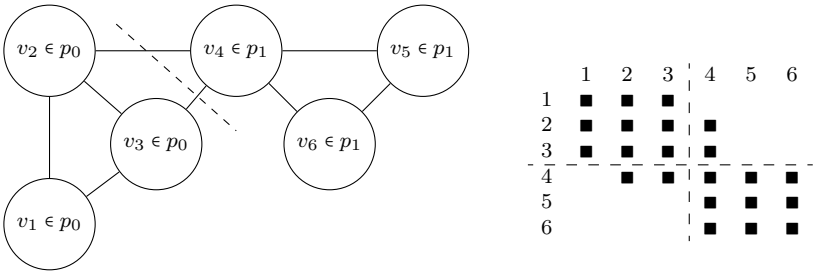
In this paper a deterministic parallel graph coloring algorithm for shared memory architecture is proposed. This new parallel graph coloring algorithm utilizes techniques of achieving parallelism that we have not been able to find in any published work.

2 New Graph Coloring Algorithm

In this paper a new parallel graph coloring algorithm is proposed, the Normann Öfverstedt First Fit algorithm, for convenience henceforth called NÖFF. NÖFF is an algorithm for undirected graphs assuming a shared memory programming model and a graph data structure supporting an initial partitioning without traversing the graph. The general scheme is an inner loop where safe vertices are colored in parallel with an arbitrary heuristic and an outer loop that runs until all vertices have been colored. Between each outer iteration the remaining vertices are distributed among processors.

A key feature of the NÖFF algorithm is how the distinction between internal and interface vertices is made. Processors are assigned subsets of vertices to color in parallel. Since shared memory architecture is assumed, processors have access to all vertices and need to make the distinction between vertices that are safe and unsafe to color in parallel, in order to avoid non-determinism and race conditions. Vertices with all its adjacent vertices in its own subset are internal and safe to color in parallel. If a vertex has adjacent vertices in a proxy subset it is situated in the interface, i.e. the cross section between subsets. These vertices can not be safely colored in parallel. By forming explicit cross sections of subsets, coherent sets of interface vertices are formed. In the subsequent iteration these coherent sets are distributed among processors through a hashing scheme.

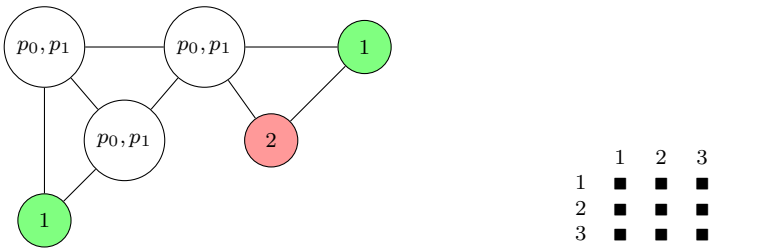
In Figure 2.1 the set-up phase of one outer iteration of the NÖFF algorithm is outlined. A graph is divided into the subsets $S_0 = (v_1, v_2, v_3)$ and $S_1 = (v_4, v_5, v_6)$. S_0 and S_1 are distributed to the processors p_0 and p_1 respectively.



(a) Graph (b) Connectivity matrix

Figure 2.1: In the set-up phase of one iteration the graph is divided and distributed. The dashed lines illustrate how subsets are formed in the graph (a) and how the distinction between interface and internal vertices is made in the matrix (b).

In the next phase of one outer iteration internal vertices of S_0 and S_1 i.e. v_1, v_5, v_6 are colored. Vertices in the cross section of S_0 and S_1 are added to an updated subset $\hat{S}_0$, in this example $\hat{S}_0 = (v_2, v_3, v_4)$. In the subsequent iteration $\hat{S}_0$ is mapped to a processor depending on the explicit identifiers linked to S_0 and S_1.



(a) Graph (b) Connectivity matrix

Figure 2.2: In the coloring phase of one outer iteration of the NÖFF algorithm the processors p_0 and p_1 color internal vertices and assign interface vertices into updated subsets. In the subsequent iteration all vertices in a subsets are handled by the same locigal thread.

The NÖFF algorithm is deterministic in the sense that the coloring is always the same for a given graph and initial partitioning. The number of iterations required to produce a proper graph coloring is dependent on the structure of the graph. There is no inherent symmetry breaking feature in the NÖFF algorithm, hence a fallback to sequential coloring is needed to guarantee termination. The complexity of each outer iteration is $O(E + V)$.

