State-Based Formal Methods in Scientific Computation

John Baugh(✉) and Tristan Dyer

Civil, Construction, and Environmental Engineering,
North Carolina State University, Raleigh, NC, USA
{jwb,atdyer}@ncsu.edu

Abstract. Control systems, protocols, and hardware design are among the most common applications of state-based formal methods, and yet the types of modeling and analysis they enable are also well-suited to problems in scientific computation, where quality, reproducibility, and productivity are growing concerns. We survey the challenges faced by developers of scientific software, characterize the nature of the programs they write, and offer some perspective on the role that state-based methods can play in scientific domains.

1 Introduction

Called a third pillar of science, computation is an indispensable tool not only for scientists, but for engineers who simulate physical and natural processes to evaluate design alternatives. Recent studies on reliability, reproducibility of results, and productivity have cast concern over what many have suspected or experienced firsthand, that existing practices of constructing scientific software are inadequate and limiting the pace of technological advancement. A disconnect between modern software engineering practice and scientific computation is apparent, and yet the unique challenges facing developers of scientific software must also be recognized: the lack of test oracles, software lifetimes and evolving needs that span decades, and the competing objectives of performance, maintainability, and portability.

We seek to address fundamental design and quality assurance challenges that are intrinsic to scientific computation and related types of numerical software. While numerous directions might be taken, our premise and motivating viewpoint is the central role that modeling can and must play in the process of designing and working with complex artifacts, including scientific programs. Culturally, the fit may be a natural one: scientists and engineers are accustomed to working with models anyway, and with the kind of automatic, push-button analysis supported by some state-based formalisms, those who develop software can focus on modeling and design instead of theorem proving.

M. Butler et al. (Eds.): ABZ 2018, LNCS 10817, pp. 392–396, 2018.
https://doi.org/10.1007/978-3-319-91271-4_29

2 Background

Despite broad and recognized impacts, the field of scientific computation faces a number of challenges. Meeting quality and reproducibility standards is a growing concern [10], as is productivity [6]. Not merely anecdotes, numerous empirical studies of software "thwarting attempts at repetition or reproduction of scientific results" have been cataloged in a recent article by Storer [9], along with their concomitant effects, including a widespread inability to reproduce results and subsequent retractions of papers in scientific journals. Productivity problems are also reported, which Faulk et al. [6] refer to as a *productivity crisis* because of "frustratingly long and troubled software development times" and difficulty achieving portability requirements and other goals.

Sources of difficulty may stem from fundamental characteristics of the problem domain, along with cultural and development practices within it. For instance, projects are often undertaken, as one might imagine, for the purpose of advancing scientific goals, so results may constitute novel findings that are difficult to validate. In the absence of test oracles, developers may have to settle for plausibility checks based on, say, conservation laws or other principles that are expected to hold. Then, if the software is successful, its lifetime may span a 20 or 30 year period, starting with development and then moving through hardware upgrades and evolving requirements that are intended to keep up with ongoing scientific advancements. Development priorities are such that traditional software engineering concerns, like time to market and producing highly maintainable code, may receive relatively less attention compared with performance and hardware utilization [6].

Proposals to address quality and productivity concerns are varied. Storer [9] places new and suggested approaches into broad categories of (a) software processes, including agile methods, (b) quality assurance practices, including testing, inspections, and continuous integration, and (c) design approaches, including component architectures and design patterns. In the category of quality assurance practices, he adds formal methods, noting a couple of experience reports, but also observing that such approaches have received considerably less attention in the scientific programming community, possibly due to "the additional challenge of verifying programs that manage floating point data."

3 Approach

Although the tools and techniques most identified with scientific computation are those of numerical analysis—where error prediction, stability, and convergence are central concerns—such an enterprise offers little guidance in the development process, where early decisions about decomposition and organization establish program structure. We suggest separating concerns, and lay out an approach informed by numerical analyses that allows scientists and engineers to represent and reason about the essential structure and behavior of the programs they create. The ideas are well-suited for lightweight tools like Alloy [7], a state-based formalism that combines declarative modeling and bounded model checking.

3.1 About Scientific Programs

We consider the application of state-based methods in a relatively uncharted domain, scientific computation, for which there is little community experience in working with formal methods. We might ask about the essential complexities, what they are, and whether formal methods might help. By way of contrast, when computer engineers model systems, they already have some experience in getting at these questions. So, for instance, when specifying a two-phase handshake protocol they know whether they can ignore what's going through the pipe: they generally have some sense of how and what to specify, and what to ignore. There is far less of this kind of experience with programs in scientific areas, so it is helpful to characterize what they are like.

When we refer to scientific computation, we think primarily of problems expressed as mathematical models, where approximate solutions are sought for differential or integral equations that have no closed form solution. As a result, they must be discretized to produce a finite system of equations that can then be solved by algebraic methods. Ocean circulation models, for instance, may be expressed as a system of partial differential equations of the hyperbolic type, and solved by finite element [11] or other numerical schemes. Because they represent aspects of the physical and natural world, the terms and parameters appearing in the equations capture rich state in the form of spatial, geometric, material, topological, and other attributes. The types of discretizations that may be employed in both time and space are varied, and each has its own performance, accuracy, and ease-of-development implications.

3.2 Separating Concerns

What we propose is something akin to the two-phase handshake protocol analogy where the data going through the pipe are, in this case, numerical expressions. We cannot ignore them, of course, but we aim to consider them separately, so we advance the following perspective:

$$\text{scientific programs} = \text{numerical expressions} + \text{interstitial machinery}$$

By *interstitial machinery* we mean the discrete data structures and algorithms throughout which numerical expressions are embedded. In many cases, the interstitial machinery is itself a complex apparatus, as we find in the class of problems above, and these are aspects of a program that warrant increased scrutiny and care. Correctness arguments for this part of scientific programs can be made without simultaneously reproducing the sometimes deep, semantic proofs of numerical analysis [8]. Instead, pertinent results may be brought into the modeling process in the form of invariants and other structural properties.

