Formally Verifying Optimizations with Block Simulations

Gourdin Léo, Bonneau Benjamin, Boulmé Sylvain, Monniaux David, Bérard Alexandre

▶ **To cite this version:**

Gourdin Léo, Bonneau Benjamin, Boulmé Sylvain, Monniaux David, Bérard Alexandre. Formally Verifying Optimizations with Block Simulations. Object-oriented Programming, Systems, Languages, and Applications (OOPSLA), Oct 2023, Cascais, Portugal. hal-04749000

HAL Id: hal-04749000
https://hal.science/hal-04749000v1

Submitted on 22 Oct 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

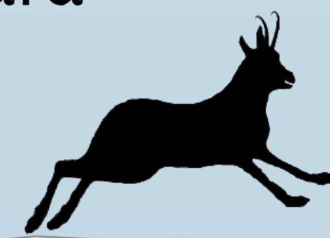
L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formally Verifying Optimizations with Block Simulations

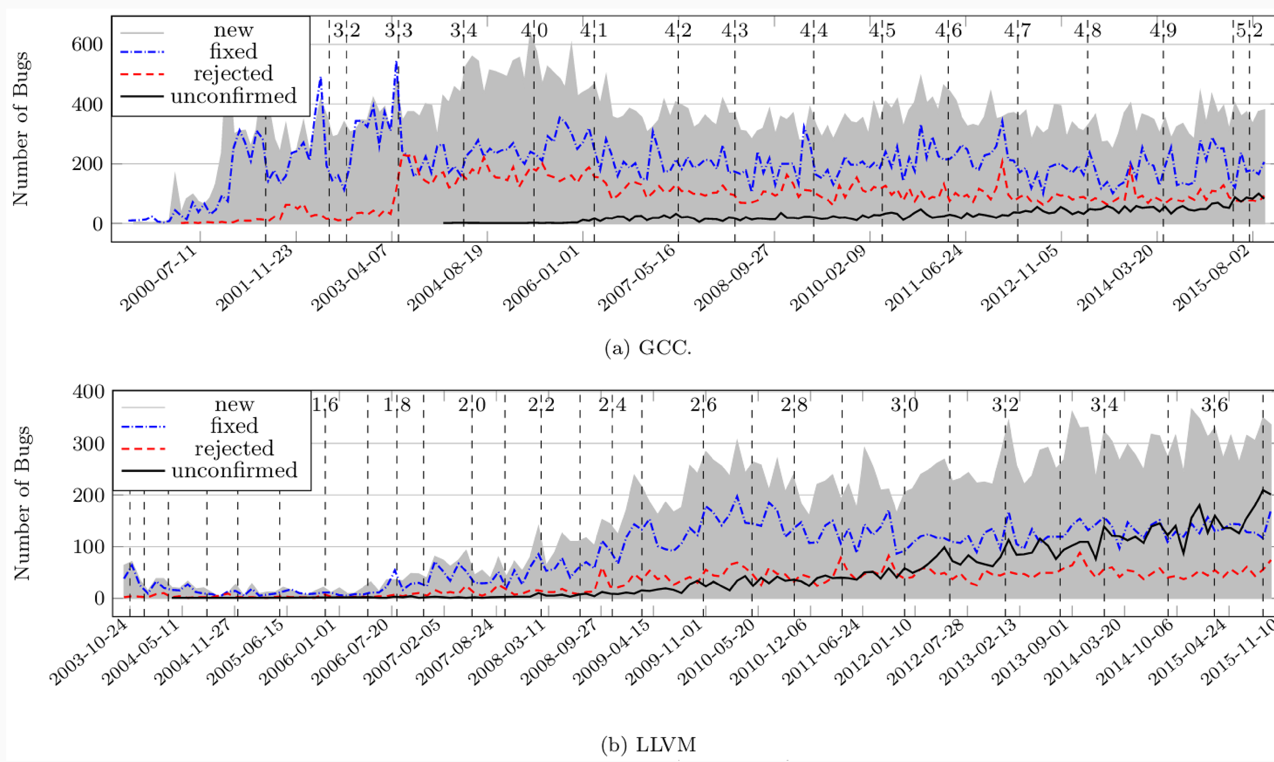
L. Gourdin, B. Bonneau, S. Boulmé, D. Monniaux & A. Bérard*

Université Grenoble Alpes, CNRS, Grenoble INP, Verimag

https://gricad-gitlab.univ-grenoble-alpes.fr/certicompil/Chamois-CompCert



Avoiding Bugs in GCC & LLVM (cf. [1])?



