ANALYSIS AND SYNTHESIS OF FLOATING-POINT Routines

Zvonimir Rakamarić
FLOATING-POINT COMPUTATIONS ARE UBIQUITOUS
CHALLENGES

- FP is “weird”
  - Does not faithfully match math (finite precision)
  - Non-associative
  - Heterogeneous hardware support

- FP code is hard to get right
  - Lack of good understanding
  - Lack of good and extensive tool support

- FP software is large and complex
  - High-performance computing (HPC) simulations
  - Machine learning
FP IS WEIRD

- Finite precision and rounding
  - $x + y$ in reals $\neq x + y$ in floating-point

- Non-associative
  - $(x + y) + z \neq x + (y + z)$

- Creates issues with
  - Compiler optimizations (e.g., vectorization)
  - Concurrency (e.g., reductions)

- Standard completely specifies only $+$, $-$, $\ast$, $/$, comparison, remainder, and square root

- Only recommendation for some functions (trigonometry)
FP IS WEIRD cont.

- Heterogeneous hardware support
  - \( x + y \times z \) on Xeon \( \neq \) \( x + y \times z \) on Xeon Phi
    - Fused multiply-add
  - Intel’s online article “Differences in Floating-Point Arithmetic Between Intel Xeon Processors and the Intel Xeon Phi Coprocessor”

- Common sense does not (always) work
  - \( x \) “is better than” \( \log(e^x) \)
  - \( (e^x-1)/x \) “can be worse than” \( (e^x-1)/\log(e^x) \)
    - Error cancellation
FLOATING-POINT NUMBERS

- IEEE 754 standard
- Sign (s), mantissa (m), exponent (exp):
  
  \((-1)^s \times 1.m \times 2^{\text{exp}}\)

- Single precision: 1, 23, 8 bits
- Double precision: 1, 52, 11 bits
3 bits for precision

- Between any two powers of 2, there are $2^3 = 8$ representable numbers
ROUNDING IS SOURCE OF ERRORS

\[(x - x) (\tilde{y} - y)\]
FLOATING-POINT OPERATIONS

- First normalize to the same exponent
  - Smaller exponent -> shift mantissa right
- Then perform the operation
- Losing bits when exponents are not the same!
UTAH FLOATING-POINT TEAM

1. Ganesh Gopalakrishnan (prof)
2. Zvonimir Rakamarić (prof)
3. Ian Briggs (staff programmer)
4. Mark Baranowski (PhD)
5. Rocco Salvia (PhD)
6. Shaobo He (PhD)
7. Thanhson Nguyen (PhD)

Alumni: Alexey Solovyev (postdoc), Wei-Fan Chiang (PhD), Dietrich Geisler (undergrad), Liam Machado (undergrad)
RESEARCH THRUSTS

Analysis

- Verification of floating-point programs
- Estimation of floating-point errors
  1. Dynamic
     - Best effort, produces lower bound (under-approximation)
  2. Static
     - Rigorous, produces upper bound (over-approximation)

Synthesis

- Rigorous mixed-precision tuning

Constraint Solving

- Search-based solving of floating-point constraints
- Solving mixed real and floating-point constraints
RESEARCH THRUSTS

Analysis

- Verification of floating-point programs
- Estimation of floating-point errors
  1. Dynamic
     - Best effort, produces lower bound (under-approximation)
  2. Static
     - Rigorous, produces upper bound (over-approximation)

Synthesis

- Rigorous mixed-precision tuning

Constraint Solving

- Search-based solving of floating-point constraints
- Solving mixed real and floating-point constraints
ERROR ANALYSIS
Absolute error: $|z_{fp} - z_{inf}|$
Relative error: $|(z_{fp} - z_{inf}) / z_{inf}|$
ERROR PLOT FOR MULTIPLICATION

Absolute Error

X values

Y values

X axis

Y axis
ERROR PLOT FOR ADDITION

- Absolute Error
- X values
- Y values
**USAGE SCENARIOS**

- Reason about floating-point computations
- Precisely characterize floating-point behavior of libraries
- Support performance-precision tuning and synthesis
- Help decide where error-compensation is needed
- “Equivalence” checking
STATIC ANALYSIS

http://github.com/soarlab/FPTaylor
CONTRIBUTIONS

- Handles non-linear and transcendental functions
- Tight error upper bounds
  - Better than previous work
- Rigorous
  - Over-approximation
  - Based on our own rigorous global optimizer
  - Emits a HOL-Lite proof certificate
    - Verification of the certificate guarantees estimate
- Tool called FPTaylor publicly available
FPTaylor TOOLFLOW

1. Given FP Expression and Input Intervals
2. Obtain Symbolic Taylor Form
3. Obtain Error Function
4. Maximize the Error Function
   - Generate Certificate in HOL-Lite
Consider $op(x, y)$ where $x$ and $y$ are floating-point values, and $op$ is a function from floats to reals.

