Computing Strongly Connected Components in Parallel on CUDA

Jiří Barnat, Petr Bauch, Luboš Brim, and Milan Češka

Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract

The problem of decomposition of a directed graph into its strongly connected components is a fundamental graph problem inherently present in many scientific and commercial applications. We show how existing parallel algorithms can be reformulated in order to be accelerated by NVIDIA CUDA technology. We design a new CUDA-aware procedure for pivot selection and we redesign the parallel algorithms in order to allow for CUDA accelerated computation. We experimentally demonstrate that with a single GTX 280 GPU card we can easily outperform optimal serial CPU algorithm. This poster is based on the technical report [2].

1. Problem and Motivation

SCC decomposition and its application

- problem of decomposing a directed graph into its strongly connected components (a maximal set of vertices such that any two vertices are mutually reachable)
- many applications leading to very large graphs and requiring high performance processing
- web analysis based on web archives such as topic tracking, time-frequency analysis of blog postings, and web community extraction
- automated verification of hardware and software such as model checking, dataflow analysis, and bad cycle detection; SCC decomposition is used as a sub-procedure and its fast performance is crucial

Parallel SCC decomposition is a tricky problem

- optimal serial Tarjan's algorithm [7] strongly relies on depth first search post ordering of vertices whose computation is known to be P-complete and thus, difficult to be computed in parallel
- asymptotic complexity of the known parallel algorithms is not optimal
- multi-core implementations of the parallel algorithms for the standard parallel shared-memory platforms were not able to outperform Tarjan's algorithm

2. Parallel Algorithms for SCC Decomposition

- Forward-Backward Algorithm (FB) [5]
  - key steps of the algorithm are depicted in Figure 1
- Coloring/Heads-off algorithm (COLORING) [6]
- Recursive OBF algorithm (OBF) [4]

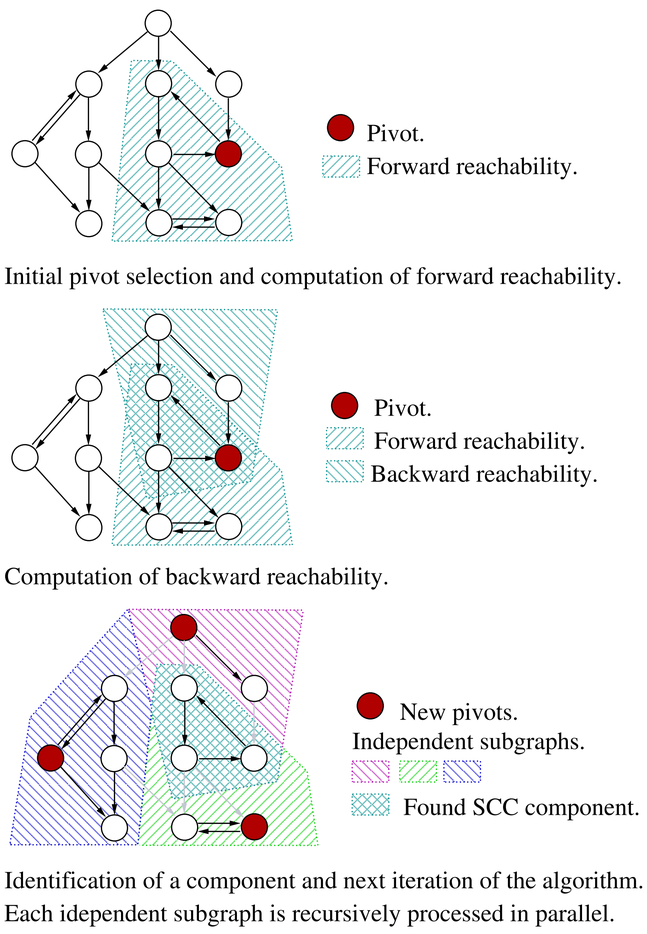

Figure 1: *Key steps of the Forward-Backward Algorithm.*

3. CUDA Accelerated SCC Decomposition

Graph representation

- appropriate adjacency list representation; dynamically linked adjacency list violates the memory requirements
- two one-dimensional arrays; one array stores the target vertices of edges sorted according to source vertex and the second array keeps indexes to the first edge emanating from the vertex (Figure 2)

