Detecting Floating-Point Errors via Atomic Conditions

Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, Lu Zhang, Zhendong Su

POPL 2020

New Orleans, Louisiana, United States

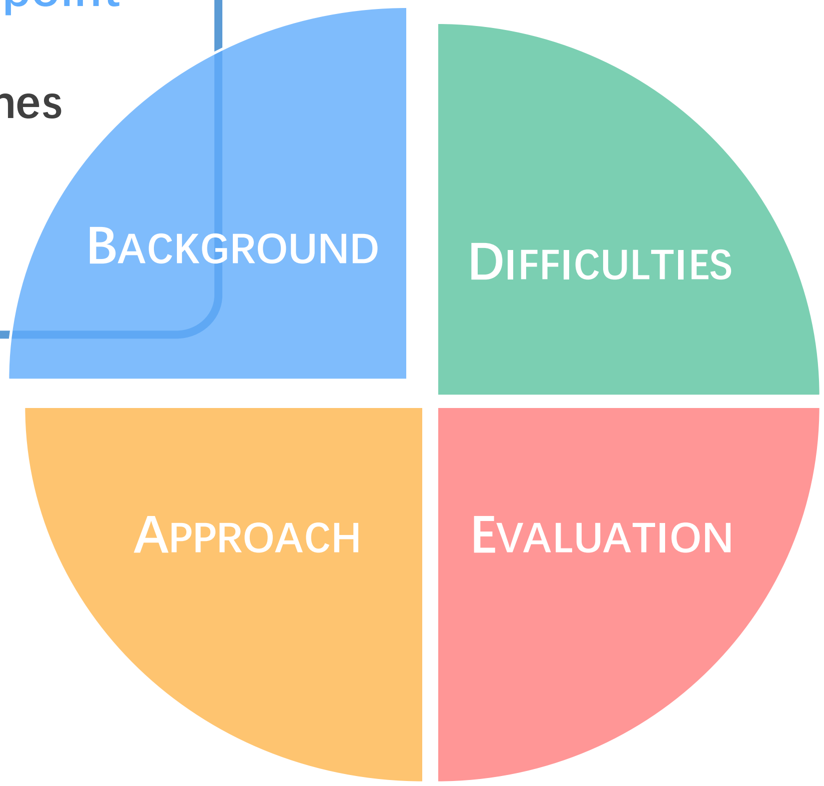
Analyzing Floating-Point Errors in a Flash

- DEMC [POPL 19]: ~8 hours
  - Analyzing 49 functions from GNU Scientific Library

- Our tool ATOMU: ~21 seconds
  - 1000+x faster
  - 40% more detected FP errors

Outline

- What is floating-point error?
- Existing approaches



Floating-Point Errors

- Some inputs may trigger significant FP errors
- Considering:

$$f(x) = \frac{1-\cos(x)}{x^2} \qquad \lim_{x\to 0} f(x) = 0.5$$

```
def f(x):
  num = 1-math.cos(x)
  den = x*x
  return num/den
```

