A Typical Verification Challenge for the GRID

Jaco van de Pol

Formal Methods and Tools, Universiteit Twente, The Netherlands

1 Introduction

The task of detecting the strongly connected components of a very large graph is at the heart of many software verification procedures, such as model checking and state space minimization. In this context, the graphs represent system behaviour, i.e. all states of a system and all state transitions that are reachable from a given initial state. This so-called state space can be prohibitively large; often it only fits in the collective internal memory of a number of machines. Hence scalable distributed algorithms that work on partitioned graphs are required.

2 The Algorithm

Remember that a strongly connected component (SCC) in a graph is a maximal subgraph in which all two vertices v, w are connected via a path from v to w and vice versa. Tarjan's original algorithm [3] for detecting all SCCs is very efficient (linear time) and based on depth first search (DFS). This is unfortunate, because it is actually impossible to efficiently parallelize DFS. Therefore, parallel SCC detection algorithms are usually based on breadth first search (BFS), even though the worst case time complexity (total amount of work) increases to quadratic for all known parallel SCC algorithms.

Given a graph (with set of vertices V) and some pivot vertex $v \in V$, one can compute the set $\texttt{Fwd}(v)$ of all nodes reachable from v by following successor edges, using distributed BFS. Similarly, the set $\texttt{Bwd}(v)$ of all nodes reachable from v by following predecessor edges can be computed.

The simplest parallel SCC algorithm [2] is based on the following observation: the intersection of $\texttt{Fwd}(v)$ and $\texttt{Bwd}(v)$ coincides with the SCC that contains v. Moreover, all other SCCs lie completely in one of the following sets: $B \setminus F$, $F \setminus B$, or $V \setminus (B \cup F)$. So these three subgraphs can be processed independently and in parallel. This is illustrated with the figure and pseudocode on the next page.

3 The implementation challenge

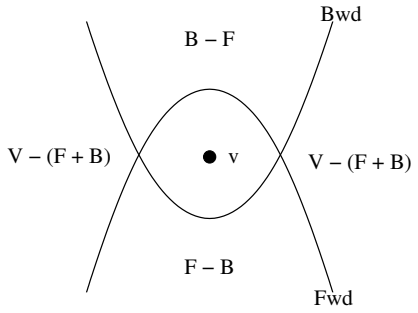
The challenge is to implement the procedure FB in a distributed and parallel way. The implementation should:

1. be efficient and scalable;
2. allow experimentation and exploration of alternatives;
3. separate implementation details from algorithmic aspects.

Dagstuhl Seminar Proceedings 08332
Distributed Verification and Grid Computing
http://drops.dagstuhl.de/opus/volltexte/2008/1629

```
Given: G = (V, E).

procedure FB(V)
  pick pivot v ∈ V;
  F := Fwd(v);
  B := Bwd(v);
  report F ∩ B;
  in parallel do
    FB(F \ B);
    FB(B \ F);
    FB(V \ (B ∪ F));
end FB
```



Remarks ad 1: The assumption is that the graph doesn't fit in the memory of a single workstation. Initially, the graph is given implicitly, by an enumerator that provides some initial vertex, and for each vertex a list of its direct successors. A first step could be to generate and store the full state space. Usually, states are partitioned over the workers by a hash function. Passing around large subgraphs (as suggested by the pseudocode) should probably be avoided.

Remarks ad 2: The implementation of FB still allows several strategies, for instance which pivot vertex is picked, and how the three recursive calls are scheduled. Note that each instance of FB contains already a distributed reachability procedure. In [1] we showed that handling the three recursive calls sequentially is outperformed by handling all recursive calls in parallel. Maybe the optimum is somewhere in between. [1] also introduced a number of improved – but more complicated – algorithms, in particular recursive OBF.

Remarks ad 3: In order to easily experiment with this and other verification algorithms, it would be nice to have a framework for distributed graph algorithms, that hides the synchronization and communication details. In this way, it becomes easier to focus on the consequences of purely algorithmic choices.

The complicating factor seems to be that the computation is data-driven. In particular, the workers share a large data structure that cannot be easily moved around. As a consequence, the computation is not purely functional.

Acknowledgements. The author wishes to thank all participants of the Dagstuhl seminar 08332 for the nice discussion.

References

1. J. Barnat, J. Chaloupka, and J.C. van de Pol. Improved distributed algorithms for SCC decomposition. In: I. Cerna and B. Haverkort (eds), Proceedings of PDMC'07. *ENTCS*, 198(1):63–77, 2008.
2. W. McLendon III, B. Hendrickson, S.J. Plimpton, and L. Rauchwerger. Finding strongly connected components in distributed graphs. *J. Parallel Distrib. Comput.*, 65(8):901–910, 2005.
3. R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on computing*, pages 146–160, 1972.