Floating-Point Analysis Tools

Mike Lam

James Madison University
Motivation

- IEEE floating-point arithmetic
  - Ubiquitous in scientific computing
  - More bits => higher accuracy (usually)
  - Fewer bits => higher performance (usually)
Issues

• Roundoff error
  – Finite-width field; must round to fit
  – Not uniform across range

• Cancellation
  – Loss of significant digits due to subtraction

• Non-associativity
  – Optimization can impact accuracy
Historical Approaches

- Forwards and backwards error analysis
- Interval and affine arithmetic
- “Just use doubles”
Modern Tools

- Static vs. dynamic
- Level of analysis
  - AST (“source-level”)
  - Linear IR
  - Assembly/machine code
- Sound vs. unsound
- Varying degrees of scalability
Tool Categories

- **Error bound**
  - Sound rounding error guarantees on small subroutines

- **Accuracy improvement**
  - Re-write portions of your program to reduce rounding error

- **Dynamic analysis**
  - “Quick & dirty” info about your program (e.g., cancellations)

- **New format prototyping**
  - Experiment with alternative representations (e.g., float or posit)

- **Mixed precision**
  - New version of program that uses single precision in some places for a speedup (e.g., 20% improvement on LULESH)
Tools

• Error Bound
  – FLUCTUAT
  – Gappa
  – CompCert
  – Astrée
  – FPTaylor
  – Rosa
  – Real2Float
  – PRECiSA

• Accuracy Improvement
  – Herbie/Herbgrind
  – Salsa

• Dynamic Analysis
  – CRAFT
  – S3FP
  – FLiT

• New Format Prototyping
  – SHVAL
  – Template Library

• Mixed Precision
  – CRAFT
  – Precimionous
  – FPTuner
  – Daisy
  – ADaPT

tinyurl.com/fpanalysis
Error Bound Tools

- Sound upper bound on output error
  - Sometimes other properties as well
- Mostly from PL community
  - Thus, mostly based on static analysis

- FLUCTUAT (2002)
- Gappa (2005)
- CompCert & Astrée (2015)
- FPTaylor (2015)
- Rosa (2017)
- Real2Float (2017)
- PRECiSA (2017)
Error Bound Tools

Mostly unusable on HPC codes
– May be useful for proving properties of microkernels or numerical library routines

From https://dl.acm.org/citation.cfm?doid=3034774.3015465
Accuracy Improvement Tools

• Transform code to improve accuracy
• Herbie (2015)
  – Improve accuracy of Scala code
  – Web interface: https://herbie.uwplse.org
  – Herbgrind: Valgrind plugin for C codes

\[ z = \sqrt{x+1} - \sqrt{x} \]

\[ z = \frac{1}{\sqrt{x+1} + \sqrt{x}} \]

• Salsa (2016)
  – Improve accuracy of C-like DSL
  – Unavailable
FPBench

Compare FP tools: ~ 100 accuracy benchmarks

<table>
<thead>
<tr>
<th>Benchmark sources</th>
<th>Features used</th>
<th>Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosa</td>
<td>Arithmetic</td>
<td>Textbooks</td>
</tr>
<tr>
<td>Herbie</td>
<td>Temporaries</td>
<td>Mathematics</td>
</tr>
<tr>
<td>Salsa</td>
<td>Comparison</td>
<td>Controls</td>
</tr>
<tr>
<td>FPTaylor</td>
<td>Loops</td>
<td>Science</td>
</tr>
<tr>
<td></td>
<td>Exponents</td>
<td>(unknown)</td>
</tr>
<tr>
<td></td>
<td>Trigonometry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Conditionals</td>
<td></td>
</tr>
</tbody>
</table>

Interop to compose tools: FPCore, FPImp (w/ C & Scala transl)

Growing community: Utah, JM, MPI, UW, etc.

fpbench.org

Credit: Pavel Panchekha (me@pavpanchekha.com)
Dynamic Analysis Tools

• Runtime heuristic tools
  – Usually at a compiled/binary level

• Unsound “quick and dirty” approaches
  – Results only valid for tested inputs

• Can inform mixed-precision efforts or help find bugs and numerical instabilities
CRAFT (2011)

- Dynamic binary analysis via Dyninst
- Cancellation detection
- Range (exponent) tracking
CRAFT (2013)

- Reduced-precision analysis
  - Simulate reduced precision via truncation
  - Report minimum precision for output of each instruction (w/ user-provided verification routine)
  - Generate histograms of precision requirements
CRAFT (2013)

• Scalability via heuristic search: focus on most-executed instructions

https://github.com/crafthpc/craft
S3FP (2014)

• Guided fuzzing finds program inputs that maximize floating-point error
  – Improves scalability to “real” codes (1200x1200 LAPACK routines on GPU)
  – Finds (much) higher errors than random testing

[Graph showing input vs. floating-point error]

github.com/soarlab/S3FP

Credit: Ganesh Gopalakrishnan (ganesh@cs.utah.edu)
FLiT (2017)

FLiT is a reproducibility test framework in the PRUNERS toolset (pruners.github.io).

