Runtime Model Checking of Multithreaded C Programs using Automated Instrumentation Dynamic Partial Order Reduction and Distributed Checking [⋆]

Ganesh Gopalakrishnan Yu Yang

School of Computing, University of Utah
Salt Lake City, UT 84112, U.S.A.
{ganesh,yuyang}@cs.utah.edu

1 Introduction

Conventional testing methods are inadequate for debugging multithreaded C programs because many "unexpected" thread interactions can only be manifested through very low probability event sequences that are easily missed during test creation. As a result, bugs can escape into the field, and often manifest years after code deployment [1]. While the approach of building formal models in modeling languages such as Promela [2] followed by model checking using tools such as SPIN [2] has been shown to be effective for verification, the burden of model extraction makes this approach inapplicable in general software development. Runtime model checking – first implemented in the Verisoft [3] tool – can help solve this problem by running the actual code under the control of a scheduling mechanism, and detecting violations of assertions (typically deadlocks and other safety properties).

It is well recognized that when interleaving thread actions, one must prevent the interleaving of independent actions to avoid state explosion. Static partial order reduction methods achieve this objective by a priori (through static criteria) determining persistent sets (which are a subset of actions enabled at any given state) and only interleaving actions within persistent sets. This method is ineffective in the face of information that is difficult to compute statically. For instance, given the actions of two threads `a[e1]` and `a[e2]` that are simultaneously enabled, unless one can determine whether `e1==e2`, it is not possible to tell whether the actions commute. The dynamic partial order reduction (DPOR) approach of Flanagan and Godefroid [4] avoids this difficulty by taking advantage of the known values of expressions and pointer references (in the above example checking whether `e1==e2` at runtime) to dramatically cut down the number of interleavings examined. It has been shown to be very effective in detecting safety violations and deadlocks by systematically exploring all *relevant* interleavings of the multithreaded programs under specific inputs (meaning, specific external input drivers).

[⋆] Supported in part by NSF award CNS00509379, Microsoft HPC Institutes Program, and SRC Contract 2005-TJ-1318.

2 Tutorial Focus

The original publication on DPOR [4] does not give an indication of the details of their implementation. Reading this paper may give the reader a good understanding of the potential of this technique – but not how to engineer this technique to work well in practice. For instance, to make DPOR work for a multithreaded C program, one must address these questions:

- How does one handle the plethora of synchronization primitives used in a library such as POSIX threads [5]?
- How does one detect globally shared variables? Such variables will have to be located through static analysis methods that involve techniques such as escape and alias analysis [6,7].
- How does one build an automated instrumentation facility that lets the threads communicate their global actions to a central scheduler?
- How does one employ optimizations such as lock-sets and sleep-sets to enhance the efficacy of DPOR?
- How does one scale the performance of DPOR through the use of parallelism?

This tutorial is aimed to teach the audience in a step by step fashion how to answer all the above questions, and build an actual runtime model checker that can systematically test multithreaded programs written in C. We will dissect our tool `Inspect` [8,9,10], which is our implementation of DPOR that answers all the above questions in a practical way. In particular,

- `Inspect` has detected bugs in multithreaded C programs of a few thousand lines obtained directly from public distributions, without *any* manual effort.
- The distributed version of `Inspect` implemented using the MPI library has scaled linearly up to 64 processing nodes, reducing, in one case, a 11-hour model checking run to just 11 minutes.

By the end of this tutorial, we are confident that the attendees will be able to embark on creating DPOR based model checkers for any thread based programming system. We will furnish them with the full source code of Inspect, and offer reasonable follow-up help should they have difficulties following the design.

3 Tutorial Outline

Our tutorial will be organized along the following lines:

Introduction: Survey existing testing techniques for multithreaded programs.

Tool demonstration: Demonstrate the `Inspect` runtime model checker on simple examples. Provide an overview of its inner workings by examining its stack trace.

Theory background: The attendees will learn the theoretical foundations of `Inspect` along the following lines:

- Introduction to runtime model checking

  - A survey of existing runtime model checkers, including Verisoft [3] and CHESS [11].
  - A brief introduction to partial order reduction [12].
  - Overview of dynamic partial order reduction [4].
  - Inter-procedural flow-sensitive alias analysis [7].
  - Escape analysis [6].

Design and implementation of `Inspect`: Detail the following aspects of `Inspect`:
  - An overview of the design of `Inspect`
  - How `Inspect` controls scheduling by source-to-source transformations.
  - Implementation details of `Inspect`.

Distributed runtime model checking: Detail the following aspects of a distributed version of `Inspect`:
  - Distributed dynamic partial order reduction that shows how to speed up runtime model checking by distributing work among multiple nodes in a compute cluster.
  - Implementing distributed runtime model checking using MPI [13,14].

