Techniques for the automatic debugging of scientific floating-point programs

David H. Bailey*, James Demmel†, William Kahan†, Guillaume Revy†, and Koushik Sen†

* Berkeley Lab Computing Sciences
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA

† Parallel Computing Laboratory
EECS Department
University of California, Berkeley
Berkeley, CA 94704-1776, USA

dhbailey@lbl.gov {demmel,wkahan,grevy,ksen}@eecs.berkeley.edu

Abstract

Over the past several years the field of large-scale scientific applications has been growing rapidly. Consequently the anomalies in these kinds of application, anomalies that heretofore had a minor impact, may have today a significant impact on the numerical results of these programs and their implications [1]. The work presented here proposes automatic techniques to reduce the cost of locating and remedying a wide class of numerical nuisances arising in single and multi-threaded applications.

As examples of common anomalies, let us cite rounding errors that can accumulate excessively all along a numerical program, conditional branches involving floating-point comparisons that may go astray because of the subtleties of floating-point arithmetic, anomalies due to vagaries of programming languages, overflow, benign and catastrophic cancellation, among others. When suspected, such anomalies can be located using various techniques: altering rounding modes of floating-point arithmetic hardware and observing the sensitivity of the program to those changes, increasing the precision of the calculations on some floating-point operations (by using high or even infinite precision) and observing the impact on the final result, modifying comparisons by adding an unobvious tolerance, or also using interval arithmetic, ... These techniques vary in their costs, scopes, and effectiveness.

Because anomalies due to roundoff are difficult to debug [2], we wish to offer developers whose expertise does not extend to numerical error-analysis an intelligent tool to debug floating-point programs. This tool should help locate automatically and remedy suspected anomalies, working on source code and at runtime, to shorten debugging time and thus improve the productivity of programmers.

Our tool embraces a set of transformations: increasing precision (by using double instead of single precision floating-point arithmetic), changing rounding mode, flipping between two implementations of the same computation, ... These transformations are effected by instrumenting the original code with CIL [3], which allows a given C code to be analysed and transformed. Then the parts of a program that are most sensitive to this transformation can be isolated automatically using delta-debugging [4]. This algorithm works like a binary search to determine a locally minimal

set of changes to be applied to the original code so that the returned result remains within a given tolerance of a known and more accurate result (higher or exact precision, ...). So far, these techniques have been validated mainly on codes of the LAPACK [1] library, especially on the bug in subprogram `dgges` [2] when its subroutine `dlarfg` was replaced by `dlarfp`. Of vastly many calls upon `dlarfp`, which one was first to malfunction? And for what data? Answers took days to find without the tool, minutes with it. Only then could the misbehavior of this relatively short subroutine `dlarfp` be analyzed and repaired.

References

[1] David H. Bailey. Resolving Numerical Anomalies in Scientific Computation. 2008. Available at `http://crd.lbl.gov/~dhbailey/dhbpapers/numerical-bugs.pdf`.

[2] William Kahan. How Futile are Mindless Assessments of Roundoff in Floating-point Computation? 2006. Available at `http://www.eecs.berkeley.edu/~wkahan/Mindless.pdf`.

[3] George C. Necula, Scott McPeak, S.P. Rahul, and Westley Weimer. CIL: Intermediate language and tools for analysis and transformation of C programs. In *Proceedings of Conference on Compiler Construction*, 2002.

[4] Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200, 2002.

Keywords: scientific floating-point program, automatic debugging, code transformation, delta-debugging algorithm.

[1] Linear Algebra PACKage - Available at `http://www.netlib.org/lapack`.

[2] See `http://icl.cs.utk.edu/lapack-forum/viewtopic.php?f=2&t=1783` for details.