DivSPIN
A SPIN compatible distributed model checker

– Work in progress –

M. Leucker [a] M. Weber [b] V. Forejt [c] J. Barnat [c]

[a] *Institut für Informatik, Technical University of Munich, Germany*
Martin.Leucker@in.tum.de

[b] *Lehrstuhl für Informatik II, RWTH Aachen University, Germany*
michaelw@i2.informatik.rwth-aachen.de

[c] *Faculty of Informatics, Masaryk University in Brno, Czech Republic*
{xforejt,barnat}@fi.muni.cz

Abstract

This paper describes the design and implementation ideas of an extension of the parallel and distributed model checker DiVinE to a SPIN compatible distributed model checker DivSPIN. The goal of DivSPIN is to serve as user-friendly, ready-to-use system that takes up the recent theoretical and practical developments in the area of distributed model checkers and combines them with well settled operational procedures of sequential model checkers to show the benefits of parallel model checking for typical verification tasks. For this project, the research teams located at Masaryk University Brno, Czech Republic, RWTH Aachen University, and TU Munich, Germany join their efforts.

1 Introduction

SPIN is a sequential model checking tool used by thousands of people worldwide. Promela, the modeling language of SPIN, combines syntactic constructs from several popular languages, and became de facto standard specification language extensively used in sequential enumerative verification. However, when verification engineers find themselves in the situation of needing resources beyond the capabilities of a single computer, Promela models cannot be verified.

In recent years, research has been conducted in model checking algorithms which utilize the combined resources of parallel or distributed computers to further push the borders of still tractable systems. Nevertheless, most of the so-far developed algorithms have been implemented as research prototypes

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

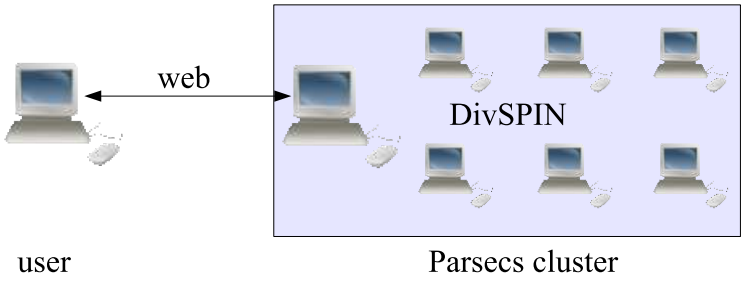

Fig. 1. Using DivSPIN

which are often not publicly available, usually undocumented, without user interface, unstable (in the sense of "prone to change"), and not optimized. These tools are mainly research vehicles, and as such not ready for widespread use by third parties.

Additionally, deployment of tools running on parallel computers is more demanding than for sequential tools. We cite high entrance costs for hardware acquisition, complex software installation procedures, but also ensuing maintenance costs. As a consequence, hardly any benchmark results of parallel and/or distributed model checking algorithms can be compared fairly, since the hardware employed for benchmarks varies from a few workstations also being used for regular tasks, to medium-sized dedicated clusters.

Recently, the DiVinE project, a distributed counterpart of standard sequential tools, has been started trying to bring advantages of distributed verification to the public. Unfortunately, DiVinE neither provides a user-friendly interface nor a widely used specification language, which are real obstacles for users that are accustomed to e. g., Promela and SPIN's graphical interface.

The goal of the DivSPIN project is to create a ready-to-use, large-scale distributed model checking tool directed at a significant part of the user base of verification tools, as well as providing hardware to run on. In particular, the goal of the project is to allow typical users of sequential model checking tools to easily access DiVinE through a user-friendly interface without the necessity of transforming Promela model into DiVinE native language.

2 Overview of DivSPIN

DivSPIN is distributed model checker running on a dedicated cluster, currently the Parsecs cluster located at RWTH Aachen University. The standard setup is shown in Figure 1. DivSPIN accepts specifications in Promela, SPIN's input language, and is able to check never claims and LTL properties. When checking LTL properties, the user can choose among several (distributed) model checking algorithms. DivSPIN is started and controlled via a web interface, to ensure that the user has a minimum of software to install and maintain.