CompCert[1] **solution** (ACM Software System Award 2021): the 1st *formally verified* (= machine-checked mathematical proof of correctness) compiler optimizing **safety-critical** software [2, 3].

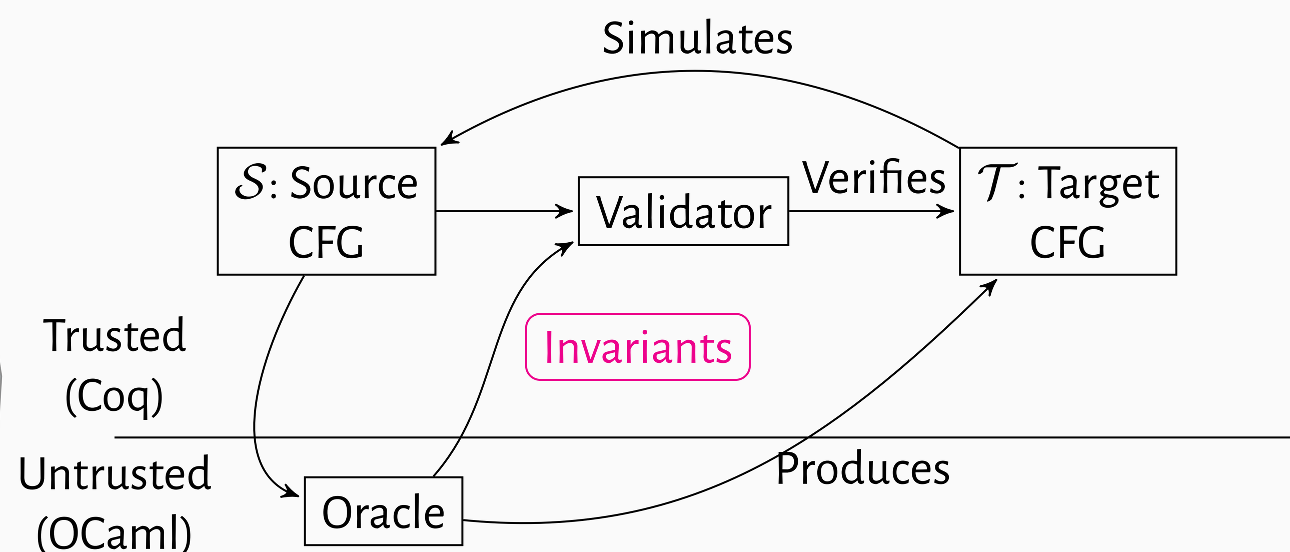
[1] https://www.absint.com/compcert/

Block Transfer Language

and its formally verified Translation Validator

The oracle takes source CFG $\mathcal{S}$ and yields its optimized version $\mathcal{T}$ along with an **invariant at the entry of each (loop-free) block**. Our verified symbolic simulation validator either ensures semantic preservation or aborts compilation.

Invariants express **relations between source and target states**.



Purposes of BTL TV

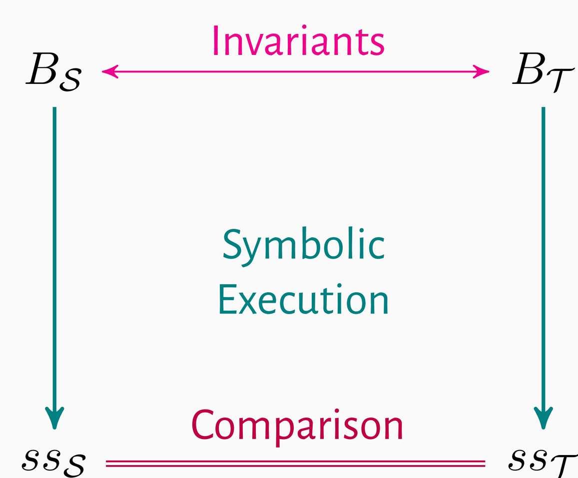
Block-by-block comparison of
a source CFG and
an oracle-provided target CFG
modulo oracle-provided invariants
and oracle-dependent rewriting rules

