Formal Verification for Scientific Computing: Trends and Progress

Stephen F. Siegel

Verified Software Laboratory, Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA, siegel@cis.udel.edu

Science has been transformed by computation. Many observers now consider simulation a "third approach" to scientific discovery, on par with experimentation and theory. But scientists have long-established methods for verifying conclusions based on those traditional approaches—the same is not true for simulation. Reports on simulation-based research often say little about the qualities of the software used, or what was done to ascertain its correctness. Sometimes the software is not even made available to the public; but even when it is, how many would bother to check it, and how would they go about doing so?

Unfortunately, there is substantial evidence that the software used in science is as unreliable as any other type of software. The extensive case studies conducted by Les Hatton, for example, showed that scientific programs are often riddled with undiscovered defects. The most insidious defects do not cause the software to fail catastrophically—they lead to erroneous yet believable results.

Testing is still the most widely-used approach for verifying correctness. Yet Dijkstra warned long ago that testing can only show that a program is wrong, never that a program is right. There are other limitations to testing. First, for nondeterministic programs, including many parallel programs, a correct test result does not even guarantee a program will produce the correct result if run again on the same test. Second, for scientific programs, there is often no way to know what the correct result is supposed to be. Third, testing is expensive—typically consuming 50% of development effort—and in science often requires long run-times and access to expensive hardware.

Into this challenging landscape step *formal methods*, which use techniques grounded in formal logic to reason about program correctness. The ideas go back to the dawn of computer science, but recent advances are now starting to make these approaches practical. *Theorem proving* approaches are the most well-known; they have steadily improved, but still require enormous skill and significant user interaction. More recent *finite-state* techniques (often referred to as *model checking*) trade some of the completeness of theorem-proving for automation and usability. Finite-state verification tools for MPI-based programs include MPI-Spin and ISP. Finally, new approaches using *symbolic execution* deal with some of the most difficult issues, including reasoning about floating-point computations and determining whether two programs are functionally equivalent.

Like fundamental problems in other scientific disciplines, the verification problem will never have a complete solution. This only makes the problem more interesting, and the steady progress and explosion of new ideas testifies to its continuing importance and fascination.

M. Ropo et al. (Eds.): EuroPVM/MPI 2009, LNCS 5759, p. 7, 2009.