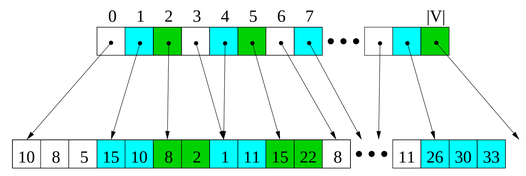

Figure 2: *Adjacency list representation.*

Redesign of the parallel algorithms

- reformulation of the recursion present in the algorithms by means of iterative procedures executed simultaneously over multiple independent subgraphs
  - forward and backward reachability restricted to the particular subgraphs
  - trimming - effective identification of trivial components by iterative elimination of vertices with no immediate predecessors or successors
  - pivot selection - selection of a single vertex for each subgraph; crucial for practical performance
- each procedure corresponding to one kernel - in the case of OBF algorithm more complicated parallelization is necessary to obtain significant speedup
- separate thread for every vertex is defined
- each thread checks if the corresponding vertex satisfies the given property, and if so, it sets the bit represeting the property for all immediate successors of the vertex
- algorithm performance is limited by memory bandwidth since for each vertex update only few instructions are executed

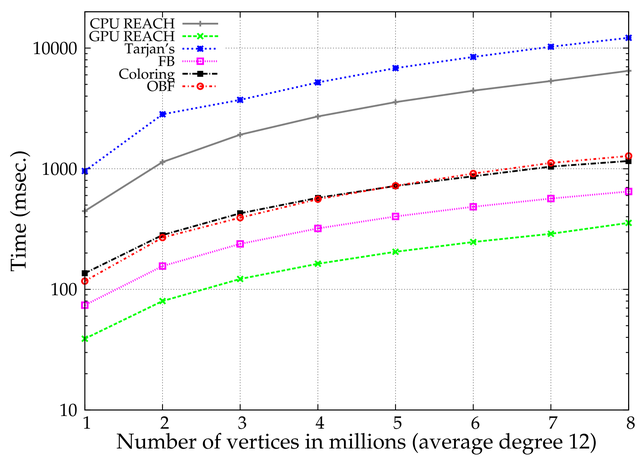

Figure 3: *Run-times for Random graphs in milliseconds.*

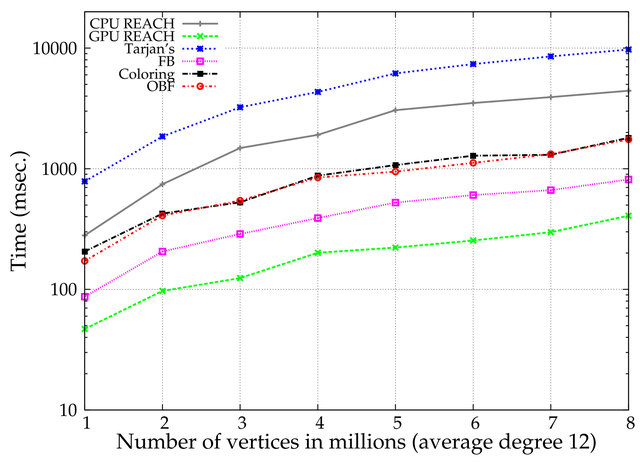

Figure 4: *Run-times for R-MAT graphs in milliseconds.*

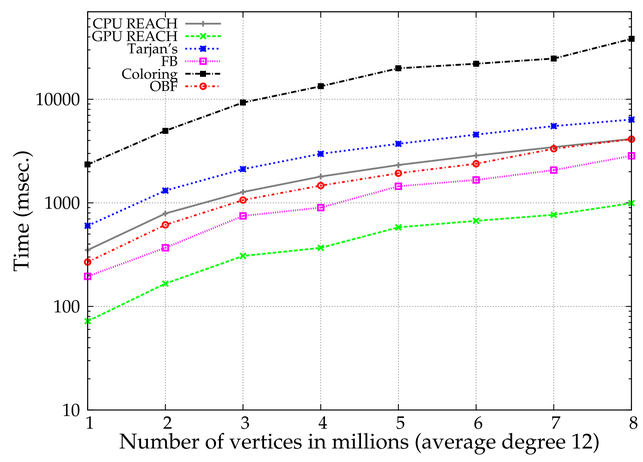

Figure 5: *Run-times for SCCA graphs in milliseconds.*

4. Experimental Evaluation

Hardware setting

- Linux workstation with a quad core AMD Phenom II X4 940 Processor @ 3GHz, 8 GB DDR2 @ 1066 MHz RAM
- NVIDIA GeForce GTX 280 GPU with 1GB of GPU RAM

