Floating-point Robustness Estimation by Concrete Testing

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

Computational errors caused by limited precision implementations of floating-point routines are a central concern in computing at all levels of scale ranging from high-end supercomputers through hand-held electronics. Software developers often allocate lower precision in order to gain performance. However, manually allocated precision may be inadequate, and can cause major failures such as [1], [2]. Unfortunately, current floating-point precision analysis methods are hard to scale to handle modern high performance computing (HPC) programs.

We focus on solving one of the many numerical precision issues: scalable floating-point robustness estimation. We define robustness differently from the definition in previous context [3]. Here comes our definition: Let a program \( P_{\text{float}} \) be given, where all variables are instantiated in finite precision. Let \( P_{\text{inf}} \) be the program with the same variables understood under infinite precision. Define a discrete outcome (a branch truth value or a floor calculation) to be divergent if \( P_{\text{float}} \) and \( P_{\text{inf}} \) calculate this outcome differently (e.g. one takes the branch, the other skips it). A divergence is a situation that such divergent outcome is observable on some discrete program output \(^1\), and the input causing the divergence is called divergence input. Define \( P_{\text{float}} \) to be more robust than \( P_{\text{float}}^2 \) if the number of divergences of the input space is lower for \( P_{\text{float}}^1 \) than for \( P_{\text{float}}^2 \). A realistic example of divergence was demonstrated in previous work [4], where a convex hull program could return a non-convex polygon as its output. Without properly analyzing the robustness, floating-point programs often tend to generate divergent outputs unexpectedly.

To provide a robustness estimation framework that assists developers to deal with the performance-robustness trade-off, we propose a concrete-testing-driven solution. Concrete testing has high scalability and incorporates effects of compilation flags or hardware. It can also provide concrete inputs for experiments, we found that the sizes of the generated sub-domains are very large, containing more than half a million of inputs. Such large sub-domains may restrict the applicability of our robustness estimation method, and exploring potential solutions is an area of future work.

We evaluated our robustness estimation method on a geometric primitive that decides 3D point orientation: given four 3D points, it decides the orientation of the fourth point with respect to the plane formed by the first three points. This primitive can be implemented by either checking the sign of the determinant of a 3x3 matrix or of a 4x4 matrix. The 3x3 method was argued to be more robust in previous work [9]. Our robustness estimation results match this argument: by fixing the determinant of a 3x3 matrix or of a 4x4 matrix, the 3x3 method was argued to be more robust in previous work [9]. Our robustness estimation results match this argument: by fixing the determinant of a 3x3 matrix or of a 4x4 matrix, the 3x3 method was argued to be more robust in previous work [9].

To conclude, our preliminary results suggest that our concrete-testing-based method is a promising approach for floating-point robustness estimation. As future work, we plan to explore potential improvements of scalability efficiency of our framework, and apply our method to more real-world applications such as convex hull construction. We also plan to explore methodologies of boundary identification which, in our current framework, relies on user-specified conditionals.

\(^1\)We refer to integers, booleans, or discrete objects.
REFERENCES


