DiVINE – A Tool for Distributed Verification^{*}

(Tool Paper)

Jiří Barnat, Luboš Brim, Ivana Černá, Pavel Moravec,
Petr Ročkal, and Pavel Šimeček

Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract. We present a tool for cluster-based LTL model-checking and reachability analysis. The tool incorporates several novel distributed-memory algorithms and provides a unique interface to use them. We describe the basic structure of the tool, discuss the main architecture decisions made, and briefly explain how the tool can be used.

1 Introduction

A few enumerative verification tools have been developed to support engineers in their verification needs. Despite significant improvements in model-checking techniques, their verification capabilities are in the case of real-life industrial models limited by the amount of data a *single* state-of-the-art computer is able to handle efficiently.

In recent years, extensive research has been conducted in parallel and distributed model-checking with the aim to push forward the frontiers of enumeratively verifiable systems [1,3,4,6,8]. Consequently, several distributed verification prototype tools emerged. The deployment and usage of a distributed tool is significantly more demanding compared to the sequential one. It assumes a cluster with properly installed message passing software and also some programming skills are required in the case the tool has to be compiled from its source codes. These are some of the reasons why distributed verification tools are used rarely, although their verification capabilities are undoubtedly bigger in comparison to the sequential tools.

The goal of Distributed Verification Environment project (DiVINE) is to provide an extensible framework to support distributed verification on clusters. DiVINE offers three means to achieve this goal: First, a library of common functions (DiVINE LIBRARY) on top of which various distributed verification algorithms can be implemented. Second, a collection of state-of-the-art distributed verification algorithms incorporated into a single software product (DiVINE TOOL) which is as easy to install as most sequential tools. And third, a ready-to-use cluster for users of sequential tools in case they need to run experiments using DiVINE TOOL without having access to their own cluster. In this paper we report on DiVINE TOOL only.

^{*} This work has been partially supported by the Grant Agency of Czech Republic grant No. 201/06/1338 and the Academy of Sciences grant No. 1ET408050503.

2 DiVINE TOOL

DiVINE TOOL is a parallel, distributed-memory enumerative model-checking tool for verification of concurrent systems. The tool employs aggregate power of network-interconnected workstations to verify systems whose verification is beyond capabilities of sequential tools.

DiVINE modelling language is rich enough to describe systems made of synchronous and asynchronous processes communicating via shared memory and buffered or unbuffered channels. System properties can be specified either directly in Linear Temporal Logic (LTL) or alternatively as processes describing undesired behaviour of systems under consideration (negative claim automata). Thanks to the DivSPIN project [2], DiVINE TOOL is also capable of verifying models written in ProMeLa.

From the algorithmic point of view, the tool is quite unique. In automata-based approach to LTL model-checking, the verification problem is reduced to problem of accepting cycle detection in the graph of Büchi automaton. Two algorithms are typically used for solving the problem: Nested Depth-First Search algorithm and Tarjan's algorithm for decomposition of the graph into strongly connected components. Unfortunately, they both strongly rely on depth-first search postorder that is known to be difficult to be computed in parallel. Therefore, new, principally different, parallel algorithms for accepting cycle detection had to be designed. These are, namely, algorithm for cycle detection using additional dependency data structure, algorithm based on negative cycles, algorithms for forward and backward elimination of trivial and non-accepting strongly connected components, algorithm for cycle detection based on breadth-first search, and algorithm based on propagation of the value of maximal accepting predecessor (see [1] for an overview). Besides these, DiVINE TOOL includes also an algorithm for distributed state space generation and an algorithm that performs sequential NestedDFS in a distributed-memory setting. More details on algorithms can be found on DiVINE project web pages [5].

DiVINE TOOL can be deployed either as a complete software package to be installed on a separate Linux cluster or as a small Java application to access a pre-installed clusters. In the first case, basic Linux administrator skills are required to install the tool, but the user is in the full control of environment settings under which distributed algorithms are to be executed and can control the tool from a command line. In the second case, the tool can be used employing DiVINE pre-installed clusters and accessed remotely via a graphical user interface. The graphical user interface (GUI) requires properly installed Java Runtime Environment. Both versions are available on DiVINE project web page [5] together with a few models determined for initial acquaintance with the tool.

An important part of the DiVINE project is the maintenance of a *public* server together with a limited number of DiVINE dedicated clusters. For security reasons registered users are allowed to connect to DiVINE public server only. New users can be registered by following instructions given on DiVINE project web pages.