Inputs graphs

- Georgia Technology graph generator suite [1]:
  - Erdös-Rényi random graph generator (Random)
  - Scalable Synthetic Compact Applications (SSCA)
  - Recursive Matrix generator (R-MAT)
- graphs produced by model checker DiVinE [3]

Comparison of the following algorithms is provided

- serial CPU-based forward reachability (CPU REACH)
- parallel CUDA-based forward reachability (GPU REACH)
- optimal serial Tarjan's algorithm
- CUDA-based FB algorithm (+ trimming)
- CUDA-based COLORING algorithm
- CUDA-based OBF algorithm (+ trimming, coloring)

Results of experimental evaluation

- run-times of the best versions of the individual algorithms are plotted in Figures 3, 4, 5 and 6
- performance of CUDA-based algorithms deeply depends on the average degree of the vertices in the graph
- the scalability and efficiency of the parallel reachability procedure effectively limit scalability and efficiency of SCC decomposition algorithms
- with a single GTX 280 GPU card the optimal serial Tarjan's algorithm can be easily outperformed
  - random graphs - 17 $\times$ speedup
  - R-MAT graphs - 12 $\times$ speedup
  - SCCA graphs - 3 $\times$ speedup
  - model checking graphs - 3 $\times$ speedup

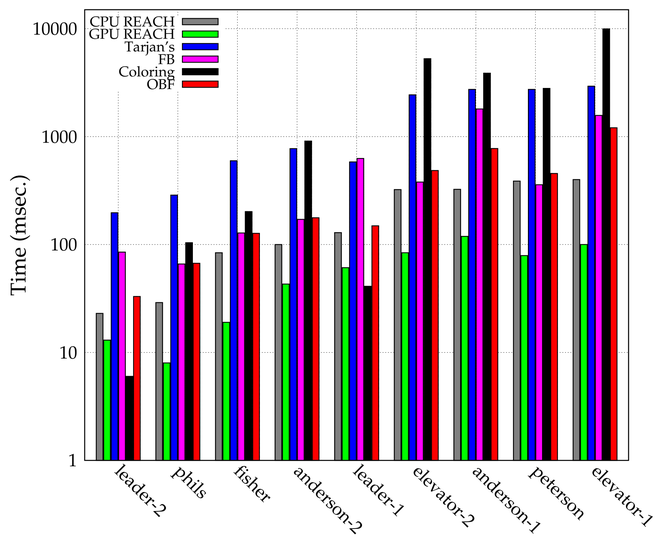

Figure 6: *Run-times for model checking graphs in milliseconds.*

- no clear winner among the particular parallel algorithms
- for the synthetic graphs (small number of large components) FB algorithm with trimming has the best times
- for model checking graphs (bigger number of large components) OBF algorithm has the best times
- COLORING algorithm exhibits unstable performance

5. Future Work

- effective utilization of multiple CUDA devices
- employing of the hierarchical memory of the upcoming generation of CUDA Fermi cards

References

[1] D.A. Bader and K. Madduri. GTgraph: A Synthetic Graph Generator Suite. Technical Report GA 30332, Georgia Institute of Technology, Atlanta, 2006.

[2] J. Barnat, P. Bauch, L. Brim, and M. Češka. Computing Strongly Connected Components in Parallel on CUDA (full version). Technical Report FIMU-RS-2010-10, Faculty of Informatics, Masaryk University, July 2010.

[3] J. Barnat, L. Brim, L. Černá, P. Moravec, P. Ročkai, and P. Šimeček. DiVinE – A Tool for Distributed Verification (Tool Paper). In *CAV '06*, volume 4144/2006 of *LNCS*, pages 278–281. Springer, 2006.

[4] J. Barnat, J. Chaloupka, and J. C. van de Pol. Improved Distributed Algorithms for SCC Decomposition. In *PDMC*, pages 65–80. University of Twente, 2007.

[5] L. K. Fleischer, B. Hendrickson, and A. Pinar. On Identifying Strongly Connected Components in Parallel. In *IPDPS '00*, volume 1800 of *LNCS*, pages 505–511. Springer, 2000.

[6] S. Orzan. *On Distributed Verification and Verified Distribution*. PhD thesis, Free University of Amsterdam, 2004.

[7] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.