DiVinE
The Distributed Verification Environment

J. Barnat, L. Brim [1], I. Černá [2], P. Šimeček

Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract

This paper presents basic concepts and current state of a general distributed verification environment (DiVinE). The environment is meant to support developers of distributed enumerative model checking algorithms, to enable unified and credible comparison of these algorithms, and to make the distributed verification available for public use.

1 Introduction

In recent years, extensive research has been conducted in parallel and distributed model checking with the aim to push forward the frontiers of still tractable systems. Many parallel and distributed algorithms have been developed and experimentally evaluated, mostly on a restricted set of verification problems. Few of them have been incorporated into existing verification tools. However, the tools are still far from the standards met by sequential tools and in most cases their availibility to public is limited.

Another important aspect related to parallel and distributed model checking algorithms is that their performance analysis as published in original papers cannot serve for their credible comparison. This is primarily due to the fact that the hardware, input models, and other circumstances differ from case to case making thus reported results incomparable. Most algorithms were implemented as research prototypes using various data structures and different optimization techniques and they simply cannot be executed on the same input data. In addition, it is impossible to assure the same conditions when redoing the experiments. Thus, in order to produce a fair comparison, nothing remains but to re-implement all algorithms on a common base and

[1] Research supported by the Academy of Sciences of Czech Republic grant No. 1ET408050503

[2] Research supported by the Grant Agency of Czech Republic grant No. 201/03/0509

re-examine their behavior on a common set of inputs and under the same circumstances.

In 2002 our group at the Faculty of informatics started the DiVinE project with the aim to develop a distributed LTL model checking verification tool and at the same time to provide a platform for development and comparison of distributed enumerative model checking algorithms. The main goals of DiVinE — *Distributed Verification Environment* can be summarized as follows:

(i) To use the environment as a platform for further development and experimental evaluation of enumerative parallel and distributed model checking algorithms.

(ii) To enable credibly evaluate existing enumerative algorithms with regard to their performance and characteristics under controlled conditions providing useful insight into their strengths and weaknesses and leading thus to a more informed way of how to choose the appropriate algorithm for given verification task.

(iii) To create a ready-to-use distributed LTL model checker as well as providing hardware to run on.

In this short presentation we aim to announce the DiVinE project to the PDMC community, report on basic concepts and ideas used in DiVinE, describe its architecture and give the current status of work.

2 DiVinE Project

The DiVinE project splits into two main parts: DiVinE ToolSet and DiVinE Library. These parts address potential DiVinE users at two different levels: at the level of a tool user (DiVinE ToolSet) and at the level of a tool developer (DiVinE Library). The overall structure of DiVinE is depicted in Figure 1.

2.1 DiVinE ToolSet

The DiVinE ToolSet is made of a set of various model checking algorithms (tools) that are accessed uniformly via a graphical user interface. An inseparable part of DiVinE ToolSet is a large collection of verification problems and case studies, i.e. a collection of models and corresponding properties to be verified. Native DiVinE specification language allows user to specify model as network of finite state machines with guarded transitions communicating via shared memory (shared variables) and buffered communication channels. This formalism has shown to be accepted by the community, which was proven by successful specification languages of model checking tools such as SPIN or UPPAAL.