Features

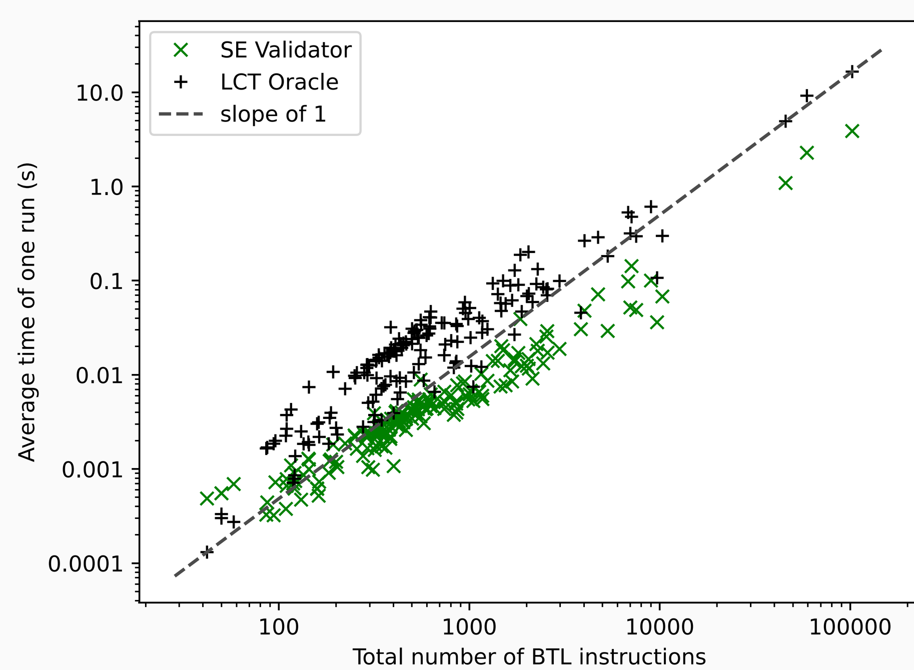
- Formally verified integration in Chamois-CompCert middle-end
- Parametrized by the architecture
- Predictible (against false alarms)
- Fast and Versatile
- **1st verified strength-reduction** of induction variables
- Validates various loop optimizations & *superblock scheduling* with code compensation and register renaming.

Block-by-block Symbolic Simulation

For each pair of blocks $(B_{\mathcal{S}}, B_{\mathcal{T}})$, it compares the symbolic *states* $(ss_{\mathcal{S}}, ss_{\mathcal{T}})$ resulting from their symbolic execution, themselves initialized from entry invariants: it tests equalities of exit invariants after normalized rewriting.



Compile Times That Scale (thanks to formally verified hash-consing)



Validating a Lazy Code Transformations Oracle

Combining and improving Lazy Code Motion [4] & Lazy Strength Reduction [5].

- Search for loop-invariant or "reducible" *candidates*;
- Supports decomposed patterns like a *left shift* + an *addition*;
- Based on data-flow analyses performed by an OCaml oracle;

```
void init7(long *t,
           unsigned long i,
           unsigned long l)
{ for (; i<l; i++) t[i]=7; }
```

C source code

Optimizing in two steps:

1. *Lifting* candidates out of the loop;
2. Inserting *compensation code* in the loop body if necessary.

Generating invariants from lifted computations and variables that remain live.

```
.L100: //NB: x0=0; x10=t; x11=i; x12=l
  bgeu x11,x12,.L101//exit if x11≥x12
  slli  x7,x11,3      // x7 = x11*2^3
  add   x13,x10,x7    // x13 = x10+x7
  addi  x5,x0,7       // x5 = 7
  sd    x5,0(x13)     // *x13 = x5
  addi  x11,x11,1     // x11 += 1
  j     .L100
```

