# Reduced-Precision Floating-Point Analysis

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

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

## Abstract

**Floating-point computation is ubiquitous in scientific computing, but rounding error can compromise the results of extended calculations.** In previous work [1], we presented techniques for automated mixed-precision analysis and configuration.

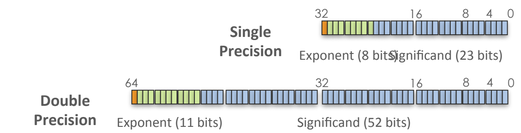
**We now present new techniques that use binary instrumentation and modification to do fine-grained floating-point precision analysis, simulating any level of precision less than or equal to the precision of the original program.** These techniques have lower overhead and provide more general insights than previous mixed-precision analyses. We also present a novel histogram-based visualization of a program's floating-point precision sensitivity, and an incremental search technique that gives the user more control over the precision analysis process.

## Background

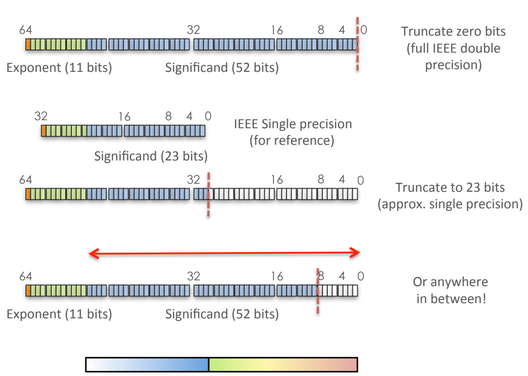
Floating-point arithmetic represents real numbers as

$(\pm\ 1.frac \times 2^{exp})$

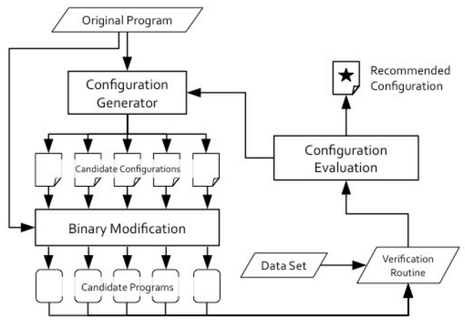
- Sign bit
- Exponent
- Significand ("mantissa" or "fraction")




## Reduced Precision




## Methods and System Overview



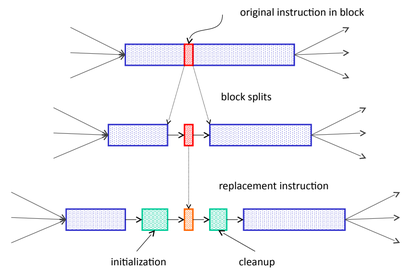

We extend our previous system [1] for auto-tuning the floating-point precision levels of a target application.

Given a representative data set and a user-defined verification routine, our system uses a systematic search to build multiple reduced-precision configurations of the application and evaluate them.

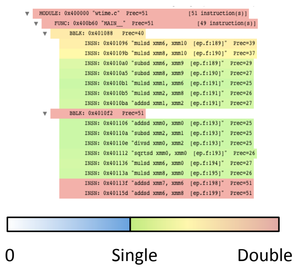
As output, the system produces a list of all the instructions in the program along with the lowest level of precision to which that instruction can be truncated while still passing verification (with all other computation held to the original precision).

The results can be visualized as a simple list of instructions and their estimated precision requirements, or with a histogram of the full precision profile of an application. These are shown in the figures below.

To replace instructions, we modify the binary by splitting the original program's basic blocks around the instruction, adding new blocks containing the replacement code and re-arranging edges between blocks.



