This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

Real(istic) Specifications of Software

2021 Workshop on the Science of Scientific-Software Development and Use

Samuel D. Pollard [2], Ariel Kellison [1,2], John Bender [2], Heidi K. Thornquist [2], and Geoffrey C. Hulette [2]

[1]Cornell University, Ithaca, New York
[2]Sandia National Laboratories, Livermore, California *

Abstract: Scientific software is used for many high-impact areas such as disease and climate modeling. However, the vast majority of scientific codes do not have any machine-checkable notion of correctness. In many cases, correct behavior is checked through statistical methods, either with testing or uncertainty quantification (UQ). Since UQ can be done without understanding a program's source, it can only capture behavior and not intent of the underlying software. One promising technique to provide stronger evidence of correctness for numerical programs is to leverage domain knowledge by annotating programs with real-valued specifications about their expected behavior. These annotations can provide more optimization opportunities, better software documentation, and the ability to formally prove properties of software.

Challenge

Scientific software is used in high-consequence applications such as epidemic modeling [7] but evidence of correctness is not provided beyond the inherently limited approaches of software testing. Formal methods have been used with great success in many fields, such as distributed systems [5] and hardware design [6], but have not migrated to scientific software with the same success. One reason for this is the solution domain. For example, in hardware design, behavior for every bit is typically well-specified, or at least deterministic if unspecified. But modern scientific software software execute arithmetic in parallel, which typically permits many possible solutions (for example, with floating-point arithmetic). While there have been efforts to provide reproducible parallel code [3] these have large overhead and are typically not necessary: correctness should be defined by how closely a solution models reality rather than how closely it models a previous execution.

In practice, this notion of reality is not described anywhere formally but instead is verified by domain experts who check solutions, using either testing or sophisticated UQ techniques. One strength of these methods are their scalability with respect to the size of the software. However, testing can only provide an *absence of evidence of incorrectness*, and not *evidence of correctness.*

Opportunity

Figure 1 shows an annotation of a C function. In this program, the absolute error of the square root is bounded by 2^{-43}. In this case, the property is described using Frama-C and then dispatched to the prover Gappa. Currently, these approaches work well for small-scale codes [1] but do not work well for large programs. However, the use of formal methods is not "all or nothing." Further development of specifications can prevent large classes of errors caused by invalid inputs of a function, for example.

```
/*@ requires 0.5 <= x <= 2;
 ensures \abs(\result -\sqrt(x))
 <= 2^-43 * \abs(\sqrt(x));
*/
double sqrt(double x)
{
  // implementation
}
```

Figure 1: Example real-number specification using Gappa [2] and Frama-C [4].

[*] Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Beyond correctness, these types of annotations can aid code optimization. For example, given that $0.5 \leq x \leq 2$, the square root implementation need not perform checks to determine whether x is negative and may require fewer floating-point operations to achieve the error bound of 2^{-43}. While this example is simple, the result generalizes. For example, properties of a matrix such as symmetric positive definiteness permit entire classes of algorithms which do not converge for general matrices.

Timeliness & Maturity

Automated analysis of floating-point error has seen a resurgence in popularity in the last decade, brought about from two hardware trends resulting from the energy and computing requirements of the exascale age. The first is the vast increase of parallelism from modern architectures. The second is the emergence of more hardware heterogeneity such as multiprecision and non-IEEE 754 floating-point arithmetic.

Because of the size and complexity of scientific software, scientific simulation is seen as “just” a simulation, and must in turn be verified empirically. This need not, and should not, be the case. The scientific community must take steps to increase the trust in its computer programs so we need not waste precious time, resources, and lives verifying what a computer says with empirical evidence. *A computer program should be evidence enough.*

This is a tall order. Developers have vastly different ways in which to approach and solve problems and asking them to change can be disruptive. However, real-number specifications—and their corresponding machine-checked properties—can help these developers by encoding some of the wealth of information contained in scientists' minds.

References

[1] Boldo, S., Clément, F., Filliítre, J.-C., Mayero, M., Melquiond, G., and Weis, P. Trusting computations: A mechanized proof from partial differential equations to actual program. *Computers and Mathematics with Applications* (2014).

[2] de Dinechin, F., Lauter, C. Q., and Melquiond, G. Assisted verification of elementary functions using gappa. In *Proceedings of the ACM Symposium on Applied Computing* (Dijon, France, 2006), SAC '06, ACM, pp. 1318–1322.

[3] Demmel, J., Ahrens, P., and Nguyen, H. D. Efficient reproducible floating point summation and BLAS. Tech. Rep. UCB/EECS-2016-121, EECS Department, University of California, Berkeley, June 2016.

[4] Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., and Yakobowski, B. Frama-C: A software analysis perspective. *Formal Aspects of Computing 27* (May 2015), 573–609.

[5] Lamport, L. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[6] Russinoff, D. M. *Formal Verification of Floating-Point Hardware Design: A Mathematical Approach.* Springer, Cham, 2019.

[7] Sagoff, J. Argonne epidemiological supercomputing model showcases innovation, 2020.