Practical Floating-point Divergence Detection

Wei-Fan Chiang, Ganesh Gopalakrishnan (advisor), and Zvonimir Rakamarić (advisor)
School of Computing
University of Utah, Salt Lake City, UT
{ wfchiang, ganesh, zvonimir } @cs.utah.edu

Developing programs involving floating-point data comes with a great temptation of using low bit-width type (e.g., 32-bit) in order to achieve higher performance, including lower storage and memory bandwidth usage [1], [2]. However, floating-point arithmetic is highly non-intuitive, and using low precision tends to induce non-reproducible bugs and hard-to-debug situations [3], [4]. A specific problem caused by insufficient precision is output deviation between floating-point and real number computations. Such deviation typically implies an "incorrect" outcome in which the root-cause is usually the conversion of imprecise floating-point value to discrete data type. For example, in a program that draws a 2D convex polygon, floating-point values flowing into conditionals (e.g., if-else statemets) could cause control path deviations that eventually result in a non-convex polygon being drawn. This problem is called *divergence* and has been widely discussed in the literature [5], [6].

With the increasing role of geometry in critical applications (e.g., manufacture of prosthetics using 3D printing and robot motion planning), divergence becomes a life- and resource-critical issue, and must be systematically tackled. Previous approaches [7], [6] suggest setting up *padding values* to identify suspicious floating-point-to-discrete type conversions. However, the padding values need to be properly set by either a mathematical proof or a thorough testing, and generating a proof is usually tedious and error-prone. In our work, we focus on testing techniques that search for inputs that trigger divergences. The main challenges of testing are dealing with the sheer number of input combinations, non-uniform floating-point distribution, and layers of floating-point operations (e.g., non-linear operations). Previous approaches based on symbolic analysis [8], [9] suffer from poor scalability. On the other hand, we propose systematic search methods that, to the best of our knowledge, are the most scalable (in terms of input size) among all known methods. We identify two types of floating-point programs which have different types of discrete features. For each program type, we propose an effective search method that is likely to quickly trigger discrete feature deviations.

The first search method we propose is *abstract binary search* or ABS. It handles programs in which the numerical program output can be projected to a discrete value space. Take the 2D convex polygon drawing program as an example: we can map a polygon (program output) to an integer (discrete value) that counts the number of vertices in the polygon. For this type of programs, divergences usually cause deviations in the discrete values representing program output. For example, imprecise floating-point arithmetic maybe results in a 4-vertex non-convex polygon, while real number arithmetic results in a 3-vertex convex polygon. ABS starts searching from two random inputs which do not cause deviation but have distinct discrete representing values. From two such inputs, ABS derives/enumerates a new input by calculating the geometric midpoint of the original two inputs (by taking each input as a multi-dimensional vector), and then recursively continues this enumeration process.

The other search method we propose is *guided random testing* or GRT. It handles programs in which the program output must satisfy some post-condition described as a conjunction of inequalities. Satisfying the post-condition is the program's discrete feature. For example, calculating the variance of a set of numbers must return a non-negative result. For this type of programs, potential divergences must violate the post-condition. GRT employs our search engine S3FP [10], which efficiently enumerates inputs causing high error on some term(s) in the post-condition inequalities. Empirically, such high-error causing inputs usually violate the post-condition as well.

Our experimental results suggest that both ABS and GRT can efficiently find divergence-causing inputs for their respective program types. Table I summarizes our experimental results. We evaluated our approach on a benchmark suite containing realistic numerical routines that take thousands of floating-point numbers as input. We compared it against random testing, which is the only known method scalable enough to handle our benchmarks. Our benchmarks for evaluating ABS include 2D convex hull generation, shortest path computation, and primitives for deciding the geometric relation between objects. The results show that ABS detects divergences by enumerating only a few hundred inputs. This shows its efficiency as compared to random testing, which detects at most one divergence with 10^6 input enumerations. Our benchmarks for evaluating GRT include variance calculation, sorting, and prefix sum calculation. The results show that GRT detects divergences by enumerating only a few thousand inputs. It is still much more efficient than random testing, which usually fails to detect any divergences even with 10^6 input enumerations.

To conclude, our experimental results suggest that ABS and GRT are both efficient divergence detection methods capable of handling practical numerical routines. Our work is currently limited to detecting divergences solely caused by floating-point round-off errors. In the future, we plan to detect heterogeneity-induced divergence scenarios, which are output deviations among platforms using different floating-point arithmetic hardware.

References

[1] M. Taufer, O. Padron, P. Saponaro, and S. Patel, "Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs," in *IPDPS*, 2010, pp. 1–9.

Benchmark	# of Input Values	# Enum. to Detect One Div.	
		ABS	Random
Convex Hull	2000	468	>1000000
Shortest Path	2045	127	>1000000
3D Line x Triangles	18	624	>1000000
Triangle x Triangle	12	386	>1000000

(a) Results of ABS

Benchmark	# of Input Values	# Enum. to Detect One Div.	
		GRT	Random
Variance Calculation	10000	610	>1000000
Prefix Sum	8192	1030	>1000000
Standard Sorting	10000	242	>1000000
Approx. Sorting	10000	20600	>1000000

(b) Results of GRT

TABLE I: Experimental Results

[2] M. O. Lam, J. K. Hollingsworth, B. R. de Supinski, and M. P. LeGendre, "Automatically adapting programs for mixed-precision floating-point computation," in *ICS*, 2013, pp. 369–378.

[3] "Patriot missile failure," http://www.fas.org/spp/starwars/gao/im92026.htm.

[4] Q. Meng, A. Humphrey, J. Schmidt, and M. Berzins, "Preliminary experiences with the uintah framework on intel xeon phi and stampede," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (XSEDE)*, 2013, pp. 48:1–48:8.

[5] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap, "Classroom examples of robustness problems in geometric computations," *Computational Geometry: Theory and Applications*, pp. 61–78, 2008.

[6] C. Ericson, *Real-time collision detection*. Elsevier Amsterdam/Boston, 2005.

[7] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete & Computational Geometry*, pp. 305–363, 1996.

[8] Y. Gu, T. Wahl, M. Bayati, and M. Leeser, "Behavioral non-portability in scientific numeric computing," in *Parallel Processing - 21st International Conference on Parallel and Distributed Computing*, 2015, pp. 558–569.

[9] G. Paganelli and W. Ahrendt, "Verifying (in-)stability in floating-point programs by increasing precision, using SMT solving," in *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2013, pp. 209–216.

[10] W.-F. Chiang, G. Gopalakrishnan, Z. Rakamarić, and A. Solovyev, "Efficient search for inputs causing high floating-point errors," in *PPoPP*, 2014, pp. 43–52.