// Using double precision (64 bits)

```
>>> f(1e-7)
0.4996003610813205
```

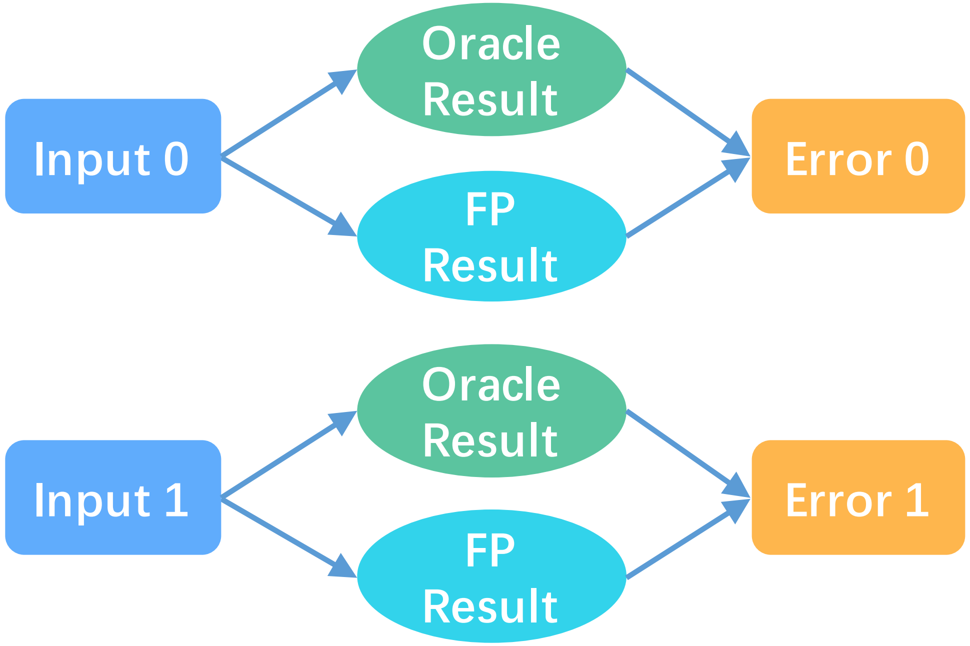
Accurate result (Oracle):

0.49999999999999583

Detecting Floating-Point Errors

- Given: A FP program $\hat{f}$
- Goal: An input triggers significant FP errors

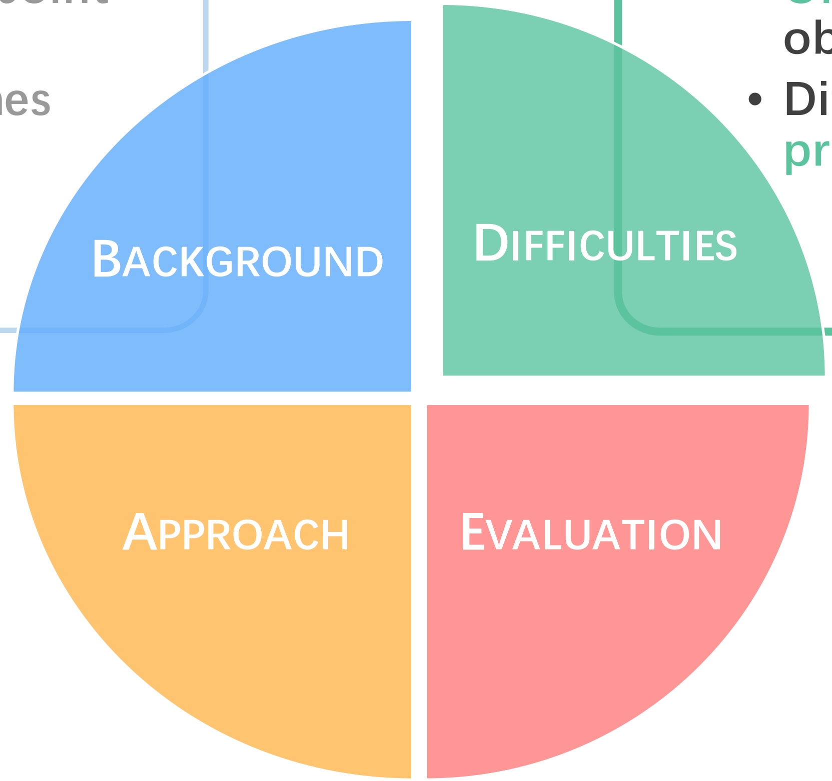
- Existing approaches:
  - Treat the FP program as a *black-box*
  - Heavily depend on the *oracle* f
- How to get the oracle f ?
  - Using *high precision program* $\hat{f}_{high}$ to simulate



Outline

- What is floating-point error?
- Existing approaches

- Oracles are hard to obtain
- Difficulties for high-precision types



The Expenses of Using $\hat{f}_{high}$

- $\hat{f}_{high}$ is expensive in *computation cost*
  - Even quadruple precision (128 bits) are 100x slower than double precision (64 bits)
  - For arbitrary precision (MPFR), the overhead further increases