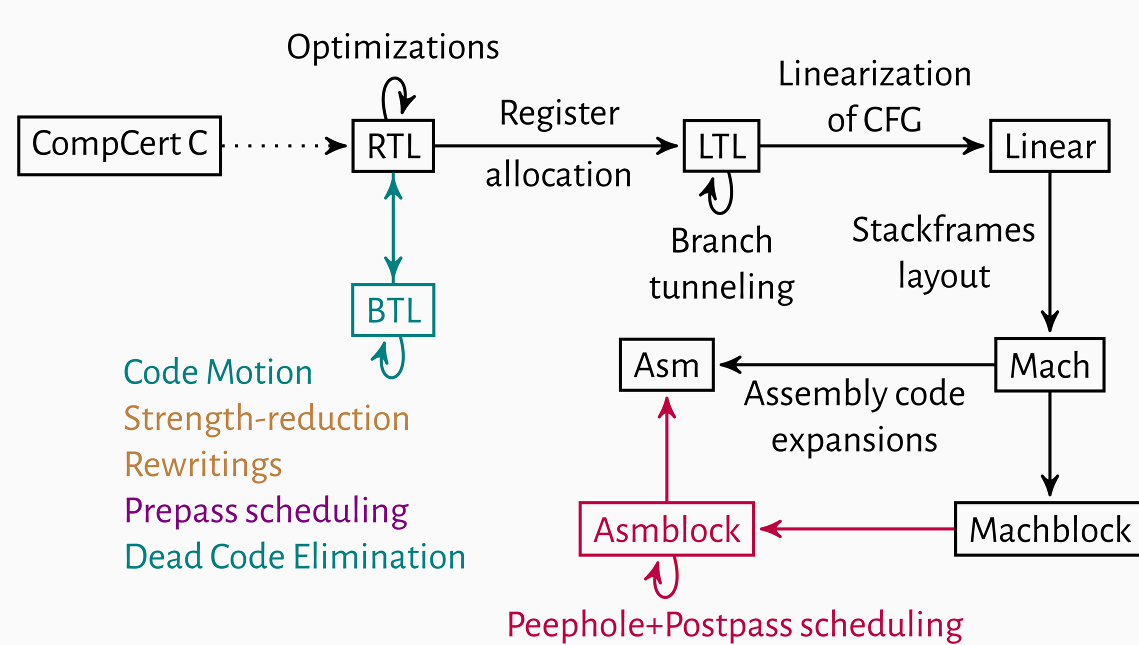
RISC-V ASM Before

→

```
  slli  x7,x11,3      // x7 = x11*2^3
  add   x13,x10,x7    // x13 = x10+x7
  addi  x5,x0,7       // x5 = 7
.L100:
  //invariant x5_T = 7 ∧ x13_T = x10_S+x11_S×8
  //           ∧ x11_T = x11_S ∧ x12_T = x12_S
  bgeu  x11,x12,.L101//exit if ...
  sd    x5,0(x13)     // *x13 = x5
  addi  x13,x13,8     // x13 += 8
  addi  x11,x11,1     // x11 += 1
  j     .L100
```