In a typical scenario, user proceeds as follows. First, a user specifies a

model using his or her favorite editor. Furthermore, he or she formulates properties to be checked. For verification, the user selects DivSPIN's web page. Both model and property can be either uploaded from a file or typed directly into a text field. Then the user defines a verification job, henceforward called *task*. Therefore the user chooses to constrain the hardware used for the task or leave the decision to the system. Additionally, the she selects among several model checking algorithms to be used. More specifically, the user can choose several algorithms and select to stop if either the first or all have terminated. This allows comparison of different algorithms. Then the task is either executed immediately or it is submitted to a queue of waiting tasks depending on user specification and available hardware resources. During the verification task, DivSPIN reacts with status messages. Besides the web interface, a stand-alone interface, and an Eclipse plug-in are conceivable.

For the technical side, DivSPIN relies heavily on DiVinE. The DiVinE framework, described in the next section, is the back-end providing model checking algorithms. New components to be developed for DivSPIN are the Promela front-end and the web interface, both described in more detail in the next sections.

3 The Core of DivSPIN – DiVinE

The DiVinE project was started in order to support developers of *distributed* enumerative model checking algorithms, enable detailed, unified, and credible comparison of these algorithms, and make distributed verification available to the public. The project consists of two parts, the DiVinE library and the DiVinE toolset. The purpose of the library is to provide potential programmers with those necessary parts of the implementation that are common to all distributed algorithms and thus make the development of a distributed algorithm easier. The set of the algorithms that are implemented on the base of the library forms the toolset. Algorithms in the toolset have common interface, accept the same inputs, and are easily compared. This makes the DiVinE project an optimal platform for experimental evaluation of new research ideas related to parallel and distributed model checking [BBČŠ05,Div].

4 Promela Frontend

As already stated in the introduction, our choice of Promela as modeling language was driven by the wish to stay compatible to SPIN and enable SPIN's user base to change tools without effort if there is need. To interface with DiVinE, we need to obtain the meaning of Promela programs in form of state space models. Unfortunately, no complete formal semantics for Promela is available. Prior work [HN96,Wei97,Bev97] on this topic turned out to be incomplete, partly outdated, and even wrong in some places. In order to put our work on a formal basis, we decided to rigorously formalize

Promela, and then derive an interpreter from this specification. Generally, our main source of information were the informal description in [Hol03, ch. 7], with resorting to experimentation with SPIN to clear up ambiguities. Our requirements for Promela semantics are guided by pragmatism. We would like to have a simple and complete formal model, which incidentally should be effortless to implement, even with good performance in practice. Also, we need to take into account requirements imposed by our main interest, distributed model checking algorithms. For example, it must be possible to take snapshots of an interpreter's state, and restart the computation on another computer solely from this snapshot.

We have chosen a two-tier approach employing a compiler from Promela to a simple byte-code language, and a tiny virtual machine (VM) which interprets the compiler's output. The current implementation consists of about 3,100 lines of C code, and another 1,100 lines for testing and measuring purposes (interactive simulation, depth-/breadth-first search, hash tables).

Only the VM will be embedded into DivSPIN, thus implementational complexity is splitted into two parts. As additional benefit, implementation of VM and compiler could be carried out in parallel, based on our formal specification, allowing rapid development. It turned out that the compiler is straight-forward to implement as well, using standard text-book techniques.

As major selling points of the VM-based approach we see the reduced complexity in semantic specification *and* implementation, and good performance results. Measurements showed that our VM generates state spaces fast enough that it will not be a dominating factor in DivSPIN, and it is even in the same range as SPIN's state space generation. Note, that we deliberately kept actual model checking separate from the VM, as this is part of DiVinE's duties.