IEEE round-off errors are specified as

$$op(x, y) \cdot (1 + e_{op}) + d_{op}$$

For normal values

For subnormal values

Only one of $e_{op}$ or $d_{op}$ is non-zero:

- $|e_{op}| \leq 2^{-24}$, $|d_{op}| \leq 2^{-150}$ (single precision)
- $|e_{op}| \leq 2^{-53}$, $|d_{op}| \leq 2^{-1075}$ (double precision)
ERROR ESTIMATION EXAMPLE

- Model floating-point computation of \( E = \frac{x}{x + y} \) using reals as

\[
\hat{E} = \frac{x}{(x + y) \cdot (1 + e_1)} \cdot (1 + e_2)
\]

\(|e_1| \leq \epsilon_1, \ |e_2| \leq \epsilon_2\)

- Absolute rounding error is then \(|\hat{E} - E|\)

- We have to find the max of this function over
  - Input variables \( x, y \)
    - Exponential in the number of inputs
  - Additional variables \( e_1, e_2 \) for operators
    - Exponential in floating-point routine size!
SYMBOLIC TAYLOR EXPANSION

- Reduces dimensionality of the optimization problem

Basic idea
- Treat each $e$ as “noise” (error) variables
- Now expand based on Taylor’s theorem
  - Coefficients are symbolic
  - Coefficients weigh the “noise” correctly and are correlated

- Apply global optimization on reduced problem
  - Our own parallel rigorous global optimizer called Gelpia
  - Non-linear reals, transcendental functions
ERROR ESTIMATION EXAMPLE

\[ \tilde{E} = \frac{x}{(x + y) \cdot (1 + e_1)} \cdot (1 + e_2) \]

expands into

\[ \tilde{E} = E + \frac{\partial \tilde{E}}{\partial e_1}(0) \times e_1 + \frac{\partial \tilde{E}}{\partial e_2}(0) \times e_2 + M_2 \]

where \( M_2 \) summarizes the second and higher order error terms and \( |e_0| \leq \varepsilon_0, |e_1| \leq \varepsilon_1 \)

Floating-point error is then bounded by

\[ |\tilde{E} - E| \leq \left| \frac{\partial \tilde{E}}{\partial e_1}(0) \right| \times \varepsilon_1 + \left| \frac{\partial \tilde{E}}{\partial e_2}(0) \right| \times \varepsilon_2 + M_2 \]
Using global optimization find constant bounds

\( M_2 \) can be easily over-approximated

Greatly reduced problem dimensionality

Search only over inputs \( x, y \) using our Gelpia optimizer

\[
\forall x, y. \quad \left| \frac{\partial \tilde{E}}{\partial e_1}(0) \right| = \left| \frac{x}{x+y} \right| \leq U_1
\]

\[
|\tilde{E} - E| \leq \left| \frac{\partial \tilde{E}}{\partial e_1}(0) \right| \times \epsilon_1 + \left| \frac{\partial \tilde{E}}{\partial e_2}(0) \right| \times \epsilon_2 + M_2
\]
ERROR ESTIMATION EXAMPLE

Operations are single-precision (32 bits)

\[ |\tilde{E} - E| \leq U_1 \times \epsilon_{32\text{-}bit} + U_2 \times \epsilon_{32\text{-}bit} \]

Operations are double-precision (64 bits)

\[ |\tilde{E} - E| \leq U_1 \times \epsilon_{64\text{-}bit} + U_2 \times \epsilon_{64\text{-}bit} \]
RESULTS FOR JETENGINE

ejtEngine, $x_1 \in [-5, 5], x_2 \in [-20, 5]$, Double Precision
SUMMARY

- New method for rigorous floating-point round-off error estimation
- Our method is embodied in new tool FPTaylor
- FPTaylor performs well and returns tighter bounds than previous approaches
SYNTHESIS

http://github.com/soarlab/FPTuner
MIXED-PRECISION TUNING

Goal:
Given a real-valued expression and output error bound, automatically synthesize precision allocation for operations and variables
APPROACH

- Replace machine epsilons with symbolic variables
  \[ s_0, s_1 \in \{ \varepsilon_{32\text{-bit}}, \varepsilon_{64\text{-bit}} \} \]

\[ |\tilde{E} - E| \leq U_1 \times s_1 + U_2 \times s_2 \]

- Compute precision allocation that satisfies given error bound
  - Take care of type casts