3 Interacting with DiVinE by Using GUI

The description of command line interface is beyond the scope of this paper. Therefore, we focus on controlling DiVinE TOOL with GUI only. GUI is implemented as a client-server application where the server part is responsible for the control of the tool. This means the server maintains currently verified models, executes distributed algorithms, monitors cluster load, etc.

The client window is divided into three parts. In the main working area, models and properties are specified, and outputs of distributed algorithms are displayed. Another part of the client window has a tree-like structure and is used to browse currently loaded models including corresponding properties and verification results. The third part displays messages reporting changes in the status of running algorithms.

A new *verification project* is started by pressing the *New model* button. The system to be verified can be written directly into the main window or imported from a local file. Having specified a model of the system the user is expected to provide properties the system should meet using the button *Add property*. Besides distributed state space generation, the tool is capable of verifying full range of LTL formulae over state-based atomic propositions. Atomic propositions are specified using the keyword `#define`, e.g. `#define p x>3`, the formula is specified using the keyword `#property`, e.g. `#property FG(p)`. Property specification can also be imported from a local file. The pair model-property is called a *task*. User can assign several distributed algorithms to be run for a given task. The number of workstations to be used can be specified for every algorithm as well. Individual algorithms are initiated with the button *Execute*.

Each algorithm produces two different types of output that can be accessed with the client: the standard output and log files. While the standard output is used to report progress in the computation and final verification results, logs are used to generate multiple statistics to support the performance analysis. For each computer participating in the computation, the logged values include the amount of memory currently allocated by the algorithm, number of sent and received messages, time spent in user and kernel space, size of queue of unexplored states, etc. Client displays the last logged values with refresh rate around five seconds, which allows the user to monitor the status of the computation in almost real time.

All specified models, properties and verification results are stored on the server until they are explicitly removed. Therefore, the user can disconnect from the server, while initiated algorithms are still running, and reconnect later to collect the verification results. It is also possible to specify and initiate new tasks during computation of others. Hence, several tasks can be computed in parallel.

4 Conclusion

DiVinE is a tool for enumerative model checking of LTL properties on a cluster of workstations. We performed numerous experiments that clearly demonstrates

the tool is capable to handle systems intractable by a single machine. E.g. for some classical verification problems the results on a cluster with 20 workstations were: Anderson’s mutual exclusion problem – space required was 10GB of memory/verification took about 40 minutes, Dining Philosophers – 9GB/20 minutes, Leader Election – 17GB/46 minutes. For more examples see the tool web page. Another interesting performance characteristic is the scalability. The figure shows typical behaviour of algorithms with respect to the number of workstations involved.

References

1. J. Barnat, L. Brim, and I. Černá. Distributed Analysis of Large Systems. In *Proc. of the 4th International Symposium on Formal Methods for Components and Objects (FMCO 05)*, LNCS. Springer, 2006.
2. J. Barnat, V. Forejt, M. Leucker, and M. Weber. DivSPIN – A SPIN compatible distributed model checker. In *Proc. 4th International Workshop on Parallel and Distributed Methods in veriFiCaTion*, pages 95–100, 2005.
3. G. Behrmann, T. S. Hune, and F. W. Vaandrager. Distributed timed model checking — how the search order matters. In *Proc. of the 12th International Conference on Computer Aided Verification*, volume 1855 of LNCS, pages 216–231. Springer, 2000.
4. B. Bollig, M. Leucker, and M. Weber. Parallel model checking for the alternation free μ -calculus. In *Proc. of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’01)*, volume 2031 of LNCS, pages 543–558. Springer, 2001.
5. DiVINE project web page: <http://anna.fi.muni.cz/divine/>.
6. H. Gavel, R. Mateescu, and I.M Smarandache. Parallel State Space Construction for Model-Checking. In *Proc. of the 8th International SPIN Workshop on Model Checking of Software (SPIN’01)*, volume 2057 of LNCS, pages 200–216. Springer, 2001.
7. F. Holmen, M. Leucker, and M. Lindstrom. Uppdmc: A distributed model checker for fragments of the μ -calculus. *Electronic Notes in Theoretical Computer Science*, 128(3):91–105, 2005.
8. F. Lerda and R. Sisto. Distributed-memory model checking with SPIN. In *Proc. of the 5th International SPIN Workshop*, volume 1680 of LNCS, pages 22–39. Springer, 1999.