Using Dyninst to Measure Floating-point Error

Mike Lam,
Jeff Hollingsworth and Pete Stewart

Dyn inst

Floating-Point Overview

- Floating-point representation components
  - Sign (+/-)
  - Significand/mantissa (s)
  - Exponent (e)
  - Base (b) - implicit, usually 2
- Observations
  - Actual value is $\pm s \cdot b^e$
  - Minimum positive value is called machine ε
  - Precision = # of bits in significand
  - Error = |TrueValue - StoredValue|

Problems

- Quantization error
  - Introduced at initialization
- Roundoff error
  - Accumulates through operations
- Cancellation error
  - Occurs under specific conditions

Previous Solutions

- Reduce error by increasing precision
  - Double-precision (64 bits)
  - Quad-precision (128 bits)
  - Arbitrary-precision libraries
- Track error
  - Manually insert shadow value analysis in source code
  - Static analysis to bound error mathematically

Motivation

- Goal
  - Measure floating-point error automatically
- Benefits
  - Less human effort
  - Less error-prone
  - Better performance by selectively using lower precision where error is at acceptable levels

Our Solution

- Binary instruction instrumentation
  - No intermediate representation (like w/ Valgrind)
  - We can ignore compiler optimizations
  - No source code required (like w/ static analysis)
  - No special hardware required (like w/ PAPI)
- Dyninst API
  - Minimal tool code (<1500 LOC)
  - Support for static binary rewriting

Our Solution

- Two parts
  - Dyninst mutator to insert instrumentation
    Mutator(Mutatee) => Mutant
  - Runtime library with analysis routines
- Instrument all floating point instructions
  - In x87, all d8, d9, dc, dd, and de opcodes
  - Insert calls to profiler library with:
    - Raw instruction bytes
    - Register values (EAX-EDX,ESP,EBP)

Quantization Error

- Overview
  - We can't store a true value directly
  - Error will be less than machine ε
- Current status
  - Exact solution requires *a priori* knowledge
  - Currently have to ignore or bound
    - Upper limit is a function of machine ε

Roundoff Error

- Overview
  - Error grows with calculations
  - Example: (A is true value, a is stored value)

$$a-\varepsilon \le A \le a+\varepsilon$$
$$b-\varepsilon \le B \le b+\varepsilon$$

Error: ε

$$(a-\varepsilon)+(b-\varepsilon) \le A+B \le (a+\varepsilon)+(b+\varepsilon)$$
$$(a+b)-2\varepsilon \le A+B \le (a+b)+2\varepsilon$$

Error: 2ε

Roundoff Error

- Sparse shadow value table
  - Maps memory addresses to errors
  - Need to maintain error through FP registers
  - First eight addresses correspond to registers ST0 through ST7
- Instrument main()
  - At entry point, initialize shadow value table
  - At exit point, print shadow value table

Roundoff Error

- For each FP instruction:
  - Use XED to extract info about each operand:
    - Location (register number or memory address)
    - Current value
    - Read/write/both
  - Read current error from shadow value table
  - Calculate new error
  - Save error back to shadow value table

Roundoff Error

- Current status
  - Works for rough upper bounds, but is handicapped because of quantization error
  - Unclear which formulas are best
  - Problem with moves through integer registers

Cancellation Error

- Overview
  - Loss of significance during operations
  - Example: 1.0 - 1.0 = 4.44089e-16
  - "Catastrophic" loss of significance
- Current status
  - Detect this by examining floating-point operands

Cancellation Error

- Message queue
  - Contains list of cancellation error events
- Instrument main()
  - At entry point, initialize message queue
  - At exit point, print messages to standard output

Cancellation Error

- For each FP instruction:
  - Use XED to extract value of each operand
  - Calculate resulting value and compare magnitudes (exponents)
    - If $e_{ans} < \max(e_x, e_y)$ there is a cancellation
- For each cancellation event:
  - Record a "priority:" $\max(e_x, e_y) - e_{ans}$
  - Save event information to message queue

Results

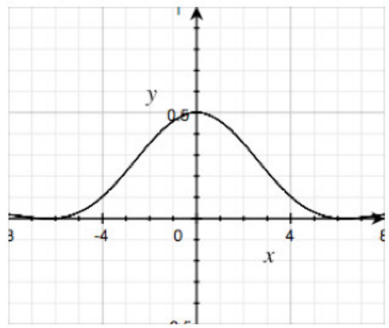
- archPI.c
  - Approximates π using two different series and compares results against the known value
  - Series #1
    - 2 cancellations of 51 binary digits each
    - Series diverges to NaN
  - Series #2
    - No cancellations
    - Final series result is 3.14159265358979578
    - Actual value of π is 3.14159265358979323

Results

- exponential.c
  - Approximates e^x using the Taylor series and checks against built-in exp(x) function
  - For e^{50}
    - No cancellations
    - Taylor result: 5.18470552858707624e+21
    - Builtin result: 5.18470552858707205e+21
  - For e^{-50}
    - 55 cancellations (worst priority: 57)
    - Taylor result: -5.6676e+04
    - Builtin result: 1.9287e-22

Results

- catastrophic.c
  - Approximates (1 - cos x) / x^2, which is very close to 0.5 in the interval [-4e-8, 4e-8]
  - x = -8e-7
    - 6 cancellations (worst priority: 42)
    - Result: 4.999473e-01
  - x = -8e-9
    - 5 cancellations (worst priority: 29)
    - Result: 0.000000e+00
  - So cancellations are not necessarily indicative of a real problem



$$y = \frac{(1 - \cos x)}{x^2}$$

Future Work

- Known issues
  - Address quantization error
  - Shared library handling (ex. libm.so routines)
  - Roundoff analysis “loses” errors
- Improve reporting
  - Filter events to find true problems
  - Stack traces for events
  - Aggregate error by instruction or variable
  - Visualization or IDE integration

Future Work

- Use InstructionAPI instead of XED
- Add program slicing to reduce amount of instrumentation
- Profile overhead and optimize
- Expand to more platforms and instruction sets

Conclusion

- We are using Dyninst to automatically insert floating-point error measurement
  - Detection of cancellation events
  - Tracking for roundoff error
- Preliminary results are promising
- Many areas for future work

Thank you!