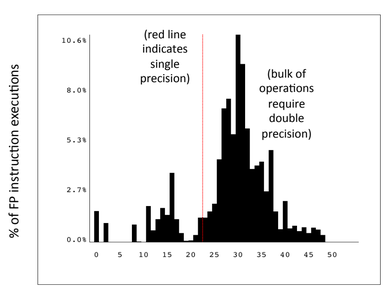

### Precision Requirement Instruction Lists




0 Single Double

Previous work [1,3] only reported binary "single or double precision" results for each instruction or variable. Our new analysis gives a much finer-grained insight into an instruction's precision sensitivity.

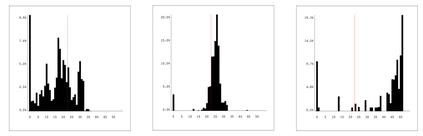
### Precision Requirement Histograms




# of bits in truncated significand

**NAS SP.A**

### At-A-Glance Comparisons



**NAS BT.A**

(more amenable to mixed-precision)

**NAS MG.A**

(in between)

**LAMMPS RHODO**

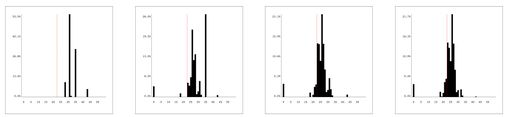
(less amenable to mixed-precision)

Precision requirement histograms provide a quick way to compare the precision requirements across different functions, modules, or even data sets. Left-weighted distributions suggest code that is more amenable to mixed-precision calculation, while right-weighted distributions suggest code that is less amenable to mixed-precision calculation.

## Result: Faster Searching

| App | Wall Time (S) Mixed [1] | Reduced | Speedup |
|---|---|---|---|
| cg.A | 1,305 | 532 | 59.2% |
| ep.A | 978 | 562 | 42.5% |
| ft.A | 825 | 411 | 50.2% |
| lu.A | 514,332 | 68,386 | 86.7% |
| mg.A | 2,898 | 984 | 66.0% |
| sp.A | 422,371 | 236,055 | 44.1% |

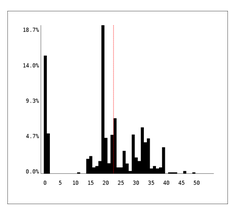
This analysis imposes less overhead (above) than the mixed-precision analysis described in previous work [1]. This is largely because the inserted code is simpler, requiring a smaller amount of state to be saved and avoiding floating- point conversion. Further, the automated search can be run incrementally (below), terminating the search early when it begins to converge.



5% (5m) 0.5% (10m) 0.05% (24m) Full (29m)

## Result: Case Study

We analyzed CLAMR [2], a proxy application for cell-based adaptive mesh refinement. Testing on the CPU-only version, we ran the automated search described above to determine the approximate precision requirements of each instruction and function in the program.




**CLAMR**

We found a significant reduction in storage costs. For the default problem size, the storage reduction is around 25%; for additional physics, the memory savings would approach 50% (a factor of two).

In the end, the code author built three versions of the CLAMR code at three different precision levels. Our analysis helped to inform the decisions on how to create the three levels in a consistent manner.

CRAFT is open source (LGPLv3).

Visit our source code repository (scan the QR code!) to download the source code for the analysis framework described in this poster as well as other floating-point analysis tools:

`http://sourceforge.net/projects/crafthpc/`

**References:**

[1] Lam, M. O., Hollingsworth, J. K., de Supinski, B. R., & Legendre, M. P. (2013). Automatically Adapting Programs for Mixed-Precision Floating-Point Computation. In Proceedings of the 27th International ACM Conference on Supercomputing (ICS '13).

[2] Nicholaeff, D., Davis, N., & Robey, R. (2013). Cell-based Adaptive Mesh Refinement on Hybrid Architectures. Retrieved from http://www.github.com/losalamos/ExascaleDocs

[3] Rubio-González, C., Nguyen, C., Nguyen, H. D., Demmel, J., Kahan, W., Sen, K., .. Hough, D. (2013). Precimonious: Tuning Assistant for Floating-Point Precision. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on (SC'13).