- $\hat{f}_{high}$ is expensive in *development cost*. One cannot simply change all variables to high-precision types because of:
  - Precision-related operations
  - Precision-specific operations

The Expenses of Using $\hat{f}_{high}$

- **Precision-related operations**
  - Widely exist in numerical libraries

- Example: calculating sin(x) for x near 0 based on *Taylor series* at x=0:

$$\sin(x) = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + O(x^8)$$

- Accurate results need:
  - Higher precision types
  - Manually add more terms

```
double sin(double x) {
  if (x > -0.01 && x < 0.01) {
    double y = x*x;
    double c1 = -1.0 / 6.0;
    double c2 = 1.0 / 120.0;
    double c3 = -1.0 / 5040.0;
    double sum =
     x*(1.0 + y*(c1 + y*(c2 + y*c3)));
    return sum;
  }
else { ... } }
```

The Expenses of Using $\hat{f}_{high}$

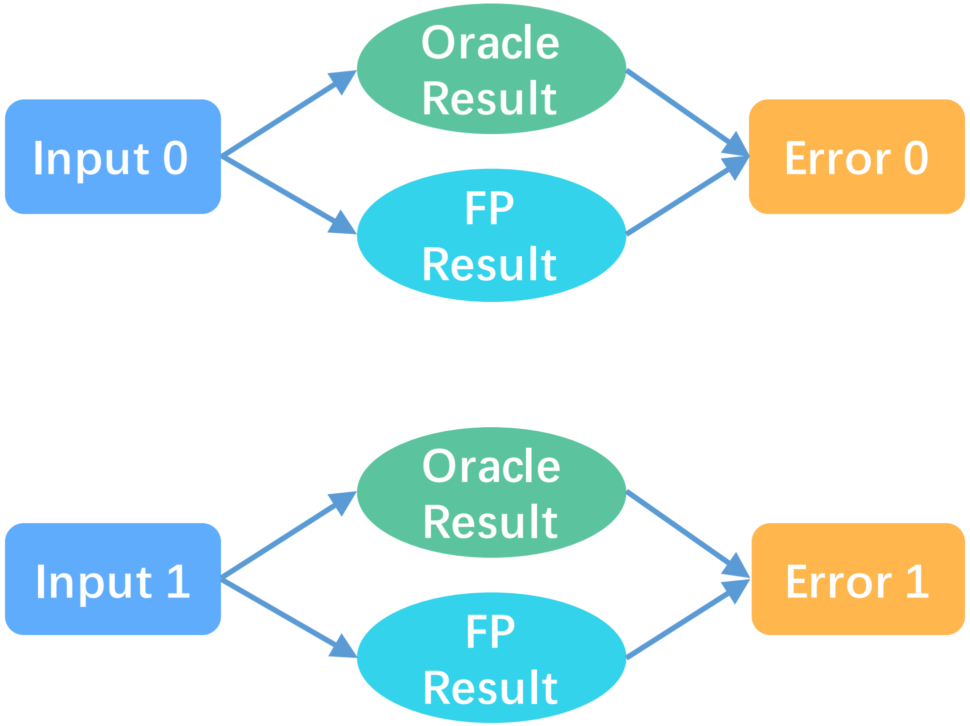
- **Precision-specific operations**
- A simplified example from GNU C Library:

```
double round(double x) {
  double n = 6755399441055744.0; // 3 << 51
  return (x + n) - n; }
```

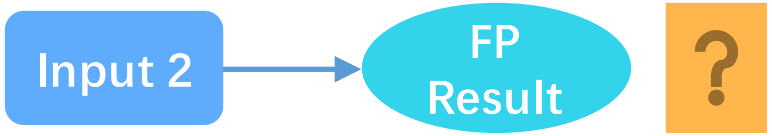
Magic number and only works on *double precision* (64 bits).

- Semantics: rounding x to nearest integer value
- Higher precision types will violate the semantics and lead to wrong results

Need for Oracle-Free Approach



- Existing approaches need oracle result to distinguish the inputs
- Oracles are hard to obtain
  - Development cost
  - Computation cost
- How to analyze FP programs without oracle?