Beyond appealing to experience, a supporting idea for this claim is the following: the numerical analyses performed for scientific computations often apply, unchanged, throughout a broad range of implementation choices and modifications, changes in libraries and solvers, and diverse hardware upgrades, over the life of the program.

3.3 Examples

Applying this perspective, the following studies show how finite state models can be used to draw useful conclusions about scientific software:

Hurricane Storm Surge. Used in production by the U.S. Army Corps of Engineers and others, ADCIRC is a large-scale ocean circulation model that simulates hurricane storm surge. In this study [3], we consider implementation choices for a performance enhancement made by our group, and use models developed in Alloy to make guarantees about them, in particular that they are equivalence preserving. The study is motivated by complex interactions between the enhancement and ADCIRC's discrete wetting and drying algorithm, which operates on a finite element mesh to accommodate advancing and receding flood waters.

Coupled Earth Models. Numerical models of the earth capture interactions between atmospheric, ocean, land surface, sea ice, and other components, which execute concurrently and exchange data during runtime. By modeling read-write behavior and the timestamps associated with updates, race-free phasing arrangements can be generated, thereby preventing data from either being overwritten too soon or becoming stale. This approach is applied to a research prototype of simultaneously executing ocean circulation models for which the exchange of data must be coordinated [2].

Structural Analysis. Moment distribution [5] is an iterative technique, well-known among civil engineers, for finding the internal member forces that develop in building structures when external forces are applied to them. In its most general form, the method is similar to asynchronous, chaotic relaxation algorithms, where portions of a building structure converge numerically at differing rates as the computation unfolds, depending on process scheduling. The nondeterminism available here is also inherent in methods used to solve elliptic partial differential equations, which may exploit nondeterminism in different ways depending on problem characteristics and hardware features. In an unpublished specification that appears online [1], we make use of a numerical study [4] and predicate abstraction in a modeling approach that facilitates refinement checking.

The examples above span a range of scales from production to research software to what might be considered a toy problem, moment distribution, and yet the problems share features that suggest a role for state-based methods:

- Structure: by supporting *implicitness* in a specification, Alloy allows arbitrary spatial discretizations to be considered in the analysis, e.g., the varied topological relationships that exist in real building structures.
- Behavior: by not imposing fixed idioms, it can accommodate specifications of different styles and with different approaches to parallelism that may be encountered, e.g., in library interfaces like MPI, OpenMP, and OpenCL.

While other approaches might be considered, state-based methods like Alloy seem particularly appropriate for the types of modeling and analysis we describe, and for the support it provides for conceptual design.

4 Conclusions

Numerical concerns figure prominently in scientific computation, and yet the major sources of complexity in actual software, from our perspective, have more to do with the interstitial machinery that ties them together. Separating concerns, along the lines we have suggested, should allow state-based methods to find productive use in a domain that could benefit from the kind of modeling and push-button analysis they provide. Invariants and other structural properties often follow directly from numerical analyses, both for algorithms and for data structures, facilitating safety, liveness, and fairness checks that can be put together in a variety of ways beyond the ones we mention.

Given the fundamental role of computation in the conduct of modern science, the development and adoption of better design practices could have far-reaching benefits. Toward that end, we suggest a focus on essential complexities and scientifically relevant computational abstractions, as advocated by Faulk et al. [6], using precise and expressive notations that support exploration and analysis. Future work in this direction may lead to new insights and deeper understanding, as well as auxiliary tools and instructional materials that make these advances more accessible to scientists and engineers in traditional areas.

References

1. Alloy models from the paper. http://www4.ncsu.edu/~jwb/alloy/
2. Altuntas, A., Baugh, J.: Verifying concurrency in an adaptive ocean circulation model. In: Proceedings of the First International Workshop on Software Correctness for HPC Applications, Correctness 2017, pp. 1–7. ACM (2017)
3. Baugh, J., Altuntas, A.: Formal methods and finite element analysis of hurricane storm surge: a case study in software verification. Sci. Comput. Program. (2017, in press). https://doi.org/10.1016/j.scico.2017.08.012
4. Baugh, J., Liu, S.: A general characterization of the hardy cross method as sequential and multiprocess algorithms. Structures **6**, 170–181 (2016)
5. Cross, H.: Analysis of continuous frames by distributing fixed-end moments. In: Proceedings of the American Society of Civil Engineers, pp. 919–928 (1930)
6. Faulk, S., Loh, E., Van De Vanter, M.L., Squires, S., Votta, L.G.: Scientific computing's productivity gridlock: how software engineering can help. Comput. Sci. Eng. **11**(6), 30–39 (2009)
7. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press, Cambridge (2012)
8. Linz, P.: A critique of numerical analysis. Bull. Am. Math. Soc. **19**(2), 407–416 (1988)
9. Storer, T.: Bridging the chasm: a survey of software engineering practice in scientific programming. ACM Comput. Surv. (CSUR) **50**(4), 47:1–47:32 (2017)
10. Wilson, G.V.: Where's the real bottleneck in scientific computing? Am. Sci. **94**(1), 5–6 (2006)
11. Zienkiewicz, O.C., Taylor, R.L., Nithiarasu, P.: The Finite Element Method for Fluid Dynamics, 7th edn. Butterworth-Heinemann, Oxford (2013)