Precision Tuning by Static Analysis

Dorra Ben khalifa[1] and Matthieu Martel[1]

[1] University of Perpignan, LAMPS Laboratory, France
first.last@univ-perp.fr

Keywords: Numerical accuracy, computer arithmetic, compiler optimization, static analysis

As more and more critical decisions and tasks relying on complex computations are delegated to machines, the needs for design methods as well as verification and validation techniques strongly increase, to ensure that the numbers given by the computers may be trusted. While most of existing work concerns verification and validation techniques, assisted design method are strongly desired since it is extremely difficult to understand the reasons why the implementation of a formula is numerically inaccurate and how to improve it. This is because the floating-point arithmetic is particularly not intuitive. The values have a finite number of digits and algebraic laws like associativity or commutativity do not hold. It is then necessary to provide tools to the programmers, to help them to validate and increase the numerical quality of their codes and, broadly, to develop more quickly more reliable numerical codes.

Recently, program transformation techniques [2] have been developed to improve the accuracy of numerical codes [2, 5] as well as precision tuning tools [6, 3, 1, 4]. In the present work, we focus on the latter subject. Precision tuning consists of determining the minimal formats of the numbers used in a program in order to ensure some accuracy specified by the user on the results of the computation. This allows compilers to select the most appropriate formats (for example IEEE754 half, single, double or quad formats) for each variable. It is then possible to save memory, reduce CPU usage and use less bandwidth for communications whenever distributed applications are concerned. Our technique for precision tuning is also easily generalizable to the fixed-point arithmetic for which it is important to determine data formats, for example in FPGAs.

Our approach to precision tuning combines a forward and a backward static analysis, done by abstract interpretation. The forward analysis propagates safely the errors on the inputs and on the results of the intermediary operations in order to determine the accuracy of the results. Next, based on the results of the forward analysis and on assertions indicating which accuracy the user wants for the outputs at some control points, the backward

analysis computes the minimal precision needed for the inputs and intermediary results in order to satisfy the assertions.

We express the forward and backward transfer functions as a set of constraints made of propositional logic formulas and relations between affine expressions over integers (and only integers, even if the analyzed program contains non-linear computations.) As a consequence, these constraints can be easily checked by a SMT solver (we use Z3 in practice). The advantage of the solver appears in the backward analysis, when one wants to determine the precision of the operands of some binary operation $x * y$ with $* \in \{+, -, \times, \div\}$, in order to obtain a certain accuracy on the result. In general, it is possible to use a more precise x with a less precise y or, conversely, to use a more precise y with a less precise x. Because this choice arises at almost any operation, there is a huge number of combinations on the admissible formats of all the data in order to ensure a given accuracy on the results. Instead of using an ad-hoc heuristic, we encode our problem as a set of constraints and we let a well-known, optimized solver generate a solution.

References

[1] W.-F. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamaric, *Rigorous floating-point mixed-precision tuning*, In *POPL*, pages 300–315. ACM, 2017.

[2] N. Damouche, M. Martel, and A. Chapoutot, *Improving the numerical accuracy of programs by automatic transformation*, STTT, 19(4):427–448, 2017.

[3] E. Darulova and V. Kuncak, *Sound compilation of reals*, In POPL '14 , pages 235–248. ACM, 2014.

[4] M. Martel, *Floating-point format inference in mixed-precision*, In NFM, volume 10227 of LNCS, pages 230–246, 2017.

[5] P. Panchekha, A. Sanchez-Stern, J. R. Wilcox, and Z. Tatlock, *Automatically improving accuracy for floating point expressions*, In PLDI, pages 1–11. ACM, 2015.

[6] C. Rubio-Gonzalez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, *Precimonious: tuning assistant for floating-point precision*, In HPCNSA, pages 27:1–27:12. ACM, 2013.