Outline

Background

- What is floating-point error?
- Existing approaches

Difficulties

- Oracles are hard to obtain
- Difficulties for high-precision types

Approach

- Analysis based on Atomic Condition
- A novel detecting approach: Atomu

Evaluation

Analyzing the Floating-Point Error

- Atomic Operation
  - Elementary arithmetic: $+, -, \times, \div$.
  - Basic functions: sin, tan, exp, log, sqrt, pow, …
- Errors in atomic operations
  - Guaranteed to be small by IEEE-754 and GNU C Library reference

- Why does significant error still exist?

- Certain operations may amplify the FP errors

Analyzing the Floating-Point Error

- Condition Numbers
  - Measures the inherent stability (sensitivity) of a mathematical function

$$Err_{rel}(f(x), f(x+\Delta x)) = Err_{rel}(x, x+\Delta x) \cdot \left|\frac{xf'(x)}{f(x)}\right|$$

  - The condition number $\Gamma_f(x) = \left|\frac{xf'(x)}{f(x)}\right|$ measures how much the relative error will be *amplified* from input to output.

  - Example: $\Gamma_{\cos}(x) = |x \cdot \tan(x)|$

Key Insight

Atomic condition: condition numbers on atomic FP operations

- We can analyze FP programs by leveraging atomic condition
  - Errors amplified by atomic conditions
  - Atomic conditions are dominant factor for FP errors
- We can use native FP types for computing atomic conditions
  - Without high precision types
  - Accelerating the analysis

Motivation Example

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

$$\lim_{x \to 0} f(x) = 0.5$$

```
def f(x):
  v1 = cos(x)
  v2 = 1.0 - v1
  v3 = x * x
  v4 = v2 / v3
  return v4
```

Error Amplification by Atomic Condition when x = 1e-7

Input	Atomic condition	Output
1.0e-7	1e-14	9.9999999999999500**4**e-01
1.0 9.9999999999999500**4**e-01	2.0016e+14 2.0016e+14	4.99**600361081320443**e-15
1.0e-7 1.0e-7	1 1	9.999999999999998**41**e-15
4.99**600361081320443**e-15 9.999999999999998**41**e-15	1 1	4.99**600361081320499**e-01

Error Propagation and Atomic Condition

- Atomic Operation OP:
  - Error in input ε_x
  - Error in output ε_z
  - Atomic condition $\Gamma_{op}(x)$
  - Introduced error $\mu_{op}(x)$

$$\varepsilon_z = \varepsilon_x \underline{\Gamma_{op}(x)} + \underline{\mu_{op}(x)}$$

// Can be generalized to multivariate with partial derivatives

- The *introduced error* is guaranteed to be small. The *atomic condition* is the *dominant factor* of floating-point error.