Hundreds of compilations are compared against a baseline compilation.

Credit: Ganesh Gopalakrishnan (ganesh@cs.utah.edu)
Prototyping New Formats

- IEEE single or half precision
- Arbitrary precision (e.g., MPFR)
- “Next generation” formats
  - John Gustafson’s Unum and Posit formats
  - Peter Lindstrom’s ZFP compression
  - See CoNGA’18 talk
    - https://www.youtube.com/watch?v=snbBRzu4LVI
SHVAL (2016)

- SHadow Value Analysis Library ([github.com/lam2mo/shval](https://github.com/lam2mo/shval))
- Pintool for simulating alternative representations
- Maintains shadow values for all memory locations
- Executes shadow operations for all computation

---

**Original machine code:**

```c
double sum = 0.0;
for (int i = 0; i < 10; i++) {
    sum += 0.1;
}
printf("%25.20f\n", sum);
```

**Inserted shadow code:**

```c
pxor xmm0, xmm0  // (set to 0.0)
mov eax, 10
movsd xmm1, 0x400628  // (load 0.1)
loop:
    sub eax, 1
    addsd xmm0, xmm1  // (increment)
    jne loop
    movsd 0x8(rsp), xmm0  // (store sum)
    xmm[0] = convert(0.0)
    xmm[1] = convert(* (0x400628))
    xmm[0] += xmm[1]
    mem[ esp + 0x8] = xmm[0]
```

**Fig. 3. Sample C program**

**Fig. 4. Compiled assembly of program from Figure 3**

---

**Table 1: Analysis results on sample program**

<table>
<thead>
<tr>
<th>Shadow Value Type</th>
<th>Exp Size</th>
<th>Frac Size</th>
<th>Final Shadow Value</th>
<th>Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit (native single)</td>
<td>8</td>
<td>23</td>
<td>1.000000</td>
<td>1.19e-07</td>
</tr>
<tr>
<td>64-bit (native double)</td>
<td>11</td>
<td>52</td>
<td>1.000000000000000000000000</td>
<td>0</td>
</tr>
<tr>
<td>128-bit GNU MPFR</td>
<td>15</td>
<td>112</td>
<td>1.000000000000000000000055</td>
<td>1.11e-16</td>
</tr>
<tr>
<td>Unum (3,2)</td>
<td>8</td>
<td>4</td>
<td>(0.9375, 1.1875)</td>
<td>0.059</td>
</tr>
<tr>
<td>Unum (3,4)</td>
<td>8</td>
<td>16</td>
<td>(0.9999847412109375, 1.000045776371875)</td>
<td>1.53e-05</td>
</tr>
<tr>
<td>Unum (4,6)</td>
<td>16</td>
<td>64</td>
<td>1.000000000000000000000551...182</td>
<td>1.11e-16</td>
</tr>
</tbody>
</table>
SHVAL (2017)

• Aggregate error by instruction or memory location over time
  – Higher overhead, more information

• Apriltags case study using mixed precision
  – 1.7x speedup on average with only 4% error
  – 40% energy savings in embedded experiments

Fig. 1. Error trace per memory location. A darker pixel indicates higher error.

Fig. 8. Tradeoff between performance gain and error for every function in the Apriltags library.

Credit: Ramy Medhat (ramy.medhat@uwaterloo.ca)
SHVAL (2018)

- Trace execution and build data flow graph
- Keep both single and double precision versions
- Color nodes by error (single vs. double)
- Inherent scaling issues
  - Idea: aggregate!
Template Library (2016)

- Configurable floating-point types
- Template metaprogramming for efficiency
- Optional instrumentation to measure statistics

Credit: Scott Lloyd (lloyd23@llnl.gov)
Mixed Precision Tools

• Early precision auto-tuning projects
  – CRAFT [ICS’13]
    • Mike Lam, UMD (now JMU)
    • Instruction-centric search using Dyninst
    • github.com/crafthpc/craft
  – Precimonious [SC’13]
    • Cindy Rubio-Gonzáles, UC Berkeley (now UC Davis)
    • Variable-centric search using LLVM
    • github.com/corvette-berkeley/precimonious

