COMPARISON OF HIGH-PERFORMANCE GRAPH COLORING ALGORITHMS

Lukas Gnam[a], Paul Manstetten[a], Siegfried Selberherr[b], Josef Weinbub[a]

[a]Christian Doppler Laboratory for High Performance TCAD at the
[b]E360 - Institute for Microelectronics

INTRODUCTION

The advent of modern many- and multi-core architectures offers the option of utilizing parallel computing to reduce the overall runtime of applications. An attractive option is to decompose a computational task into independent sets, which enables independent parallel processing, i.e., without computational dependencies which potentially limit parallel performance. Such techniques are typically used in, e.g., linear algebra [1], mesh adaptation [2], and community detection [3].
A prominent approach to identify independent sets is to use graph coloring algorithms, in particular so-called distance-1 algorithms [3], which color a graph $G(N,E)$ such that no two neighboring nodes have the same color. Subsequently, the nodes of each color represent an independent set. One drawback of most of these algorithms is the resulting skewness in the population of the independent sets, yielding possibly insufficient workload for the actual parallel processing steps which follow the graph coloring. In this work we compare the results of recently developed parallel distance-1 graph coloring algorithms against well known serial algorithms with respect to the number of colors, population sizes, and performance.

COLORING ALGORITHMS

The *Greedy* algorithm [3, 4] assigns each graph node the smallest permissible color by checking the already assigned colors of the neighboring nodes. An adoption of the *Greedy* algorithm, the *Greedy-LU* algorithm, alleviates the resulting skewness of the distribution of the color populations, by assigning the least used color to the active graph node [3]. In addition to the two serial algorithms, we investigate two shared-memory parallel algorithms: (a) The *Iterative Parallel* algorithm of Çatalyürek et al. [4], which conducts a parallel initial coloring with a subsequent color conflict detection and resolvement step resulting in an unbalanced coloring. (b) The *Scheduled Reverse* algorithm of Lu et al. [3], which generates a balanced coloring using a parallel recoloring approach after an initial *Greedy* coloring.

RESULTS AND DISCUSSION

As first test graph we use a graph representing a tetrahedral mesh of a three-dimensional tri-gate transistor (*Trigate*) with 177 093 nodes, a maximum number of node degree of 50, and an average node degree of 27.31. The second graph is from the University of Florida Sparse Matrix Collection [5], representing the internet topology (*Internet*) obtained from daily traceroutes in 2005 with 1 696 415 nodes, and a maximum and average degree of 23 633 and 8.72, respectively. All benchmarks were performed on a single node of the Vienna Scientific Cluster 3 [6].
In Figure 1 we depict the resulting color populations for the two investigated graphs. As expected, the *Greedy-LU* algorithm produces the best balancing in both cases, but uses the highest number of colors. For the *Internet* graph it requires about 6 times more colors than the unbalanced *Greedy* algorithm. The *Iterative Parallel* algorithm produces similar color populations as the *Greedy* algorithm. The *Scheduled Reverse* algorithm manages to alleviate the skewness resulting from the initial *Greedy*

coloring, but for the *Internet* graph it results in high population differences for higher color classes. In Table 1 we compare the execution times of all investigated algorithms. Our investigation shows that the *Scheduled Reverse* algorithm balances an initial *Greedy* coloring, but at the cost of being nearly two times slower for the *Internet* graph compared to the fastest algorithm (i.e., *Greedy*). Additionally, we show that the *Iterative Parallel* algorithm achieves a speedup of almost 5 for *Trigate* and about 11 for the *Internet* graph. The *Scheduled Reverse* algorithm performs worse in terms of speedup, mostly because it is based on an initial coloring, and an additional serial preparation step, before the actual parallel recoloring can be conducted.

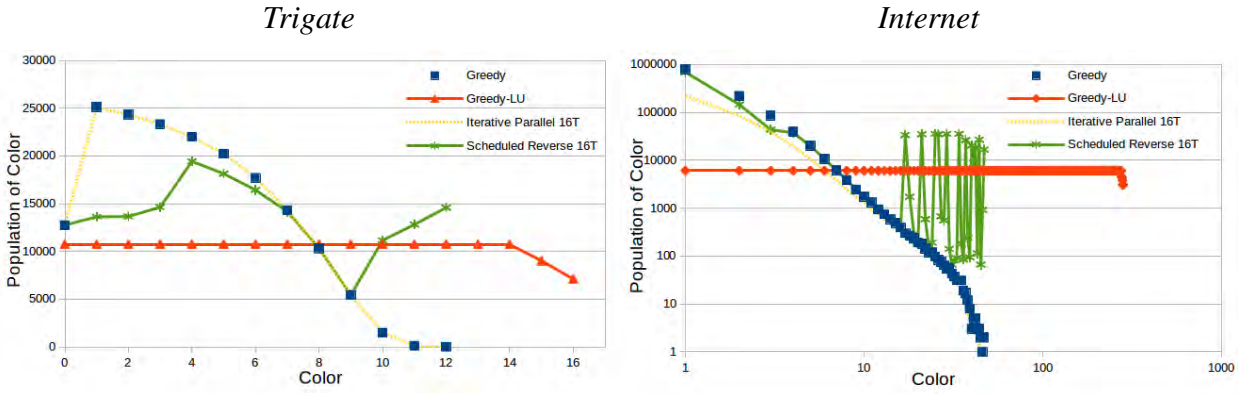

Figure 1: Color population of the two test graphs using 16 threads (16T) for the parallel algorithms.

Graph	*Greedy*	*Greedy-LU*	*It.Par. 1T*	*It.Par. 16T*	*Sched.Rev. 1T*	*Sched.Rev. 16T*
Trigate	0.033	0.096	0.129	0.027	0.062	0.045
Internet	0.462	6.352	5.502	0.492	1.017	0.800

Table 1: Execution times in seconds of the algorithms for the test graphs. For the *Iterative Parallel* (*It.Par.*) and *Scheduled Reverse* (*Sched.Rev.*) algorithms the results obtained with 1 (1T) and 16 threads (16T) are shown.

CONCLUSION

We showed that the *Greedy-LU* algorithm performs best regarding the balancing of the color populations, but can be nearly 14 times slower than the unbalanced *Greedy* algorithm. Adding execution time to the consideration, the *Iterative Parallel* algorithm produces the best results regarding both, parallel scalability and coloring quality.

ACKNOWLEDGMENT

The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged as is the support by the TU Wien IP project *"Parallel 3D Mesh Generation for Bio-Micro- & Nanoelectromechanical Systems"*. The computational results presented have been achieved using the Vienna Scientific Cluster (VSC).

REFERENCES

[1] M. Fratarcangeli *et al.*, ACM Trans. Graph., **35**(6), 214:1-214:9, 2016.
[2] L. Gnam *et al.*, Proc. of IMR, 2017.
[3] H. Lu *et al.*, IEEE Trans. Par. Dist. Sys., **28**(5), 1240-1256, 2017.
[4] Ü. Çatalyürek *et al.*, Par. Comp., **38**(10), 576-94, 2012.
[5] T.A. Davis *et al.*, ACM Trans. Math. Softw., **38**(1), 1:1-1:25, 2011.
[6] http://vsc.ac.at