4 Prerequisites and Duration

Attendees are expected to have some experience with multithreaded programming in C/C++. Prior experience with model checkers is desirable, but not required. We will organize our lecture slides so that even if the attendees are unable to understand the whole theory behind `Inspect`, they will still able to build a verification tool along the lines of `Inspect`.
We are planning a half-day tutorial lasting 3 hours, organized as follows:

- introduction: 20 minutes
- tool demonstration: 10 minutes
- theory background: 60 minutes
- break: 30 minutes
- design and implementation of `Inspect`: 70 minutes
- distributed runtime model checking: 20 minutes

5 Biography of the Tutors

Professor Ganesh Gopalakrishnan has been on faculty at the School of Computing, University of Utah, for 21 years. His interests span a whole array of topics that are fully described in his research webpage
`http://www.cs.utah.edu/formal_verification`. The highlight of the formal methods research program in his group has been to develop domain specific verification tools. Recently, he offered a tutorial at the FMCAD 2004 conference on shared memory consistency models along with two industrial researchers. The slides of this tutorial are online at
`http://www.cs.utah.edu/formal_verification/presentations/`

`fmcad04_tutorial2/gopalakrishnan`. Prof. Gopalakrishnan has over 120 publications and is active in various program committees, including the program committee of Computer Aided Verification (CAV) during 2006, 2007, and 2008. Prof. Gopalakrishnan is helping organize a pre-CAV 2008 workshop called EC2 ("Enabling concurrency: efficiency and correctness") with webpage
`http://www.cs.utah.edu/~ganesh/ec2/ec2-prelim-plans.pdf`.
In the area of partial order reduction, his group has developed the following tools in the past:

– The two-phase partial order reduction algorithm [15], and the PV tool that embodies this algorithm.
– A symbolic partial order reduction algorithm for Murphi [16], and the POeM tool that prototyped this algorithm.
– A DPOR algorithm for MPI parallel programs [17], and the ISP tool that is in an active state of development.

Yu Yang is a PhD student working under the supervision of Prof. Ganesh Gopalakrishnan. Mr. Yang spent the summer of 2006 working for NEC Research, and spent the summer of 2007 in Cadence Berkeley Laboratories. He has six refereed publications in formal verification. He has also helped implement two other verification tools at Utah, namely a multiprocessor execution checker (MPEC) for the Intel Itanium architecture, and a distributed model checker for the Murphi language.

References

1. Edward A. Lee. The problem with threads. volume 39, pages 33–42, Los Alamitos, CA, USA, 2006. IEEE Computer Society Press.
2. Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual.* Addison-Wesley, 2004.
3. Patrice Godefroid. Model checking for programming languages using verisoft. In *POPL*, pages 174–186, 1997.
4. Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In Jens Palsberg and Martín Abadi, editors, *POPL*, pages 110–121. ACM, 2005.
5. David R. Butenhof. *Programming with POSIX Threads.* Addison-Wesley, 1998.
6. Alexandru Salcianu and Martin Rinard. Pointer and escape analysis for multithreaded programs. In *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 12–23, New York, NY, USA, 2001. ACM Press.
7. K. D. Cooper and K. Kennedy. Fast interprocedual alias analysis. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 49–59, New York, NY, USA, 1989. ACM.
8. http://www.cs.utah.edu/∼yuyang/inspect.
9. http://www.cs.utah.edu/∼yuyang/inspect/fm08-submission.pdf.
10. Yu Yang, Xiaofang Chen, Ganesh Gopalakrishnan, and Robert M. Kirby. Distributed dynamic partial order reduction based verification of threaded software. In *SPIN*, pages 58–75. Springer, 2007.

11. http://research.microsoft.com/projects/CHESS/.
12. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.
13. Marc Snir and Steve Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA, 1998.
14. http://www.mpi forum.org/docs/docs.html.
15. Ratan Nalumasu and Ganesh Gopalakrishnan. An efficient partial order reduction algorithm with an alternative proviso implementation. *Formal Methods in System Design*, 20(3):231–247, 2002.
16. Ritwik Bhattacharya, Steven M. German, and Ganesh Gopalakrishnan. Exploiting symmetry and transactions for partial order reduction of rule based specifications. In Antti Valmari, editor, *SPIN*, volume 3925 of *Lecture Notes in Computer Science*, pages 252–270. Springer, 2006.
17. Salman Pervez, Ganesh Gopalakrishnan, Robert M. Kirby, Robert Palmer, Rajeev Thakur, and William Gropp. Practical model-checking method for verifying correctness of mpi programs. In Franck Cappello, Thomas Hérault, and Jack Dongarra, editors, *PVM/MPI*, volume 4757 of *Lecture Notes in Computer Science*, pages 344–353. Springer, 2007.