Parallel Algorithms for Model Checking

Jaco van de Pol

University of Twente, Formal Methods and Tools, Enschede, The Netherlands
J.C.vandePol@utwente.nl

Model checking [1, 5] is an automated verification procedure, which checks that a model of a system satisfies certain properties. These properties are typically expressed in some temporal logic, like LTL and CTL. Algorithms for LTL model checking (linear time logic) are based on automata theory and graph algorithms, while algorithms for CTL (computation tree logic) are based on fixed-point computations and set operations.

The basic model checking procedures examine the state space of a system exhaustively, which grows exponentially in the number of variables or parallel components. Scalability of model checking is achieved by clever abstractions (for instance counter-example guided abstraction refinement), clever algorithms (for instance partial-order reduction), clever data-structures (for instance binary decision diagrams) and, finally, clever use of hardware resources, for instance algorithms for distributed and multi-core computers.

This invited lecture will provide a number of highlights of our research in the last decade on high-performance model checking, as it is implemented in the open source LTSmin tool set[1] [10], focusing on the algorithms and datastructures in its multi-core tools.

A lock-free, scalable hash-table maintains a globally shared set of already visited state vectors. Using this, parallel workers can semi-independently explore different parts of the state space, still ensuring that every state will be explored exactly once. Our implementation proved to scale linearly on tens of processors [12].

Parallel algorithms for NDFS. Nested Depth-First Search [6] is a linear-time algorithm to detect accepting cycles in Büchi automata. LTL model checking can be reduced to the emptiness problem of Büchi automata, i.e. the absence of accepting cycles. We introduced a parallel version of this algorithm [9], despite the fact that Depth-First Search is hard to parallelize. Our multi-core implementation is compatible with important state space reduction techniques, in particular state compression and partial-order reduction [11, 15] and generalizes to timed automata [13].

A multi-core library for Decision Diagrams, called Sylvan [7]. Binary Decision Diagrams (BDD) have been introduced as concise representations of sets of Boolean vectors. The CTL model checking operations can be expressed directly on the BDD representation [4]. Sylvan provides a parallel implementation of BDD operations for shared-memory, multi-core processors. We also provided successful experiments on

[1] http://ltsmin.utwente.nl, https://github.com/utwente-fmt/ltsmin.

distributed BDDs over a cluster of multi-core computer servers [14]. Besides BDDs, Sylvan also supports Multi-way and Multi-terminal Decision Diagrams.

Multi-core algorithms to detect Strongly Connected Components. An alternative model-checking algorithm is based on the decomposition and analysis of Strongly Connected Components (SCCs). We have implemented a parallel version of Dijkstra's SCC algorithm [2, 8]. It forms the basis of model checking LTL using generalized Büchi and Rabin automata [3]. SCCs are also useful for model checking with fairness, probabilistic model checking, and implementing partial-order reduction.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
2. Bloemen, V., Laarman, A., van de Pol, J.: Multi-core on-the-fly SCC decomposition. In: PPoPP'16, pp. 8:1–8:12. ACM (2016)
3. Bloemen, V., Duret-Lutz, A., van de Pol, J.: Explicit state model checking with generalized Büchi and Rabin automata. In: SPIN'17: Model Checking of Software. ACM SIGSOFT (2017)
4. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^20 states and beyond. Inf. Comput. **98**(2), 142–170 (1992)
5. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. The MIT Press (1999)
6. Courcoubetis, C., Vardi, M.Y., Wolper, P., Yannakakis, M.: Memory-efficient algorithm for the verification of temporal properties. Formal Methods Syst. Des. **1**, 275–288 (1992)
7. van Dijk, T., van de Pol, J.: Sylvan: multi-core framework for decision diagrams. Int. J. Softw. Tools Technol. Transfer (2016)
8. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
9. Evangelista, S., Laarman, A., Petrucci, L., van de Pol, J.: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 269–283. Springer, Heidelberg (2012)
10. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: high-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 692–707. Springer, Heidelberg (2015)
11. Laarman, A., Pater, E., van de Pol, J., Hansen, H.: Guard-based partial-order reduction. STTT **18**(4), 427–448 (2016)
12. Laarman, A., van de Pol, J., Weber, M.: Boosting multi-core reachability performance with shared hash tables. In: FMCAD 2010, pp. 247–255 (2010)
13. Laarman, A.W., Olesen, M.C., Dalsgaard, A.E., Larsen, K.G., van de Pol, J.C.: Multi-core emptiness checking of timed Büchi automata using inclusion abstraction. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 968–983. Springer, Heidelberg (2013)
14. Oortwijn, W., van Dijk, T., van de Pol, J.: Distributed binary decision diagrams for symbolic reachability. In: SPIN'17: Model Checking of Software. ACM SIGSOFT (2017)
15. Valmari, A.: A stubborn attack on state explosion. Formal Methods Syst. Des. **1**(4), 297–322 (1992)