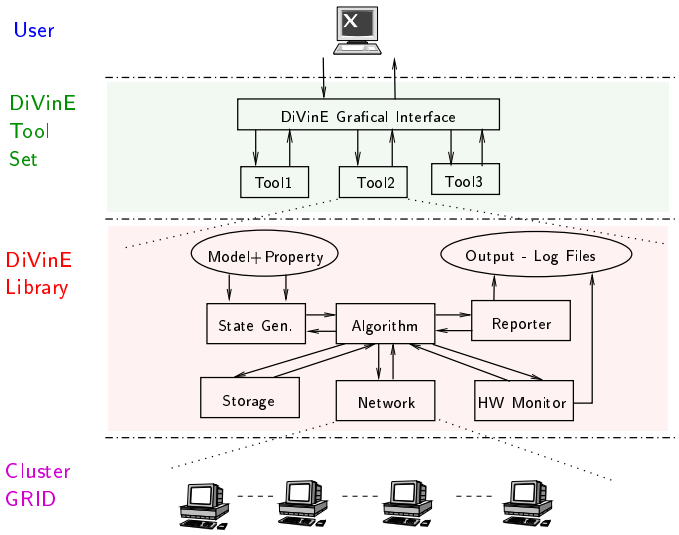

Fig. 1. DiVinE architecture

2.2 DiVinE Library

DiVinE Library is expected to be used by researchers who intend to design, implement and experimentally evaluate a distributed model checking algorithm. The library is designed to provide potential programmer with a plethora of functions that are typically needed when an implementation of such an algorithm should be done. Therefore, the programmer may focus on the main part of the algorithm and need not waste time implementing uninteresting, but necessary functions.

As can be seen from Figure 1, DiVinE Library is divided into several more or less independent modules:

State Generator is a module that provides functions needed for state space generation. These include primarily functions to compute the initial state of the system and to obtain immediate successors of a given state. In the case of LTL model checking, the module is also responsible for computing asynchronous product of system and property Büchi automata, so it also provides function to identify accepting states of the synchronous product automaton. This module also provides programmer with an interface to access the structure of the model. It implements functions for evaluating expressions, computing transitions of a single finite automaton, etc.

Storage module is responsible for storing states to the local memory. It provides functions for inserting states to a set of visited states, querying states in the set, and removing states from the set. In addition, the module is capable of storing additional constant-sized piece of information, the so called appendix, to every state stored in the set. The module is able to provide a reference to a stored state, so the algorithm can manipulate states and access their appendixes using relatively small state references.

The purpose of *HW Monitor* and *Reporter* modules is to continuously

monitor running algorithm and provide the algorithm with the feedback on hardware utilization as well as to produce logs describing the behavior of the algorithm during time of its execution. The standard POSIX signal mechanism is used to scan and log measured quantities every second.

Network module is the core module of the DiVinE Library project. This module provides basic routines for communication in the distributed setting such as send and receive primitives. In addition to the necessary basic network primitives, the module also implements mechanism for message buffering, functions for sending and receiving urgent messages, functions for partitioning state space, etc. As for high-level primitives, the module mainly provides functions for synchronization. In particular, the module supports

(i) *barrier synchronization*, which is implemented by a function that, if called on a computer, postpones computation on the computer until all other computers call the same function (this synchronization was adopted from MPI standard).

(ii) *termination detection*, which is implemented by a probe function returning `true` if all messages that were sent were also received and processed, and all participating computers are idle, and returning `false` otherwise. The termination detection can be repeatedly used for synchronization of participating computers during the computation as well as for collecting global information such as the global number of states visited, global number of sent messages, etc.

Other modules that are not depicted in Figure 1 include functions supporting partial order reduction, time profiling, counterexample generation, and property automaton decomposition.

3 Current State

The DiVinE project is being implemented in `C++` employing MPI standard for network communication. Currently, all basic functions of DiVinE Library are implemented, hence, the current version of the library can be used to produce a fully working distributed model checking algorithm. Although the library misses parts such as load balancing module, or support for sent state caching, it is ready for public use [10].

As for DiVinE ToolSet, we have already implemented and tested distributed state space generation algorithm as well as several distributed LTL model checking algorithms [2,3,1,7]. Currently, we are able to run these algortihms on models specified in native DiVinE languge, which allows us to fairly compare implemented algorithms in many details. We are also able to perform guided simulation of a model specified in DiVinE language and check model for unreachable code.

Both DiVinE Library and DiVinE ToolSet have been successfully tested with `mpich`, `LAM/MPI`, and `GridMPI/YAMPII` implementations of the

MPI standard. The DiVinE project is distributed freely under GPL.

4 Future Work

In the future, we would like to improve both design and implementation of the library, develop appropriate user unterface, improve existing model checking algorithms as well as implement new ones [8,9,7]. Our first goal now is to complete and release first stable version of the library and tool set. Note, that alpha versions are already available at [10].

