Extending the CompCert certified C compiler

with instruction scheduling and control-flow integrity (CFI)

October 2019

Sylvain.Boulme@univ-grenoble-alpes.fr

Verimag, Grenoble-INP

Issue : *optimizing* compiler for *safety-critical* software

Compilation bugs in most C compilers (GCC, LLVM, etc).

Attested by randomized differential testing :
Eide-Regehr'08, Yang-et-al'11, Lidbury-et-al'15, ...

Tests of ***optimizing*** *compilers* **cannot cover** all corner cases !

Strong *safety-critical* requirements of

Avionics (DO-178), Nuclear (IEC-61513), Automotive (ISO-26262), Railway (IEC-62279)
often established at the **source** level with
human review of the *compiled code*. ← intractable if *optimized*

One solution : a **formally proved** compiler !
formal proof = computer-aided review of the compiler code w.r.t its spec.
⇒ up-to-date & very sharp (formal) documentation of the compiler
that also helps "*external developers*" (like us at Verimag)

Overview of `https://github.com/AbsInt/CompCert`

Input most of ISO C99 + a few extensions
Output (32&64 bits) code for **RISC-V**, PowerPC, ARM, x86

Developed since 2005 by Leroy-et-al at Inria
Commercial support since 2015 by AbsInt (German Company)
Industrial uses in Avionics (Airbus) & Nuclear Plants (MTU)

Unequaled level of trust for industrial-scaling compilers
Correctness proved within the Coq proof assistant

Performance of generated code (for PowerPC and ARM)

2× *faster* than `gcc -O0`
10% *slower* than `gcc -O1` and 20% than `gcc -O3`.

Example In MTU systems (emergency power for Nuclear Plants)
28% *smaller* WCET than with a previous *unverified* compiler.

Understanding the formal correctness of CompCert

Formally, correctness of compiled code is ensured modulo

- correctness of C formal semantics in Coq
- correctness of assembly formal semantics in Coq
- absence of undefined behavior in the source program

Formal semantics $\simeq$ relation between "programs" and "behaviors"
i.e. a (possibly non-deterministic) interpretation of programs

for C : formalization of ISO C99 standard
for assembly : formalization/abstraction of ISA

Source program assumed to be without undefined behavior

```
int x, t[10], y;
...
if (...) {
  t[10]=1; // undefined behavior: out of bounds
  // the compiler could write in x or y,
  // or prune the branch as dead-code, ...
```

Informal view of CompCert formal correctness

Observable Value = int or float or address of global variable

Trace = a sequence of *external function* calls (or *volatile accesses*)
each of the form "$f(v_1, \ldots, v_n) \mapsto v$" where f is name

Behavior = one of the four possible cases (of an execution) :

- an infinite trace (of a diverging execution)
- a finite trace followed by an infinite "silent" loop
- a finite trace followed by an integer exit code (terminating case)
- a finite trace followed by an error (UNDEFINED-BEHAVIOR)

Semantics = maps each *program* to a set of *behaviors*.

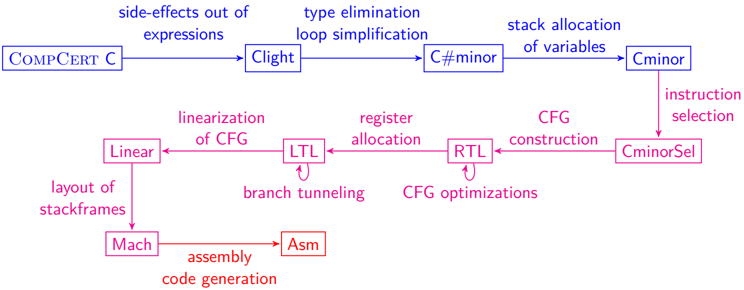
Correctness of the compiler

For any source program S,
if S has no UNDEFINED-BEHAVIOR,
and if the compiler returns some assembly program C,
then any behavior of C is also a behavior of S.

NB : under these conditions, C has no UNDEFINED-BEHAVIOR.

Modular design of CompCert in Coq

Components independent/parametrized/specific w.r.t. the target



And now, Verimag's Mach → Asm for **two** targets

1. The "K1c" VLIW core of Kalray :
   the 1st (scaling) certified compiler that optimizes ILP ?
2. A variant of RISC-V with encryption and CFI.

Instruction scheduling for CompCert/Kalray's K1c

Joint work with C. Six (Kalray/Verimag) and D. Monniaux (CNRS/Verimag)

Kalray's K1c = a 6-issue VLIW with a 7-stage pipeline,
e.g. with instruction level parallelism (ILP) in 2D

bundles of (upto) 6 instructions may run in parallel
at each of the 7 pipeline stages.

with a very *predictible* semantics : in-order & interlocked.
⇝ simplify WCET estimations & compilers design !