Generated states are almost equal in size to states generated by SPIN, with differences of a few bytes per states due to additional book keeping of our VM. A state contains all information necessary to restart the VM for further state generation, as requested above. States are represented as opaque byte sequences, thus eliminating any overhead usually imposed by converting them into a format suitable for network transmission.

5 The Web Interface

DivSPIN's web user interface offers an easy way to initiate and control the verification task. The user interface is a web page viewable with common browsers, which remote controls verification tools running on a dedicated cluster through a server process running on its master node (cf. fig. 1). It is attractive for potential users because they avoid installing (and keeping up-to-date) any verification tools themselves, which proved to be a non-trivial task in the past due to the often prototypical nature of these tools. Besides keeping software dependencies down, users can always work with the latest and most featureful tool version without additional effort, because it is the

cluster operators' responsibility to update their software. Instead, only a web browser is needed on the client side.

The client part of the interface hides all technical details related to initiation of a distributed computation. Users can submit multiple verification jobs to the cluster, track and manipulate their status until completion, and finally retrieve results. Since it is the server part of the interface that is responsible for submitting and monitoring jobs to its associated cluster, the user can safely disconnect while his jobs are being processed. The server is capable of informing the user about the status of running jobs, for example by e-mail. If the user remains connected while verification jobs are processed, she can watch various computation statistics such as the current memory load, the number of states discovered so far, lengths of waiting queues, etc. The interface is also expected to maintain a history of the user's sessions including examined models, verified properties, and finished tasks. The user also has the possibility to set his or her personal preferences which include preferred hardware to be used, preferred algorithms to be used, etc.

The web interface will be implemented with standard web application software, using a servlet container like Apache Tomcat [TAJP]. User requests for verification tasks or their manipulation arrive at the server solely in form of HTTP requests, thus side-stepping potential problems with networks restricted by firewalls. The servlet container handles these requests and triggers corresponding actions to control the cluster. Feedback to users is provided in form of web pages. Detailed information of verification runs, like state spaces or error trails, can be downloaded as well for later use.

In order to provide a more GUI-like user experience, standard JavaScript resp. ECMAScript [ECMA99] is used to update web pages which display statistics about running and finished jobs, etc.

6 Current State

A first version of the DiVinE library [Div] is available. We have a working and stable C implementation of the virtual machine part for our Promela frontend, and first steps of interfacing it with DiVinE have already been accomplished successfully. The accompanying compiler is implemented in Java, and complete enough to compile most examples coming with the official SPIN distribution, but before production use it needs a thorough code review to assure correct implementation of our specification. Eventually, we plan to make both VM and compiler available publically.

The biggest part of remaining work needs to go into the web interface for cluster control, which is currently under development. On the hardware side, the *Parsecs* cluster at RWTH Aachen University is available for use.

References

[BBČŠ05] J. Barnat, L. Brim, I. Černá, and P. Šimeček. DiVinE – Distributed Verification Environment. Submitted to PDMC'05's short presentations., 2005.

[Bev97] W. Bevier. Towards an operational semantics of PROMELA in ACL2. In *Proceedings of the 3rd International SPIN Workshop*, April 1997.

[Div] DiVinE. http://anna.fi.muni.cz/divine.

[ECMA99] European Computer Manufacturers Association. ECMA-262: ECMAScript language specification. Available at `http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM`, December 1999.

[HN96] Gerard J. Holzmann and V. Natarajan. Outline for an operational-semantics definition of PROMELA. Technical report, Bell Laboratories, July 1996.

[Hol03] Gerald J. Holzmann. *The SPIN model checker: primer and reference manual*. Addison-Wesley, Boston, MA 02116, September 2003.

[TAJP] The Apache Jarkata Project. Apache Jakarta Tomcat. `http://jakarta.apache.org/tomcat/`.

[Wei97] Carsten Weise. An incremental formal semantics for PROMELA. In *Proceedings of the 3rd International SPIN Workshop*, 1997.