In a graph with a low degree, i.e. a low number of interconnections, the number of internal vertices is generally higher in each iteration, hence fewer outer iterations are required. The opposite often holds for a graph with a high degree. In addition to this the number of vertex connections between multiple partitions also influence the number of outer iterations.

In Algorithm 1 the NÖFF algorithm is presented in pseudo code.

Algorithm 1 Normann Öfverstedt First Fit

```
1: while ∃c_i = ∅ do
2:     for ∀V_i do                                  ▷ In parallel over subsets S.
3:         if c_i = 0 then
4:             C ← ∅                                ▷ Reset interface set I and color set C.
5:             I ← ∅
6:             for ∀V_j adjacent to V_i do
7:                 if v_j ∉ S(v_i) then
8:                     I ← S(v_j) ∪ I               ▷ Form interface set I.
9:                 else
10:                    C ← c_j ∪ C                  ▷ Form set of forbidden colors C.
11:                end if
12:            end for
13:            if I ≠ ∅ then
14:                Ŝ(v_i) ←hash(S(v_i) ∪ I)         ▷ Subset updated for next iteration.
15:            else
16:                c_i ← min(c ∉ C)                 ▷ Color V_i with first color not in C.
17:            end if
18:        end if
19:    end for
20:    Synchronization.
21: end while
```

3 Implementation and Test Configuration

For testing and benchmarking the NÖFF algorithm is implemented in plain C/C++ using the low-level pthreads-library for multithreading. The implementation is fairly well optimised for efficiency and performance to reduce noise and systematic error from the measured run times and resulting speed-up calculations.

The graphs are converted to the CSR-format, which is described in [3], 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 [3].

The heuristic selected to color internal vertices in this work is Greedy Multi-Coloring. 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 $\mathrm{O}(1)$ for the `Insert`-operation and $\mathrm{O}(\Delta)$ 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 auxiliary memory required to operate this data structure is $O(\Delta)$.

The parallel graph algorithm proposed in this paper utilizes a hash function $H : N \rightarrow N$ to pseudo-randomly assign internal vertices to a common processor for the subsequent iteration. For this hashing the FNV-algorithm [7] is employed in our test implementation because it is widely used and experimentally has displayed good dispersion properties combined with low computational cost.

The hardware on which the tests are conducted is *Gullviva* at the Division of Scientific Computing at Uppsala University, CPU: AMD Opteron (Bulldozer) 6274, 2.2 GHz, 16-cores, dual socket.

Time measurements are implemented by enclosing the NÖFF algorithm with calls to `gettimeofday`. The test problems are selected to be large enough for the actual algorithm runtime to dominate timer precision issues and cache effects which can easily distort benchmarks if care is not taken.

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.

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

A new high performance parallel graph coloring algorithm is proposed in this paper. NÖFF is a deterministic algorithm designed for use on sparse matrices that are diagonally dense. All benchmarks presented here regarding execution time are the best result out of 3 runs. This is done to negate the effect of fluctuation caused by the workload on the open lab hardware.

The Table 4.1 contains benchmarks of the NÖFF algorithm. The table is organized such that the eight first matrices are diagonally dense. These test matrices exhibits parallel speed-up. The last seven matrices in the table are not diagonally dense. Judging by the execution time in parallel the NÖFF algorithm did not achieve speed-up on these matrices. For some of these problematic matrices the execution time is in fact hurt by the parallelization. The quality of the colorings is high for all test matrices with a slight trend of an increasing number of colors.