• Influx of new projects 2017-present
  – FPTuner, Daisy, ADaPT
CRAFT (2013)

Mixed-Precision Analysis System

Original Program

Configuration Generator

Mixed-Precision Configurations

Binary Modification

Basic Block Patching

In-place Instruction & Operand Conversion

Mixed-Precision Programs

Configuration Evaluation

Recommended Configuration

Data Set

Verification Routine

Hierarchical Heuristics
CRAFT (2013)
CRAFT (2013)

<table>
<thead>
<tr>
<th>NAS Benchmark (name.CLASS)</th>
<th>Candidate Instructions</th>
<th>Configurations Tested</th>
<th>% Dynamic Replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt.A</td>
<td>6,262</td>
<td>4,000</td>
<td>78.6</td>
</tr>
<tr>
<td>cg.A</td>
<td>956</td>
<td>255</td>
<td>5.6</td>
</tr>
<tr>
<td>ep.A</td>
<td>423</td>
<td>114</td>
<td>45.5</td>
</tr>
<tr>
<td>ft.A</td>
<td>426</td>
<td>74</td>
<td>0.2</td>
</tr>
<tr>
<td>lu.A</td>
<td>6,014</td>
<td>3,057</td>
<td>57.4</td>
</tr>
<tr>
<td>mg.A</td>
<td>1,393</td>
<td>437</td>
<td>36.6</td>
</tr>
<tr>
<td>sp.A</td>
<td>4,507</td>
<td>4,920</td>
<td>30.5</td>
</tr>
</tbody>
</table>

• Issue: does not translate well to a mixed-precision implementation  
  • (difficult to go from instruction-level to source-level transformations)  
• Issue: each instruction is tested in isolation  
  • (combinations of replacements are often not passing)  
• Issue: cannot ensure speedup  
  • (instrumentation + casting overhead + lack of vectorization)
CRAFT (2013)

• Memory-based analysis
  – Replacement candidates: output operands
  – Analysis found several valid variable-level replacements
  – Does not consider computation (only storage)

<table>
<thead>
<tr>
<th>NAS Benchmark (name.CLASS)</th>
<th>Candidate Operands</th>
<th>Configurations Tested</th>
<th>% Executions Replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt.A</td>
<td>2,342</td>
<td>300</td>
<td>97.0</td>
</tr>
<tr>
<td>cg.A</td>
<td>287</td>
<td>68</td>
<td>71.3</td>
</tr>
<tr>
<td>ep.A</td>
<td>236</td>
<td>59</td>
<td>37.9</td>
</tr>
<tr>
<td>ft.A</td>
<td>466</td>
<td>108</td>
<td>46.2</td>
</tr>
<tr>
<td>lu.A</td>
<td>1,742</td>
<td>104</td>
<td>99.9</td>
</tr>
<tr>
<td>mg.A</td>
<td>597</td>
<td>153</td>
<td>83.4</td>
</tr>
<tr>
<td>sp.A</td>
<td>1,525</td>
<td>1,094</td>
<td>88.9</td>
</tr>
</tbody>
</table>
CRAFT (2018)

• Basic handling of variable-level source tuning
  – Initial approach: look for and modify double-precision declarations in source code using pattern matching
  – Current work: use TypeForge (a Rose tool) for AST-level modifications

• Search for best speedup
  – Output is rewritten version of the fastest configuration that passed verification

```plaintext
double x[ ];
double y[ ];
double dx[ ];
double dy[ ];
double alpha;
double beta;
double gamma;
```

```plaintext
double x[ ];
double y[ ];
float dx[ ];
float dy[ ];
double alpha;
double beta;
float gamma;
```

(non-replaceable)  (non-replaceable)  (replaceable)
(non-replaceable)  (replaceable)  (replaceable)
(non-replaceable)  (replaceable)  (replaceable)

Precimonious (2013)

- Dynamic LLVM IR variable tuning
- Delta-Debugging search algorithm

(a) Quadratic number of variants
(b) Binary search on quadratic search space
# Precimonious (2013)

## Original Type Configuration

<table>
<thead>
<tr>
<th>Program</th>
<th>L</th>
<th>D</th>
<th>F</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gaussian</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>roots</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>polyroots</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rootnewt</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sum</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fft</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blas</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EP</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>CG</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>arclength</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>simpsons</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

