Modifying Floating-Point Precision with Binary Instrumentation

Michael Lam
University of Maryland, College Park

Jeff Hollingsworth, Advisor
Floating-point represents real numbers as \((\pm \text{sgnf} \times 2^{\text{exp}})\)

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

Floating-point numbers have finite precision
- Single-precision: 24 bits (~7 decimal digits)
- Double-precision: 53 bits (~16 decimal digits)
Example

1/10 ➞ 0.1

0x3DCCCCCD = 00111101 11001100 11001100 11001101

Single-precision

0xFB99999999999A = 00111111 10111001 10011001 10011001 10011001 10011001 10011001 10011010

Double-precision

Images courtesy of BinaryConvert.com
Motivation

• Finite precision causes round-off error
  o Compromises “ill-conditioned” calculations
  o Hard to detect and diagnose

• Increasingly important as HPC scales
  o Double-precision data movement is a bottleneck
  o Streaming processors are faster in single-precision (~2x)
  o Need to balance speed (singles) and accuracy (doubles)
Previous Work

• Traditional error analysis (Wilkinson 1964)
  o Forwards vs. backwards
  o Requires extensive numerical analysis expertise

• Interval/affine arithmetic (Goubault 2001)
  o Conservative static error bounds are largely unhelpful

![Graph showing value over time with high and low bounds]
Previous Work

• Manual mixed-precision (Dongarra 2008)
  o Requires numerical expertise

1: \( \text{LU} \leftarrow \text{PA} \)
2: solve \( \text{Ly} = \text{Pb} \)
3: solve \( \text{Ux}_0 = y \)
4: for \( k = 1, 2, \ldots \) do
5: \( r_k \leftarrow b - A x_{k-1} \)
6: solve \( \text{Ly} = P r_k \)
7: solve \( \text{Uz}_k = y \)
8: \( x_k \leftarrow x_{k-1} + z_k \)
9: check for convergence
10: end for

Red text indicates steps performed in double-precision (all other steps are single-precision)

Mixed-precision linear solver algorithm

• Fallback: ad-hoc experiments
  o Tedious, time-consuming, and error-prone
Our Goal

Develop automated analysis techniques to inform developers about floating-point behavior and make recommendations regarding the precision level that each part of a computer program must use in order to maintain overall accuracy.
Framework

CRAFT: Configurable Runtime Analysis for Floating-point Tuning

• Static binary instrumentation
  o Controlled by configuration settings
  o Replace floating-point instructions with new code
  o Re-write a modified binary

• Dynamic analysis
  o Run modified program on representative data set
  o Produces results and recommendations
Advantages

• Automated
  o Minimize developer effort
  o Ensure consistency and correctness

• Binary-level
  o Include shared libraries without source code
  o Include compiler optimizations

• Runtime
  o Dataset and communication sensitivity
Implementation

- Current approach: in-place replacement
  - Narrowed focus: doubles → singles
  - In-place downcast conversion
  - Flag in the high bits to indicate replacement
Example


1. movsd 0x601e38(%rax, %rbx, 8) ➞ %xmm0

2. mulsd -0x78(%rsp) ➞ %xmm0

3. addsd -0x4f02(%rip) ➞ %xmm0

4. movsd %xmm0 ➞ 0x601e38(%rax, %rbx, 8)
**Example**

\[ \text{gvec}[i,j] = \text{gvec}[i,j] \times \text{lvec}[3] + \text{gvar} \]

1. \texttt{movsd 0x601e38(%rax, %rbx, 8) \Rightarrow %xmm0}
   
   \textit{check/replace -0x78(%rsp) and %xmm0}

2. \texttt{mulss -0x78(%rsp) \Rightarrow %xmm0}
   
   \textit{check/replace -0x4f02(%rip) and %xmm0}

3. \texttt{addss -0x20dd43(%rip) \Rightarrow %xmm0}

4. \texttt{movsd %xmm0 \Rightarrow 0x601e38(%rax, %rbx, 8)}
Implementation

original instruction in block

block splits

double → single conversion

initialization cleanup
Configuration
Autoconfiguration

• Helper script
  o Generates and tests a variety of configurations
  o Keeps a “work queue” of untested configurations
  o Brute-force attempt to find maximal replacement