Among others, our future goals include load balancing support, sent state caching mechanism, or generalization of the interface of state generator module, which should not be according to our vision so strictly binded to DiVinE native specification language. We would also like to continue with building the database of models.

In order to support verification engeneers who get used to specify models in ProMeLa, we have also started the DivSpin project in cooperation with the Research Groups at RWTH Aachen and TU München. The goals of the project are to extend DiVinE with support of ProMeLa specification language allowing thus tools in DiVinE ToolSet to verify SPIN models, and to build web-accessed distributed platform for distributed verification.

References

[1] J. Barnat, L. Brim, and J. Chaloupka. Parallel Breadth-First Search LTL Model-Checking. In *18th IEEE International Conference on Automated Software Engineering (ASE'03)*, pages 106–115. IEEE Computer Society, Oct. 2003.

[2] J. Barnat, L. Brim, and J. Stříbrná. Distributed LTL Model-Checking in SPIN. In Matthew B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN'01)*, volume 2057 of *LNCS*, pages 200–216. Springer-Verlag, 2001.

[3] J. Barnat, L. Brim, and I. Černá. Property Driven Distribution of Nested DFS. In *Proceedinfs of the 3rd International Workshop on Verification and Computational Logic (VCL'02 – held at the PLI 2002 Symposium)*, pages 1–10. University of Southampton, UK, Technical Report DSSE-TR-2002-5 in DSSE, 2002.

[4] G. Behrmann, T. S. Hune, and F. W. Vaandrager. Distributed timed model checking — how the search order matters. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 216–231. Springer-Verlag, 2000.

[5] A. Bell and B. R. Haverkort. Sequential and distributed model checking of petri net specifications. In Luboš Brim and Orna Grumberg, editors,

1st International Workshop on Parallel and Distributed Model-Checking (PDMC'02), volume 68.4 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002.

[6] B. Bollig, M. Leucker, and M. Weber. Parallel model checking for the alternation free μ-calculus. In Tiziana Margaria and Wang Yi, editors, *Proceedings 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-Verlag, 2001.

[7] L. Brim, I. Černá, P. Moravec, and J. Šimša. Accepting predecessors are better than back edges in distributed ltl model-checking. In *Formal Methods in Computer-Aided Design (FMCAD 2004), Austin, Texas, Proceedings*, volume 3312 of *Lecture Notes in Computer Science*, pages 352–366. Springer, 2004.

[8] L. Brim, I. Černá, P. Krčál, and R. Pelánek. Distributed LTL model checking based on negative cycle detection. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *Proceedings of Foundations of Software Technology and Theoretical Computer Science (FST–TCS'01)*, volume 2245 of *LNCS*, pages 96–107. Springer-Verlag, 2001.

[9] I. Černá and R. Pelánek. Distributed explicit fair cycle detection. In Thomas Ball and Sriram K. Rajamani, editors, *Model Checking Software, 10th International SPIN Workshop*, volume 2648 of *LNCS*, pages 49–73. Springer-Verlag, 2003.

[10] DiVinE – Distributed Verification Environment. http://anna.fi.muni.cz/divine.

[11] H. Garavel, R. Mateescu, and I.M Smarandache. Parallel State Space Construction for Model-Checking. In Matthew B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN'01)*, volume 2057 of *LNCS*, pages 200–216. Springer-Verlag, 2001.

[12] O. Grumberg, T. Heyman, and A. Schuster. Distributed symbolic model checking for μ-calculus. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th Conference on Computer-Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computertech Science*, pages 350–362. Springer-Verlag, July 2001.

[13] Flavio Lerda and Riccardo Sisto. Distributed-memory model checking with SPIN. In *Proceedings of the 5th International SPIN Workshop*, volume 1680 of *Lecture Notes in Computer Science*, pages 22–39. Springer, 1999.

[14] U.Stern and D. L. Dill. Parallelizing the murφ verifier. In O. Grumberg, editor, *Proceedings of Computer Aided Verification (CAV '97)*, volume 1254 of *LNCS*, pages 256–267, Berlin, Germany, 1997. Springer.