Two main contributions of our CompCert backend

1. an (abstract) formal semantics of VLIW assembly expressing parallel execution of instructions within *bundles*
2. a certified instruction scheduler performing assembly optimization w.r.t the 2D of ILP

a speedup of more 50% on the code generated by CompCert
coming around 10% slower than GCC-O2 (Kalray's backend)
& generally 20% faster than GCC-O1 (without scheduling)

Issue : CompCert and Control-Flow Integrity (CFI) ?

```
status pay(float amount, id client, id vendor){
        if (auth(client)) goto transaction;
        return ABORTED;
transaction:
        /* perform the transaction */
```

CompCert's formal correctness implies that

> the generated **assembly** cannot run code at `transaction` without being entered "normally" in function `pay`

under the two following conditions

- no undefined-behavior in the source (e.g. no BOV)
- trustworthy runtime environment (e.g. no hardware attack)

⇝ very restrictive conditions w.r.t practice !

Overview of CFI in CEA's IntrinSec

Works of O. Savry and its team at CEA-LETI

CEA's IntrinSec = a RISC-V variant (still under design) with
code/data encryption with CF&data access-control

Control-Flow Integrity **(in an adversarial context)**
provided by access-control on both

- the CF : ensuring that CF cannot "*enter into functions*" except at : function entry + return-address (RA) from callees
- the stack : ensuring that only "authorized instructions" can modify RA in the stack (e.g. no buffer-overflows).

Actually, the processor aborts to prevent unsecure behaviors :

Buffer-overflows can modify RA on the stack,
but then, abort on the load into RA register

CF access control for CompCert/CEA's IntrinSec

Joint work with P. Torrini (Verimag) & hints from M.L. Potet (Verimag) and O. Savry (LETI)

Our contributions

- Extend CompCert's RISC-V model with IntrinSec's instructions of CF access control
- Make CompCert generate instructions of CF access control
- Formally prove the compiler correctness (work in progress)

Future works

- Support of data access-control
- Informal CFI properties of the platform
- Toward a formalization of some CFI properties ? Issue : CompCert's models too high-level for expressing attacks ?

Conclusions

CompCert = a *moderately*-optimizing C compiler
with an *unprecedented* level of trust in its correctness

> "CompCert *is the only compiler we have tested for which* Csmith *cannot find wrong-code errors. This is not for lack of trying : we have devoted about six CPU-years to the task.*
> [...] *developing compiler optimizations within a proof framework* [...] *has tangible benefits for compiler users.*"

Yang-et-al'11 (from randomized differential testing)

CompCert ready to be included into **chip codesign**
but, in parallel of a traditional compiler !

Cons some feature could still be hard to support in CompCert
Pro *formal feedback* on the ISA (semantics & compilation process)

⇒ Convergence with RISC-V community
on safety, security, embedded systems, etc.

Appendix (offline slides)

Topics

- The Coq proof assistant
- Trust in ELF binaries produced with CompCert
- More details on the CompCert/Kalray's K1c

The Coq proof assistant

A *language* to **formalize mathematical theories** (and their proofs) **with a computer**. Examples :

- Four-color & Odd-order theorems by Gonthier-et-al.
- Univalence theory by Voevodsky (Fields Medalist).

With a high-level of confidence :

- Logic reduced to a few powerful constructs ;
  Proofs checked by a small verifiable kernel
- Consistency-by-construction of most user theories
  (promotes *definitions* instead of *axioms*)

ACM Software System Award in 2013
for Coquand, Huet, Paulin-Mohring et al.

Formally proved programs in the Coq proof assistant

The Coq logic includes a functional programming language with pattern-matching on tree-like data-structures.

Example : inserting a key x in a balanced binary tree t

```
Fixpoint add (x:key) (t:avltree): avltree :=
  match t with
  | Leaf ⇒ Node 1 Leaf x Leaf
  | Node h l y r ⇒
      match Key.compare x y with
      | Lt ⇒ bal (add x l) y r
      | Eq ⇒ Node h l y r
      | Gt ⇒ bal l y (add x r)
      end
  end
```

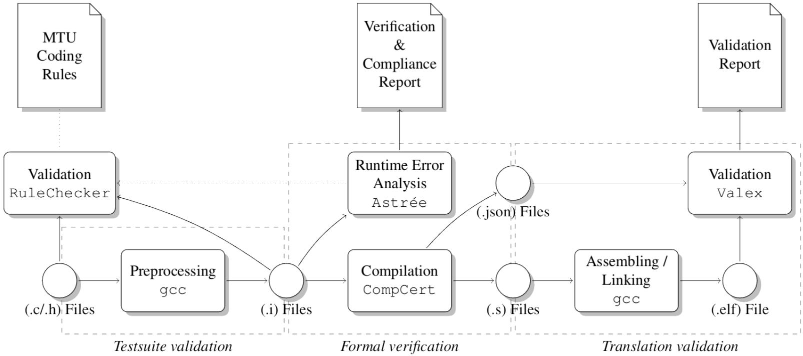
Extraction of Coq functions to OCaml
+ OCaml compilation to produce native code.