• Algorithm:
  o Initially, build individual configurations for each function and add them to the work queue
  o Retrieve the next available configuration and test it
    o If it passes, add it to the final configuration
    o If it fails, build individual configurations for any child members (basic blocks, instructions) and add them to the queue
  o Build and test the final configuration
NAS Benchmarks

• EP (Embarrassingly Parallel)
  o Generate independent Gaussian random variates using the Marsaglia polar method

• CG (Conjugate gradient)
  o Estimate the smallest eigenvalue of a matrix using the inverse iteration with the conjugate gradient method

• FT (Fourier Transform)
  o Solve a three-dimensional partial differential equation (PDE) using the fast Fourier transform (FFT)

• MG (MultiGrid)
  o Approximate the solution to a three-dimensional discrete Poisson equation using the V-cycle multigrid method
## Results

<table>
<thead>
<tr>
<th>Benchmark (name.CLASS)</th>
<th>Configs Tested</th>
<th>Instructions Replaced (Static)</th>
<th>Instructions Replaced (Dynamic)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>ep.A</td>
<td>19</td>
<td>85.1%</td>
<td>80.0%</td>
<td>3.4X</td>
</tr>
<tr>
<td>ep.C</td>
<td>19</td>
<td>85.1%</td>
<td>80.0%</td>
<td>5.5X</td>
</tr>
<tr>
<td>cg.A</td>
<td>82</td>
<td>64.6%</td>
<td>8.6%</td>
<td>3.4X</td>
</tr>
<tr>
<td>cg.C</td>
<td>54</td>
<td>75.8%</td>
<td>6.2%</td>
<td>4.5X</td>
</tr>
<tr>
<td>ft.A</td>
<td>65</td>
<td>56.5%</td>
<td>0.2%</td>
<td>4.2X</td>
</tr>
<tr>
<td>ft.C</td>
<td>65</td>
<td>57.8%</td>
<td>0.1%</td>
<td>7.0X</td>
</tr>
<tr>
<td>mg.A</td>
<td>100</td>
<td>77.0%</td>
<td>35.5%</td>
<td>5.8X</td>
</tr>
<tr>
<td>mg.C</td>
<td>104</td>
<td>75.4%</td>
<td>33.6%</td>
<td>14.7X</td>
</tr>
</tbody>
</table>

Results shown for final configuration only. All benchmarks were 8-core versions compiled by the Intel Fortran compiler with optimization enabled. Tests were performed on the Sierra cluster at LLNL (Intel Xeon 2.8GHz w/ twelve cores and 24 GB memory per node running 64-bit Linux).
Observations

• randlc / vranlc
  o Random number generators are dependent on a 64-bit floating-point data type

• Accumulators
  o Multiple operations concluded with an addition
  o Requires double-precision only for the final addition

• Minimal dataset-based variation
  o Larger variation due to optimization level
Future Work

Automated configuration tuning

SYSTEM INPUTS

Threshold = 0.10

Original program & error threshold
(all double-precision)

Candidate replacement configurations
(shaded = single-precision)

Candidate errors
(compared to original)

Chosen configuration
(under threshold w/ max replacement)
Conclusion

Automated instrumentation techniques can be used to implement mixed-precision configurations for floating-point code, and there is much opportunity in the future for automated precision recommendations.
Acknowledgements

Jeff Hollingsworth, University of Maryland (Advisor)

Drew Bernat, University of Wisconsin

Bronis de Supinski, LLNL
Barry Rountree, LLNL
Greg Lee, LLNL
Matt Legendre, LLNL
Dong Ahn, LLNL

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