Formal Verification of C Programs with Floating-Point Computations: Certified Error Bounds for Signal Processing

Tahina Ramananandro
Reservoir Labs Inc., USA
ramananandro@reservoir.com

In this **technical talk**, I will present our results related to the certification of floating-point error bounds in C implementations of signal processing algorithms. In particular, this relates to the topic of **reasoning about the uncertainty** caused by the noise arising from floating-point rounding errors and approximate computations in C programs.

Our work, funded by the DARPA Microsystems Technology Office (MTO) in the Power Efficiency Revolution for Embedded Computing Technologies (PERFECT) program, is directed at assuring the performance of signal processing when compromises are done to reduce precision to save power. Our work allows one to provably bound the uncertainty introduced by the additional noise due to such compromises.

Technical approach We introduce VCFloat [9], a verification framework for floating-point computations in C programs, based on the Coq proof assistant [4], the Flocq [3] formal specification of IEEE 754 floating-point arithmetic, and the CompCert Clight [1, 6] formal semantics for a realistic subset of C.

Since Flocq and Clight are formal Coq specifications, our approach solely relies on Coq and their faithfulness, thus guaranteeing an unprecedented level of trust in the proofs of floating-point computations in C programs, compared to previous work [2] based on heterogeneous combinations of verification tools.

For a C program, we use CompCert to generate its Clight abstract syntax, then our VCFloat framework transforms C floating-point expressions into their real-number semantics with the appropriate rounding error terms, by automatically generating and checking their validity conditions using the Coq-Interval [7] tactic library for interval arithmetic. Thus, VCFloat solves all floating-point rounding issues, so that all remaining reasoning about error bounds can be done at the level of real numbers, using Coq-Interval to automatically compute error bounds and their proofs.

For instance, we can prove the correctness of a C implementation of approximate sine against the following Coq specification:

```
forall x,
   (is_finite x = true /\ Rabs (B2R x) <= 2147483647)
-> exists y,
   (eval_funcall FSIN (Vfloat x :: nil) (Vfloat y) /\
    is_finite y = true /\ Rabs (B2R y - sin (B2R x)) <= BOUND).
```

This specification states that, if the argument `x` is a valid floating-point number no greater than 2^{31}, then `FSIN`, the C implementation of approximate sine studied, does not crash, and produces a result within some absolute error bounded by `BOUND` of the ideal real-number sine. We compute `BOUND` within Coq at the same time as we build the correctness proof, using Coq-Interval and VCFloat.

Applications and Conclusion For concreteness, we demonstrate how our approach can provide certifications of realistic C implementations of Synthetic Aperture Radar (SAR) backprojection [5], particularly the safety of energy-efficient optimizations based on approximate implementations inspired from [8]. But our work can also apply to more general settings such as simulation and numerical algorithms in High-Performance Computing.

Interestingly, our work also pertains to the issue of **proofs that cross IP boundaries** because VCFloat, and also our correctness proof of SAR backprojection with patent-protected energy-efficient optimizations, build on proof libraries from a variety of providers, each with their own licensing policy, leading to a complex problem in itself, to determine what can be shared, published, and how.

Acknowledgments

Joint work with Paul Mountcastle, Benoît Meister and Richard Lethin, Reservoir Labs Inc. This work is sponsored in part by DARPA MTO as part of the PERFECT program (issued by DARPA/CMO under Contract No: HR0011-12-C-0123). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the DARPA or the U.S. Government.

References

[1] Sandrine Blazy and Xavier Leroy. Mechanized semantics for the Clight subset of the C language. *Journal of Automated Reasoning*, 43(3):263–288, 2009.

[2] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Wave equation numerical resolution: A comprehensive mechanical proof of a C program. *Journal of Automated Reasoning*, 50(4):423–456, 2013.

[3] Sylvie Boldo and Guillaume Melquiond. Flocq: A unified library for proving floating-point algorithms in Coq. In *Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on*, pages 243–252, July 2011.

[4] The Coq development team. The Coq proof assistant. `http://coq.inria.fr`, 1984–2015.

[5] Mita D. Desai and W. Kenneth Jenkins. Convolution backprojection image reconstruction for spotlight mode synthetic aperture radar. *Image Processing, IEEE Transactions on*, 1(4):505–517, Oct 1992.

[6] Xavier Leroy. Compcert. `http://compcert.inria.fr`, 2005–2015.

[7] Guillaume Melquiond. Coq-interval. `http://coq-interval.gforge.inria.fr/`, 2008–2015.

[8] Jongsoo Park, Ping Tak Peter Tang, Mikhail Smelyanskiy, Daehyun Kim, and Thomas Benson. Efficient backprojection-based synthetic aperture radar computation with many-core processors. In *Proceedings of Supercomputing '12*, 2012.

[9] Tahina Ramananandro, Paul Mountcastle, Benoît Meister, and Richard Lethin. A Unified Coq Framework for Verifying C Programs with Floating-Point Computations. In *5th ACM/SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, 2016.