## Proposed Type Configuration

<table>
<thead>
<tr>
<th>Program</th>
<th>L</th>
<th>D</th>
<th>F</th>
<th>Calls</th>
<th># Config</th>
<th>mm:ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td>37:11</td>
</tr>
<tr>
<td>gaussian</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>201</td>
<td>16:12</td>
</tr>
<tr>
<td>roots</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1:03</td>
</tr>
<tr>
<td>polyroots</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>336</td>
<td>43:17</td>
</tr>
<tr>
<td>rootnewt</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>61</td>
<td>16:56</td>
</tr>
<tr>
<td>sum</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>325</td>
<td>28:14</td>
</tr>
<tr>
<td>fft</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1:16</td>
</tr>
<tr>
<td>blas</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1:06</td>
</tr>
<tr>
<td>EP</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>111</td>
<td>23:53</td>
</tr>
<tr>
<td>CG</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>3</td>
<td>44</td>
<td>0:57</td>
</tr>
<tr>
<td>arclength</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>33</td>
<td>0:40</td>
</tr>
<tr>
<td>simpsons</td>
<td>0</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>0:07</td>
</tr>
</tbody>
</table>

Credit: Cindy Rubio Gonzalez (crubio@ucdavis.edu)
Blame Analysis (2016)

• Shadow value data-flow analysis
• Runs program once maintaining “blame sets” to infer precision requirements
• Integrating blame analysis results causes Precimonious to converge significantly faster (9x on average)
• Lack of documentation

https://github.com/corvette-berkeley/shadow-execution
HiFPTuner (2018)

- Extension of Precimonious
- Builds network of variables using dependency analysis and edge profiling
- Hierarchical mixed precision search
- Claims better scaling
FPTuner (2017)

- Routine: Real-valued Expression
- Global Optimizer
- Error Model
- Efficiency Model
- Optimization Problem
- Optimal Mixed-precision
- Gurobi
- User Specifications
  - Input Ranges
  - Error Threshold
  - Extra Constraints

\[ \text{Python DSL} \]

Credit: Ganesh Gopalakrishnan (ganesh@cs.utah.edu)

https://github.com/soarlab/FPTuner
FP Tuner (2017)

Error threshold: 1e-3
(more permissive)

Error threshold: 1e-4

Error threshold: 1e-5
(more strict)
Daisy (2017)

- Input: real-valued expression DSL that is a subset of Scala
- Aggregates many previous analyses in a modular analysis system
  - Error bounding
  - Accuracy improvement
  - Mixed precision
- Online interface: http://daisy.mpi-sws.org
- “Limited support for conditionals or loops”
ADaPT (2018)

New tool developed at LLNL over the past year

Use first order Taylor series approximation to estimate the rounding errors in variables

$$\Delta y = f'(a) \Delta x$$ for $y=f(x)$ at $x=a$

Obtain $f'(a)$ using reverse-mode algorithmic differentiation (AD)

Credit: Harshitha Menon (gopalakrishn1@llnl.gov)
Original C Code

```c
#include <iostream>

double sum = 0.0;
double inc = 0.1;

double do_sum() {
    int i;
    for (i = 0; i < 1000; i++) {
        sum += inc;
    }
    return sum;
}

int main() {
    do_sum();
    cout << sum << endl;

    return 0;
}
```

AD Instrumented Code

```c
#include <iostream>
#include <adapt.h>
#include <adapt-impl.cpp>

AD_real sum = 0.0;
AD_real inc = 0.1;

AD_real do_sum() {
    int i;
    for (i = 0; i < 1000; i++) {
        sum += inc;
    }
    return sum;
}

int main() {
    AD_begin();
    AD_independent(inc, "inc");
    do_sum();
    cout << AD_value(sum) << endl;

    AD_dependent(sum, "sum", 8);
    AD_report();
    return 0;
}
```

- AD Libraries
- Type Changes
- Initialization
- Output
ADaPT (2018)

Used ADaPT on LULESH to create mixed precision sensitivity profile

Used the profile as a guide to develop a mixed precision version for a CUDA implementation of LULESH

Achieved speedup of 20% within error threshold of $1e^{-11}$ on GPU

Credit: Harshitha Menon (gopalakrishn1@llnl.gov)
CRAFT+ADaPT+TypeForge

• Search process and integration via CRAFT
• Source-to-source translation via TypeForge
• Optional inclusion of ADaPT analysis to narrow search
Summary

- 10+ actively-maintained floating-point analysis projects
- Diverse goals and approaches
- Many avenues for future research
Thank you!

Acknowledgements:

- Jeff Hollingsworth
- Bronis de Supinski
- Barry Rountree
- Jeff Hittinger

- Scott Lloyd
- Harshitha Menon
- Markus Schordan
- Shelby Funk

- Ramy Medhat
- Garrett Folks
- Logan Moody
- Nkeng Atabong

For more information: tinyurl.com/fpanalys

Contact me: lam2mo@jmu.edu