A Verification Technique for Deterministic Parallel Programs

Extended Abstract

Marieke Huisman
University of Twente
Enschede, Netherlands
m.huisman@utwente.nl

ACM Reference Format:
Marieke Huisman. 2017. A Verification Technique for Deterministic Parallel Programs. In *Proceedings of PPDP'17, Namur, Belgium, October 9–11, 2017,* 1 pages.
https://doi.org/10.1145/3131851.3131852

Software is omnipresent, and software failures can have tremendous costs for society and economy. Therefore, we need techniques to improve the quality of software, and to prevent software failures. Program verification can help to improve this situation, as it allows to check properties on *all* possible behaviours of a program. We focus in particular on the verification of concurrent software, which is even more error-prone, because of the possible interleavings between the different threads.

Within the VerCors project, we have developed techniques to reason about concurrent software. Initially we focused on the verification of concurrent Java programs, for which we verified correctness of concurrent data structures and various synchronisation mechanisms. However, our verification techniques also are suitable for other concurrent programming models, as demonstrated by applying them to verify OpenCL GPU applications. All our verification techniques are supported in the VerCors tool set.

In this presentation, we show that our verification techniques are also suitable to prove that correctness is preserved by a parallelizing compiler. In this approach, the complexity of parallel programming is handled by writing a sequential program augmented with parallelization compiler directives that indicate which part of code might be parallelized. A parallelizing compiler consumes the annotated sequential program and automatically generates a parallel version. This approach is often called *deterministic parallel programming*, as the parallelization of a deterministic sequential program augmented with correct compiler directives is always deterministic. Deterministic parallel programming is supported by different languages and libraries such as OpenMP [3] and is often used for financial and scientific applications.

Although it is relatively easy to write parallel programs in this way, careless use of compiler directives can easily introduce data races and consequently non-deterministic program behaviour. This paper proposes a static technique to prove that parallelization as indicated by the compiler directives does not introduce such non-determinism. Moreover it also shows how our technique reduces functional verification of the parallelized program to functional verification of the sequential program. We develop our verification technique over a core deterministic parallel programming language called PPL (for Parallel Programming Language). To show practical usability of our approach, we defie an encoding of a commonly used subset of OpenMP into PP. The VerCors tool set is extended to support this process.

In essence, PPL is a language for the composition of code blocks. We identify three kinds of *basic blocks*: a *parallel block*, a *vectorized block* and a *sequential block*. Basic blocks are composed by three binary block composition operators: *sequential composition*, *parallel composition* and *fusion composition* where the fusion composition allows two parallel basic blocks to be merged into one. An operational semantics for PPL is presented.

Our verification technique requires each basic block to be specified by an *iteration contract* [1] that describes which memory locations are read and written by a thread. Moreover, the program itself should be specified by a global contract. To verify the program, we show that the block compositions are memory safe (i.e. data race free) by proving that for all pairs of *independent iterations* (i.e. iterations that might run in parallel) all accesses to shared memory are non-conflicting, meaning that they are disjoint or they are read accesses. If all block compositions are memory safe, then it is sufficient to prove that the sequential composition of all the basic blocks w.r.t. program order is memory safe and functionally correct, to conclude that the parallelized program is functionally correct.

The main contributions of our work are the following:

- A core language, PPL, and an operational semantics which captures the main forms of parallelization constructs in deterministic parallel programming.
- A verification approach for reasoning about data race freedom and functional correctness of PPL programs.
- A soundness proof that all verified PPL programs are indeed data race free and functionally correct w.r.t. their contracts.
- Tool support that addresses the complete process of encoding of OpenMP into PPL and verification of PPL programs.

More detailed information about our work can be found in [2].

REFERENCES

[1] S.C.C. Blom, S. Darabi, and M. Huisman. 2015. Verification of Loop Parallelisations. In *FASE 2015 (LNCS)*, Vol. 9033. Springer, 202–217.
[2] S. Darabi, S.C.C. Blom, and M. Huisman. 2017. A Verification Technique for Deterministic Parallel Programs. In *NFM 2017 (LNCS)*, Vol. 10227. Springer, 247–264.
[3] OpenMP Last accessed Aug. 29, 2017. *The OpenMP API Specification for Parallel Programming*. http://openmp.org/.

PPDP'17, October 9–11, 2017, Namur, Belgium

ACM ISBN 978-1-4503-5291-8/17/10.
https://doi.org/10.1145/3131851.3131852