Reduced-Precision Floating-Point Analysis

Michael O. Lam
Department of Computer Science
James Madison University
lam2mo@cs.jmu.edu

Jeffrey K. Hollingsworth
Department of Computer Science
University of Maryland, College Park
hollings@cs.umd.edu

Floating-point computation is ubiquitous in scientific computing, but rounding error can compromise the results of extended calculations. In previous work [2], we presented techniques for automated mixed-precision analysis and configuration. In this poster, we present new techniques that use binary instrumentation and modification to do fine-grained floating-point precision analysis, simulating any level of precision less than or equal to the precision of the original program. These techniques have lower overhead and provide more general insights than previous mixed-precision analyses. We also present a novel histogram-based visualization of a program's floating-point precision sensitivity.

Our technique builds a version of a target program where every instruction can be executed at any level of precision lower than the original precision (i.e., "reduced" precision). Our techniques are implemented in the Configurable Runtime Analysis for Floating-Point Tuning (CRAFT) framework, a freely available software library for building floating-point binary analysis tools [1]. Using an automated search, we detect the smallest level of precision with which any particular instruction can run and still pass verification, assuming that the rest of the program retains its original precision. The results of the automated search give an indication of the precision level requirement of that instruction, which can be generalized to larger structures by taking the maximal value across all children.

To reduce the precision of an individual instruction, we augment it by adding code after the instruction that uses bit masking to truncate the resulting floating-point value to the desired number of bits. This analysis imposes far less overhead than the analysis performed by the mixed-precision analysis described in previous work [2]. In part, this reduced overhead is because the inserted code is simpler, requiring a smaller amount of state to be saved and avoiding the addition of floating-point conversions. We show that running a search with this type of analysis is significantly faster than running the corresponding search with mixed-precision analysis.

Using this binary truncation technique, we developed an automatic search routine that determines the general precision sensitivity of various program components. The technique uses a breadth-first search, similar to the mixed-precision search described in previous work [2]. It first determines (using a single binary search on the range 0–52) the precision level required at the program level. Then, it explores the program's sub-program components by doing binary precision searches on each subcomponent. For each binary search, the results of all floating-point instructions in the component are truncated to the precision being tested, while all other components remain at their original precision. The search gradually descends through various types of subcomponents, beginning with modules and continuing through functions and basic blocks until it reaches individual instructions.

Previous work [2, 4] reports mixed-precision replacement rates representing the percentage of individual floating-point instructions (or variables) that could be replaced with single-precision and still pass verification. In this work, we provide a novel histogram-based visualization of whole-program sensitivity, enabled by the binary analysis and searching process described earlier. The histograms provide an overview of a program's floating-point sensitivity profile, showing the percentage of floating-point instructions executed (vertical axis) binned into ranges by their final reduced-precision configuration value (horizontal axis). The exact values on the vertical axis are less important than the relative sizes of the bars and their location on the horizontal axis. This visualization provides an easily digestible, at-a-glance overview of a program's precision level sensitivity.

If the distribution lies mainly to the left of the red bar, then the program is rarely dependent on full double precision. Conversely, a clustering around the extreme right side of the graph indicates that most of the program relies on full double precision, indicating little chance for a mixed-precision implementation without major rewriting. A clustering around the red bar is perhaps the most interesting outcome, since it implies that the program needs just barely more than single precision accuracy, implying that perhaps a small amount of algorithmic reconfiguration could enable the use of single-precision arithmetic for large portions of the computation.

In the poster, we present overhead results, a demonstration of the side-by-side comparison of histograms, and the results of an example case study on the CLAMR application [3]. The results provide evidence of the usefulness and efficacy of our system.

This work is supported in part by DOE grants DE-CFC02-01ER25489, DE-FG02-01ER25510 and DE-FC02-06ER25763.

1. REFERENCES

[1] CRAFT: Configurable Runtime Analysis for Floating-point Tuning. http://sourceforge.net/projects/crafthpc/. Accessed 1 October 2015.

[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 *Proceedings of the 27th International ACM Conference on Supercomputing (ICS '13)*, page 369, New York, New York, USA, June 2013. ACM Press.

[3] D. Nicholaeff, N. Davis, and R. Robey. Cell-based Adaptive Mesh Refinement on Hybrid Architectures. Technical report, Los Alamos National Laboratory, 2013.

[4] C. Rubio-González, 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 *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on (SC'13)*, pages 1–12, New York, New York, USA, Nov. 2013. ACM Press.