⇒ **CompCert is programmed in both Coq and OCaml.**

Trust in ELF binaries produced with CompCert

Trust in binaries requires additional verifications, at least :

- absence of undefined behavior in C code (e.g. with Astrée)
- correctness of assembling/linking (e.g. with Valex)



Qualification of MTU *development chain* for Nuclear safety
from Käster, Barrho et al @ERTS'18

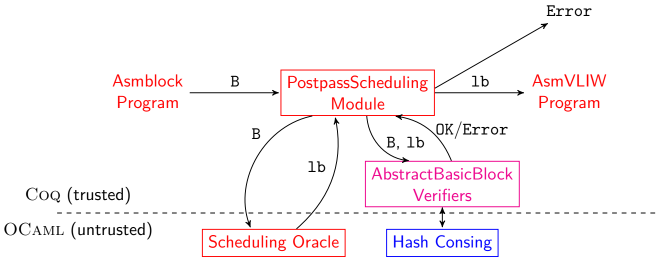
Highly-modular certified postpass scheduler in CompCert

using "untrusted-oracle / certified-verifier" architecture

Scheduling is **computed** by an *untrusted oracle*

basic-block B → inequality system —solver→ solution → bundle-list lb

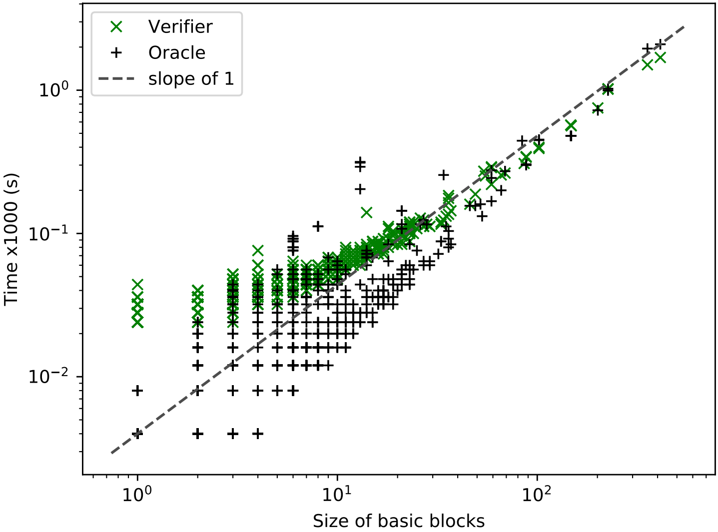
and **dynamically verified** (using symbolic evaluation of basic-blocks)



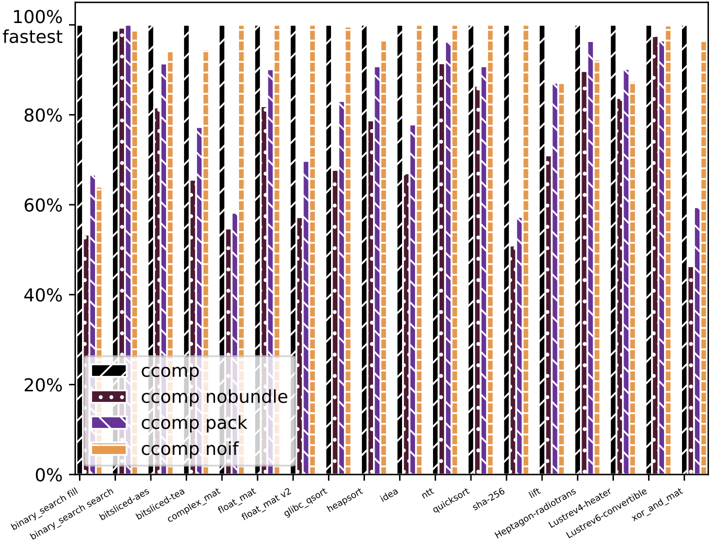
The solver is :

- by default, a greedy list scheduler (fast & near optimal)
- or, an ILP solver (optimal but very slow on some entries)

Compile-times (greedy list scheduler + its verifier)



Speedup due to our scheduler in CompCert



CompCert vs GCC on the Kalray-K1c