System-level state equality detection for the formal dynamic verification of legacy distributed applications

Marion Guthmüller\textsuperscript{a}, Gabriel Corona\textsuperscript{a}, Martin Quinson\textsuperscript{a,b,*}

\textsuperscript{a} LORIA laboratory (Université de Lorraine, CNRS, Inria Verdis), Nancy, France
\textsuperscript{b} Univ Rennes, Inria, CNRS, IRISA, F-35000 Rennes, France

A R T I C L E   I N F O

Article history:
Received 31 July 2015
Received in revised form 14 November 2017
Accepted 20 December 2017

Keywords:
Dynamic verification
MPI applications

A B S T R A C T

The ever increasing complexity of distributed systems mandates to formally verify their design and implementation. Unfortunately, the common approaches and existing tools to formally establish the correctness of these systems remain hardly applicable to most legacy HPC applications, that are commonly written in Fortran or C/C++, using the MPI standard. This work addresses the problem of automatically detecting at system-level the equality of the application’s state. This allows to automatically verify safety and liveness properties on legacy HPC applications. We present how this state equality detection can be achieved without any source code static analysis, but at runtime using memory introspection and classical debugging techniques.

We demonstrate the effectiveness of our approach through the exhaustive verification of several programs from the MPICH3 test suite and through the partial termination analysis of some applications from the Competition on Software Verification (SV-COMP).

© 2017 Published by Elsevier Inc.

1. Introduction

Model checking is an appealing automated technique to establish the correctness of distributed systems. But applying this technique to legacy applications requires a complete model of the application. Manually building such models is error-prone and labor-intensive. Keeping the resulting model up to date when the real application is modified constitutes another challenge. A guided approach such as the CEGAR abstraction/refinement methodology [1] can ease this modeling step, but the user still needs a high level of expertise in formal methods. Source code static analysis [2] can automatically reconstruct a model, thus removing the burden induced by the modeling step. This interesting approach is used in many existing tools [3–5]. Our work is in line with another approach called formal dynamic verification (or execution-based model checking), where the model is not explicitly known but only implicitly explored through the actual execution of the real application. The verification is thus performed on the concrete implementation of the application. In some sense, it is orthogonal with the static analysis, as static analysis and dynamic verification gather complementary but not redundant information: these approaches could be combined in the same tool.

Ultimately, our goal is to dynamically verify unmodified legacy distributed applications composed of sequential processes interacting through message passing. Certifying the whole MPI applicative stack is beyond our scope because it would require to also certify the runtime and environment while we only consider the application to be executed in this environment. This may not be sufficient for the most critical applications, but this already allows to find issues in real code.

\* Corresponding author at: Univ Rennes, Inria, CNRS, IRISA, F-35000 Rennes, France.
E-mail address: Martin.Quinson@ens-rennes.fr (M. Quinson).

https://doi.org/10.1016/j.jlamp.2017.12.004
2352-2208/© 2017 Published by Elsevier Inc.
We build upon the SimGrid framework, initially intended to assess the performance of distributed applications [6], and upon the SimGridMC module to dynamically verify the correctness of these applications [7]. It exhaustively explores the execution paths that the verified application could follow from a initial situation provided by the user. The considered non-deterministic choices consist in the message reception order: when two messages A and B are in flight, the verification consists in first exploring the execution path where the message A is delivered before B. The application is then rewinded to explore the execution path where B is delivered first. Since we work with legacy applications, the state consists in the heap, the stacks and all global variables. Each transition encompasses a message reception by a process, and its sequential execution up to the next message exchange. The sequential execution blocks are supposed to be deterministic. We believe that these assumptions are realistic and sufficient for mono-threaded MPI-like computational applications, which outcome typically only depends on the input data provided in the initial state and on the message delivery order. Leveraging the simulator architecture to fold the verified application into a single local process makes it easier to inspect, checkpoint and rewind the complete system state. The state space explosion problem is mitigated through Dynamic Partial Ordering Reduction (DPOR) [7].

In the current work, we tackle the problem of comparing system-level states of arbitrary legacy applications written in Fortran or C/C++. Detecting state equalities is mandatory to detect infinite executions and cycles, and makes it possible to verify liveness properties. Specifically, this article makes the following contributions: we detail the challenges posed by the system-level state equality detection. We propose solutions to mitigate each of those difficulties, leveraging debugging techniques to retrieve semantic information on the application. We show the practical effectiveness of our proposal through several sets of experiments: we detect liveness violations in a custom MPI code; we show the absence of infinite execution paths in programs from a Software Verification Competition; we exhaustively explore an infinite-time MPI application as well as several of the MPI applications from the official MPICH3 testsuite.

The remainder of this article is organized as follows: Section 2 presents why state equality is an important problem for the formal verification of legacy applications. Section 3 details our contribution, which is evaluated in Section 4. Section 5 presents the related work while Section 6 concludes this article.

2. State equality for the formal verification of legacy applications

Prior to this work, SimGridMC was stateless: only the system’s initial state was checkpointed. When rewinding the application, the initial state was first restored and then all transitions leading to the desired state were replayed. This section presents several use cases of the state equality in the context of the formal verification of legacy distributed applications.

2.1. Efficiently verifying HPC programs and state equality reduction

The stateless approach was satisfying in the considered context of Peer-to-Peer (P2P) protocols, but it is less adapted to HPC applications, as the computations make the execution path highly time-consuming in this case. It is then interesting to checkpoint more states to directly restore the desired state instead of reconstructing it iteratively. Another advantage of the stateful exploration is to reduce the size of the explored state space by cutting the exploration when reaching a state that was already explored. In principle, this reduction technique is complementary to other ones such as DPOR.

The first limitation of the existing stateful verification tools is that they either only save parts of the system state after decomposition [8] or only save a bitstate hash of each reachable state rather than the full state [9]. This is probably due to classical memory space restrictions, but not saving the intermediate states forces to recompute them on need. Nowadays, some computers are equipped with terabytes of memory. This makes it possible (and thus appealing) to checkpoint and analyze the whole memory state of medium-size applications, or smaller instances of legacy HPC applications.

Another limitation of the existing stateful verification tools is that they target abstract models. Adapting these techniques to legacy HPC applications is technically very challenging because we usually lack the abstract model of these applications that are typically written in Fortran or C/C++.

Instead of adapting bitstate hash techniques to legacy applications, we checkpoint the full application state: the heap, the stacks and the sections containing the global variables are copied. Capturing thousands to million of states, as required by typical verifications, can rapidly exhaust the available memory. In many applications, only a small part of the memory changes between consecutive states. In some applications 99% of the memory pages do not change between consecutive states. Our snapshots are thus compacted by exploiting the similarity between states: identical memory pages (i.e., memory segments of size 4 KiB) are shared between snapshots.

2.2. Verifying arbitrary liveness properties on legacy code

The execution loops detected by the state equality mechanism constitute non-progressive cycles. They play a central role in the verification of liveness properties, since the counterexample to liveness properties are infinite paths. If the application state size is bounded (in particular, if the stack size is bounded), most infinite paths contain such an execution loop. Liveness properties are then verified through the search of acceptance cycles in the Cartesian product of the application with a Büchi automaton encoding the negation of the verified property. If found, such an acceptance cycle denotes an infinite execution path which constitutes a counterexample to the property.
دریافت فوری
متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات