Certified compilation of concurrent C programs

Internship proposal 2025-2026
Jean-Marie Madiot, Inria Paris

Context Formally verified compilers potentially help establishing the safety of critical systems; for example, CompCert [Ler09] has industrial use cases. However such compilers are limited in their support of parallelism and concurrency, which is becoming a problem as some of those industrial users do want to exploit more than one of the many cores of modern processors with multithreaded programs.

One of the main difficulties in overcoming this limitation is that multicore architectures do not necessarily execute parallel programs as if all instructions were executed sequentially. This comes in particular from the fact that microprocessors have several levels of cache and reorder instructions. The study of which behaviours actually happen is the research subject called *weak* or *relaxed* memory models [AG96]. It does not only study micro architectures but also programming languages. This allow compilers to optimize programs in ways that respect standardized memory models such as, for C, C11 or C20 [Lah+17; BGV18].

Many compiler optimisations involve reorderings (and/or eliminations or introductions) of traditional (non-synchronizing, or *non-atomic*) memory accesses, load and stores. A reordering is only performed when the accesses appear to be independent. These optimisations assume the absence of race conditions, e.g. they assume that no two non-atomic stores at the same address can happen at the same time.

When programming, race conditions can be avoided by inserting *atomic* operations, or synchronisations. For example, if one thread does one non-atomic store at some location `data`, then an atomic *release* store at some location `flag`, and another thread does an atomic *acquire* load at `flag` then a non-atomic load at `data`, then if the `flag` load reads from the `flag` store, then also the `data` load will read from that `data` store, and not one that happened before.

The goal of this internship is to study the feasibility of a certified optimizing compiler for concurrent C programs that takes into account weak memory models, in particular the release-acquire fragment.

Approach Many weak memory models are *axiomatic* models: models that consider a large set of potential program (pre)executions and rule out the invalid executions according to a set of rules that applies the the global set of all memory events.

The standard approach to compiler verification, however, is based on operational semantics for both source and target languages, between which one establishes multiple *simulations* relations, which operate on operational semantics.

Some memory models for C are operational, either keeping a notion of a global state of the memory [TVD14], or by providing a semantics with thread-local views of the memory, in which each atomic operations updates the local view [Kai+17]. The motivation for the latter was to fit the separation logic Iris [Jun+18] so as to use its proof framework, but such *view* operational semantics should also be more readily amenable to the verification of optimizing compilers than axiomatic memory models.

The first step of this internship will be to consider a small language and a few compiler passes that are representative of the memory optimizations performed by CompCert, to equip this language with standard operational semantics and view operational semantics. In this setting, we will study and adapt refinement simulations so as to prove that some standard optimisations are valid, possibly under some hypotheses.

The longer-term goal is to adapt the CompCert compiler to handle standard C programs that make use of release-acquire synchronisations, a paradigm that is more fine-grained than e.g. mutual-exclusion locks.

Related work CompCertTSO [Sev+13] uses the TSO memory model on the source language C, instead of the standard C memory model. Concurrent Compcert [Ber+14; Cue20], CompCertX [Gu+15; Gu+18], and CASCompcert [Jia+19], model some or all memory operations as external calls, which does not allow to reason properly about weak memory models and compiler optimisations such as instructions reordering. Simuliris [Gäh+22] uses Iris to verify concurrent program optimizations, however their goal is to study complex loop optimisations requiring coinductive reasoning that are *not* in CompCert, and they assume sequential consistency and not weak memory models.

References

[AG96] Sarita V. Adve and Kourosh Gharachorloo. "Shared Memory Consistency Models: A Tutorial". In: *Computer* 29.12 (1996), pp. 66–76. URL: https://doi.org/10.1109/2.546611.

[Ler09] Xavier Leroy. "A Formally Verified Compiler Back-end". In: *J. Autom. Reason.* 43.4 (2009), pp. 363–446. URL: https://doi.org/10.1007/s10817-009-9155-4.

[Sev+13] Jaroslav Sevcík et al. "CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency". In: *J. ACM* 60.3 (2013), 22:1–22:50. URL: https://doi.org/10.1145/2487241.2487248.

[Ber+14] Lennart Beringer et al. "Verified Compilation for Shared-Memory C". In: *ESOP 2014, Grenoble, France, April 5-13, 2014, Proceedings*. Vol. 8410. LNCS. Springer, 2014, pp. 107–127. URL: https://doi.org/10.1007/978-3-642-54833-8%5C_7.

[TVD14] Aaron Turon, Viktor Vafeiadis, and Derek Dreyer. "GPS: navigating weak memory with ghosts, protocols, and separation". In: *OOPSLA 2014, part of SPLASH 2014, Portland, OR, USA, October 20-24, 2014*. ACM, 2014, pp. 691–707. URL: https://doi.org/10.1145/2660193.2660243.

[Gu+15] Ronghui Gu et al. "Deep Specifications and Certified Abstraction Layers". In: *POPL 2015, Mumbai, India, January 15-17, 2015*. ACM, 2015, pp. 595–608. URL: https://doi.org/10.1145/2676726.2676975.

[Kai+17] Jan-Oliver Kaiser et al. "Strong Logic for Weak Memory: Reasoning About Release-Acquire Consistency in Iris". In: *ECOOP 2017*. Vol. 74. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 17:1–17:29. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2017.17.

[Lah+17] Ori Lahav et al. "Repairing sequential consistency in C/C++11". In: *PLDI 2017, Barcelona, Spain, June 18-23, 2017*. ACM, 2017, pp. 618–632. URL: https://doi.org/10.1145/3062341.3062352.

[BGV18] Hans-J. Boehm, Olivier Giroux, and Viktor Vafeiades. *P0668R5: Revising the C++ memory model*. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0668r5.html [Accessed: 2026-01-01]. 2018.

[Gu+18] Ronghui Gu et al. "Certified concurrent abstraction layers". In: *PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. ACM, 2018, pp. 646–661. URL: https://doi.org/10.1145/3192366.3192381.

[Jun+18] Ralf Jung et al. "Iris from the ground up: A modular foundation for higher-order concurrent separation logic". In: *J. Funct. Program.* 28 (2018), e20. URL: https://doi.org/10.1017/S0956796818000151.

[Jia+19] Hanru Jiang et al. "Towards certified separate compilation for concurrent programs". In: *PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*. ACM, 2019, pp. 111–125. URL: https://doi.org/10.1145/3314221.3314595.

[Cue20] Santiago Cuellar. "Concurrent Permission Machine for modular proofs of optimizing compilers with shared memory concurrency". PhD thesis. Princeton University, USA, 2020. URL: https://arks.princeton.edu/ark:/88435/dsp01qr46r378d.

[Gäh+22] Lennard Gäher et al. "Simuliris: a separation logic framework for verifying concurrent program optimizations". In: *Proc. ACM Program. Lang.* POPL (2022), pp. 1–31. URL: https://doi.org/10.1145/3498689.