- Implemented in FPTuner tool
FP Tuner TOOLFLOW

Routine: Real-valued Expression

Generic Error Model
Efficiency Model

Optimization Problem

Optimal Mixed-precision

Gelpia Global Optimizer

Gurobi

User Specifications
Error Threshold
Operator Weights
Extra Constraints
EXAMPLE: JACOBI METHOD

- Inputs:
  - 2x2 matrix
  - Vector of size 2
- Error bound: 1e-14
- Available precisions: single, double, quad
- FPTuner automatically allocates precisions for all variables and operations
SUMMARY

- Support mixed-precision allocation
- Based on rigorous formal reasoning
- Encoded as an optimization problem
- Extensive empirical evaluation
  - Includes real-world energy measurements showing benefits of precision tuning
SOLVING

http://github.com/soarlab/OL1V3R
MOTIVATION

- Poor scalability of floating-point solvers
  - Bit-blasting: formula $\rightarrow$ circuit

- Others showed that search-based solving can be effective for various SMT theories
  - Perform the search directly on theory level

- Can we achieve similar efficiency using stochastic local search on floating-points?
  - Inspired by Z3’s qfbv-sls tactic for bit-vectors
STOCHASTIC LOCAL SEARCH

- Basic setting: local search + random choices

- Key ingredients
  - Score function
  - Neighborhood relation
  - Heuristics
SCORE FUNCTION

- score(expr, assignment) \rightarrow \text{rational}

- Intuition: the ``degree'' of satisfiability
  - 1 = satisfiable
  - Example: s(x>2, x←1.99) > s(x>2, x←0)

- Key idea: measure a distance between signed ordinal indices of two floats
  - Total order on floats
  - Neighboring floats have a distance of 1
NEIGHBORHOOD RELATION

- Define neighbors of an assignment in a search step
- Several allowed mutations
  - Bit-flipping
  - ±ulp
  - (*2), (/2) – changing exponent
HEURISTICS

- Remove equality constraints when possible
  - (assert (and (= x (+ y z)) (> x 2.0)))
    → (assert (> (+ y z) 2.0))

- Use models derived from real arithmetic as initial assignments
  - (assert (> (+ y z) 2.0)) → y = 1, z = 3/2

- Variable neighborhood search
  - Refine the neighborhood relation into 3 subgroups and switch them on the fly
EVALUATION

- Compare OL1V3R with 5 state-of-the-art floating-point solvers

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>MathSAT</td>
<td>5.5.4</td>
<td>Hybrid</td>
</tr>
<tr>
<td>CVC4</td>
<td>1.7</td>
<td>Bit-blasting</td>
</tr>
<tr>
<td>Z3</td>
<td>4.8.4</td>
<td>Bit-blasting</td>
</tr>
<tr>
<td>JFS</td>
<td>commit 2322167</td>
<td>Coverage-guided fuzzing</td>
</tr>
<tr>
<td>COLIBRI</td>
<td>revision 2176</td>
<td>Constraint propagation</td>
</tr>
</tbody>
</table>
## RESULTS

<table>
<thead>
<tr>
<th>Tool</th>
<th>Sat</th>
<th>Unsat</th>
<th>Unknown</th>
<th>Timeout</th>
<th>Diff\textsuperscript{B}</th>
<th>Diff\textsuperscript{H}</th>
</tr>
</thead>
<tbody>
<tr>
<td>OL1V3RB</td>
<td>115</td>
<td>0</td>
<td>2</td>
<td>80</td>
<td>-</td>
<td>0/16</td>
</tr>
<tr>
<td>OL1V3RH</td>
<td>131</td>
<td>0</td>
<td>2</td>
<td>64</td>
<td>16/0</td>
<td>-</td>
</tr>
<tr>
<td>MathSAT</td>
<td>125</td>
<td>1</td>
<td>7</td>
<td>64</td>
<td>13/5</td>
<td>2/9</td>
</tr>
<tr>
<td>CVC4</td>
<td>117</td>
<td>1</td>
<td>10</td>
<td>69</td>
<td>10/9</td>
<td>2/15</td>
</tr>
<tr>
<td>Z3</td>
<td>88</td>
<td>0</td>
<td>10</td>
<td>99</td>
<td>3/32</td>
<td>0/43</td>
</tr>
<tr>
<td>JFS</td>
<td>113</td>
<td>0</td>
<td>0</td>
<td>84</td>
<td>4/8</td>
<td>0/20</td>
</tr>
<tr>
<td>COLIBRI</td>
<td>118</td>
<td>32</td>
<td>4</td>
<td>43</td>
<td>14/13</td>
<td>3/18</td>
</tr>
</tbody>
</table>
SUMMARY

- Implemented a prototype for solving floating-point constraints using SLS
  - Define key ingredients (score function, neighbors)
  - Devise custom heuristics

- Compared our tool to state-of-the-art solvers and confirmed its effectiveness