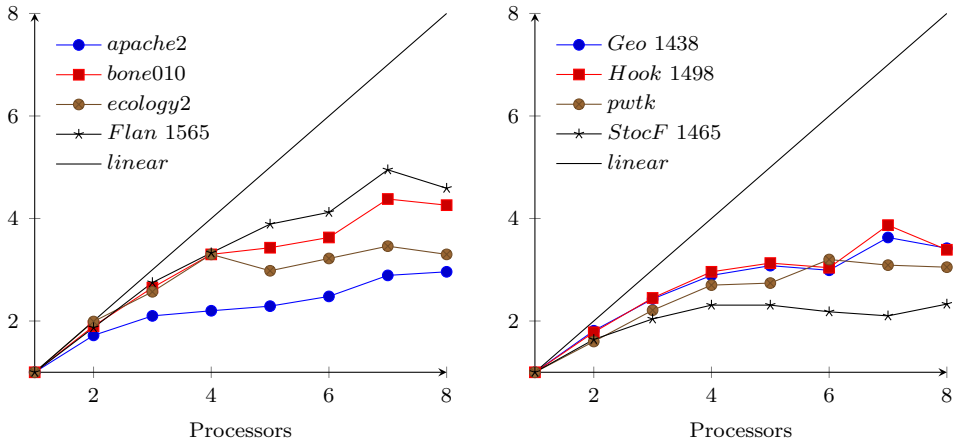
[1] http://www.cise.ufl.edu/research/sparse/MM/ - Accessed: 2015-06-08

Table 4.1: Specifications of test graphs and performance of the NÖFF algorithm. Time is in milliseconds, the number of edges E and vertices V are scaled $1/10^6$.

	Graph specifications.				Performance	
Matrix	V	E	Δ	$avg\Delta$	$Time^*$	$Colors^*$
apache2	0.7	4.8	7	5.7	50/22/16	3/4/4
bone010	1.0	71.7	80	71.6	288/117/91	39/45/45
ecology2	1.0	5.0	4	4.0	64/20/19	2/4/4
Flan 1565	15.6	117.4	80	74.0	613/184/133	42/45/45
Geo 1438	1.4	63.2	56	41.9	372/129/109	29/33/36
Hook 1498	1.5	60.9	92	39.7	365/123/108	30/33/33
pwtk	0.2	11.6	179	52.4	64/28/21	48/48/48
StocF 1465	1.5	21.0	188	13.3	184/80/79	11/11/12
G3 circuit	1.6	7.7	5	3.8	100/100/108	4/5/5
ldoor	1.0	46.5	76	46.9	275/294/478	42/49/49
msdoor	0.4	20.2	76	46.7	120/153/233	42/49/49
offshore	0.3	4.2	30	15.3	42/69/103	12/13/13
parabolic fem	0.5	3.7	6	6.0	43/237/300	5/6/6
Serena	1.4	64.5	248	45.4	376/173/303	36/39/39
thermal2	1.2	8.6	10	6.0	115/111/132	7/7/7

* Figures presented as: 1/4/8-threads.

Figure 4.1 displays the speed-up achieved for graphs with a diagonally dense sparse matrix. The speed-up on different graphs are alphabetically ordered and presented with trend lines.



(a) Matrices a-f (b) Matrices g-s

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

5 Discussion

In this work we show that an efficient deterministic parallel graph coloring of diagonally dense sparse matrices on a shared memory architecture is possible and an algorithm, NÖFF, is proposed. It should be noted that if the source graph is not sufficiently diagonally dense and sparse, performance degrades with respect to runtime. A desirable improvement for practical purposes would be to modify the algorithm variants to guarantee progress on every outer iteration such that the performance is similar to the sequential Multi-Coloring algorithm in the worst case. Future work could include investigating if this is possible while retaining the good performance seen on the favourable test-cases.

Extending the algorithmic ideas presented here to a distributed programming model would be another research topic worthy of investigation. Another variation to explore would be to combine the algorithmic framework with other orderings and heuristics than Memory Ordered First Fit, e.g. Largest First or Most Saturated First.

Bibliography

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

[2] Allwright, R. Bordawekar, P.D. Coddington, K. Dincer, C. Martin, A., "A comparision of parallel graph coloring algorithms," *Cite Seer*, 1995.

[3] Saad, Y., "Iterative methods for sparse linear systems," *Society for Industrial and AppliedMathematics*, 2003.

[4] Luby, M., "A simple parallel algorithm for the maximal independent set problem," *SIAM Journal on Computing*, 1986.

[5] Gebremedhin, A. H. Manne, F. Pothen, A, "Parallel distance-k coloring algorithms for numerical optimization," *EuroPar*, 2002.

[6] Normann, P., "Parallel graph coloring," *DIVA*, 2014.

[7] Fowler, Noll, Vo, "Fnv hash," 1991. http://www.isthe.com/chongo/tech/comp/fnv/ - Accessed: 2015-05-29.