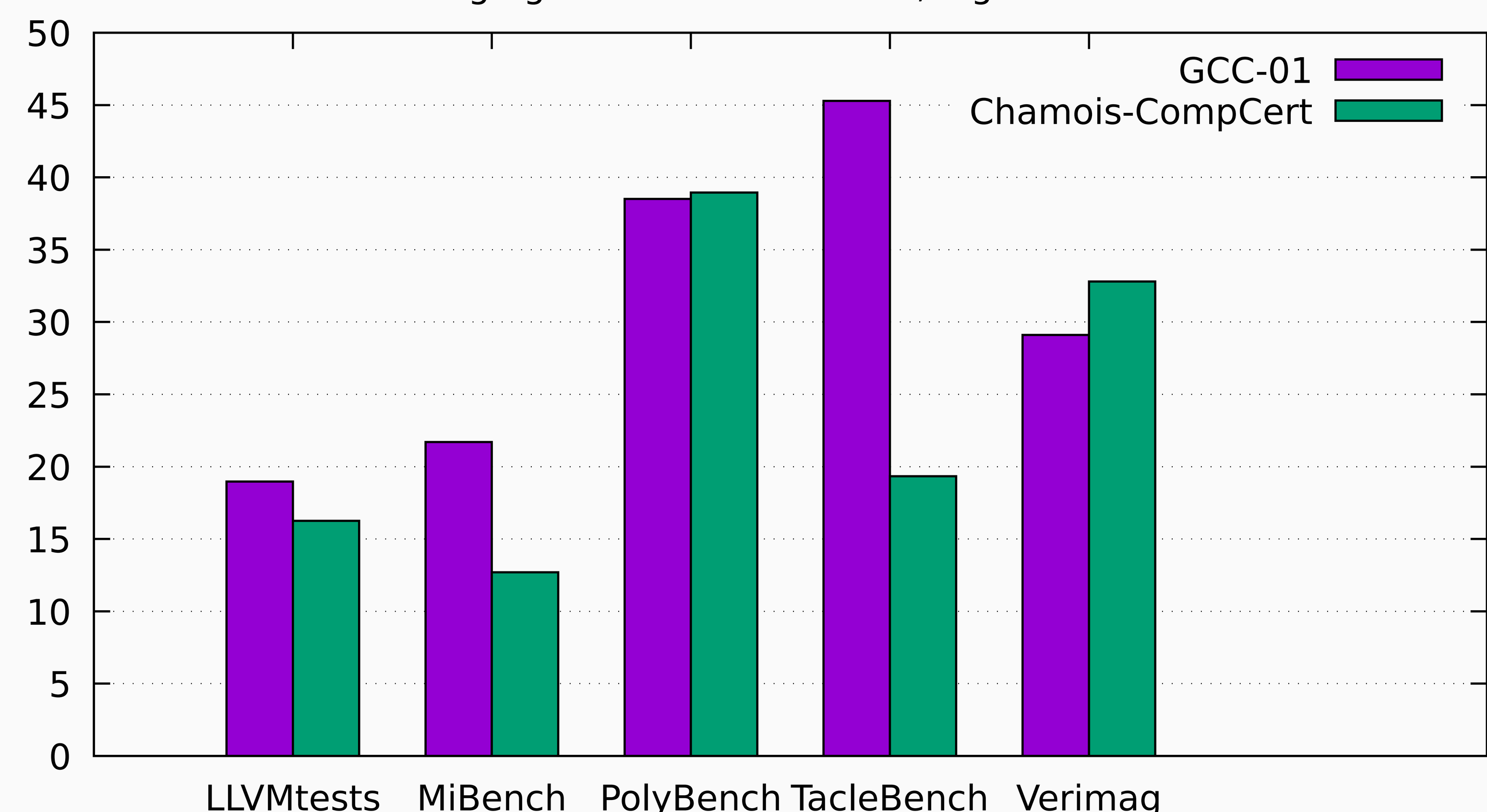
RISC-V ASM After

Chamois-CompCert Optimizations



Legend:
Brown: **RISC-V** only
Violet: AArch64 +ARMv7+**RISC-V**+KVX
Red: AArch64 +KVX
Teal: All (AArch64 +ARMv7+**RISC-V**+KVX +PPC+x86)

Optimized Generated Code That You Can Trust



References

[1] Chengnian Sun et al. "Toward understanding compiler bugs in GCC and LLVM". en. In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. Saarbrücken Germany: ACM, 2016, pp. 294–305. ISBN: 978-1-4503-4390-9. DOI: 10.1145/2931037.2931074. URL: https://dl.acm.org/doi/10.1145/2931037.2931074 (visited on 06/17/2022).

[2] Xavier Leroy. "Formal verification of a realistic compiler". In: *Communications of the ACM* 52.7 (2009). DOI: 10.1145/1538788.1538814.

[3] Daniel Kästner et al. "CompCert: Practical experience on integrating and qualifying a formally verified optimizing compiler". In: *ERTS 2018: Embedded Real Time Software and Systems*. SEE, 2018.

[4] Jens Knoop, Oliver Ruthing, and Bernhard Steffen. "Optimal Code Motion: Theory and Practice". In: *ACM Transactions on Programming Languages and Systems* 16 (1995). DOI: 10.1145/183432.183443.

[5] Jens Knoop, Oliver Rüthing, and Bernhard Steffen. "Lazy Strength Reduction". In: *Journal of Programming Languages* 1 (1993), pp. 71–91.

Git Repo

*This work is supported by the French National Research Agency in the framework of the "France 2030" program (ANR-15-IDEX-0002) and by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

LaTeX TikZposter