Error Propagation and Atomic Condition

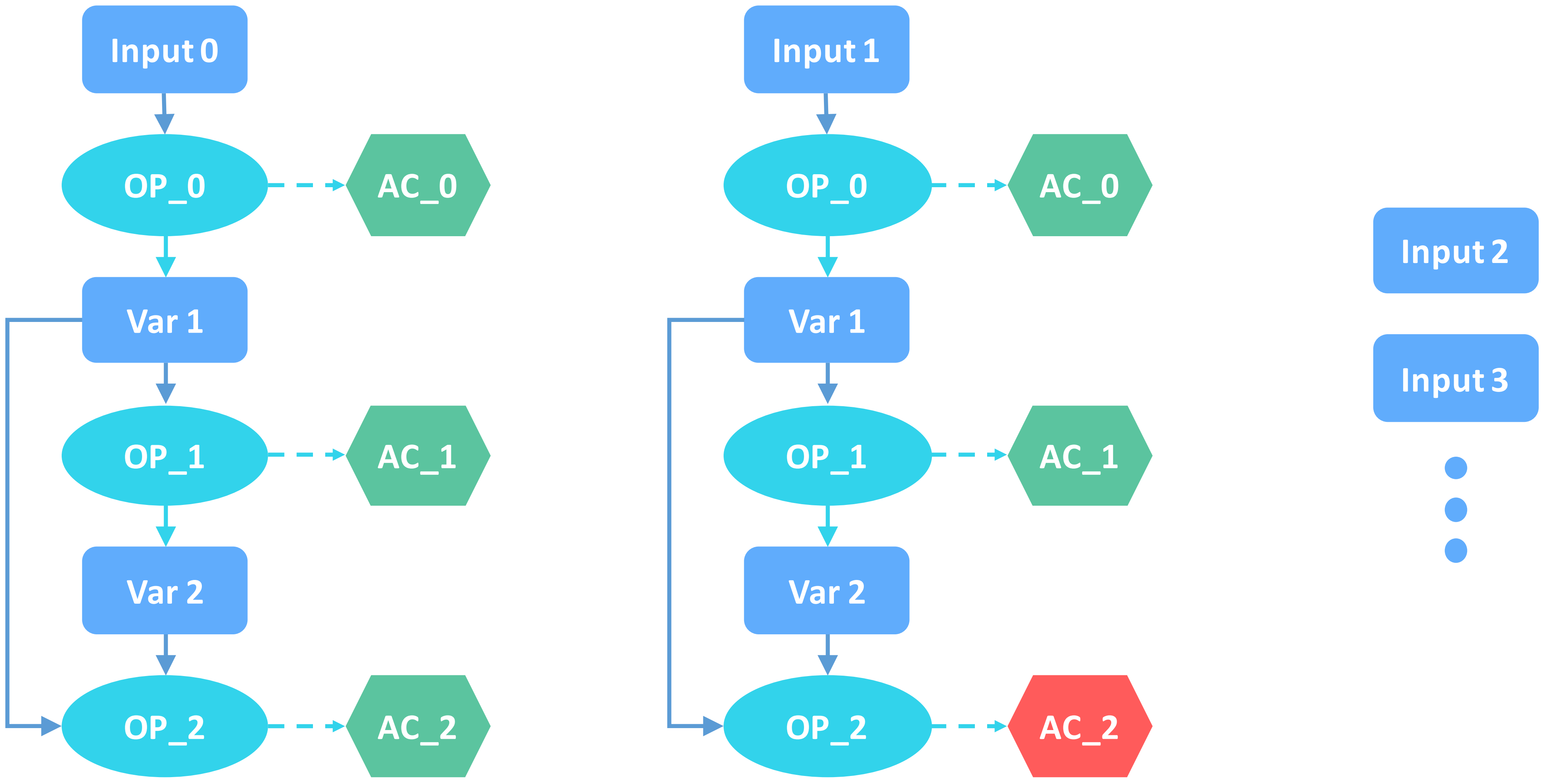
- Pre-calculated atomic condition formulae
- Potential unstable operations:
    - Atomic condition becomes significantly large ($\to \infty$) if its operand(s) falls into *danger zone*
- Stable operations:
    - Atomic condition always ≤ 1

Pre-calculated atomic condition formulae

$$\Gamma_f(x) = \left| \frac{x f'(x)}{f(x)} \right|$$

Operation	Atomic Condition	Danger Zone
$x + y$	$\left\lvert\frac{x}{x+y}\right\rvert, \left\lvert\frac{y}{x+y}\right\rvert$	$x \approx -y$
$\cos(x)$	$\lvert x * \tan(x)\rvert$	$x \to n\pi + \frac{\pi}{2}, n \in \mathbb{Z}$
$\log(x)$	$\left\lvert\frac{1}{\log(x)}\right\rvert$	$x \to 1$
…	…	…
$x * y$	$1, 1$	-
$\sqrt{x}$	0.5	-
…	…	…

Atomic Condition-Guided Search



Outline

Background

- What is floating-point error?
- Existing approaches

Difficulties

- Oracles are hard to obtain
- Difficulties for high-precision types

Approach

- Analysis based on Atomic Condition
- A novel detecting approach: ATOMU

Evaluation

- How effective?
- How fast?

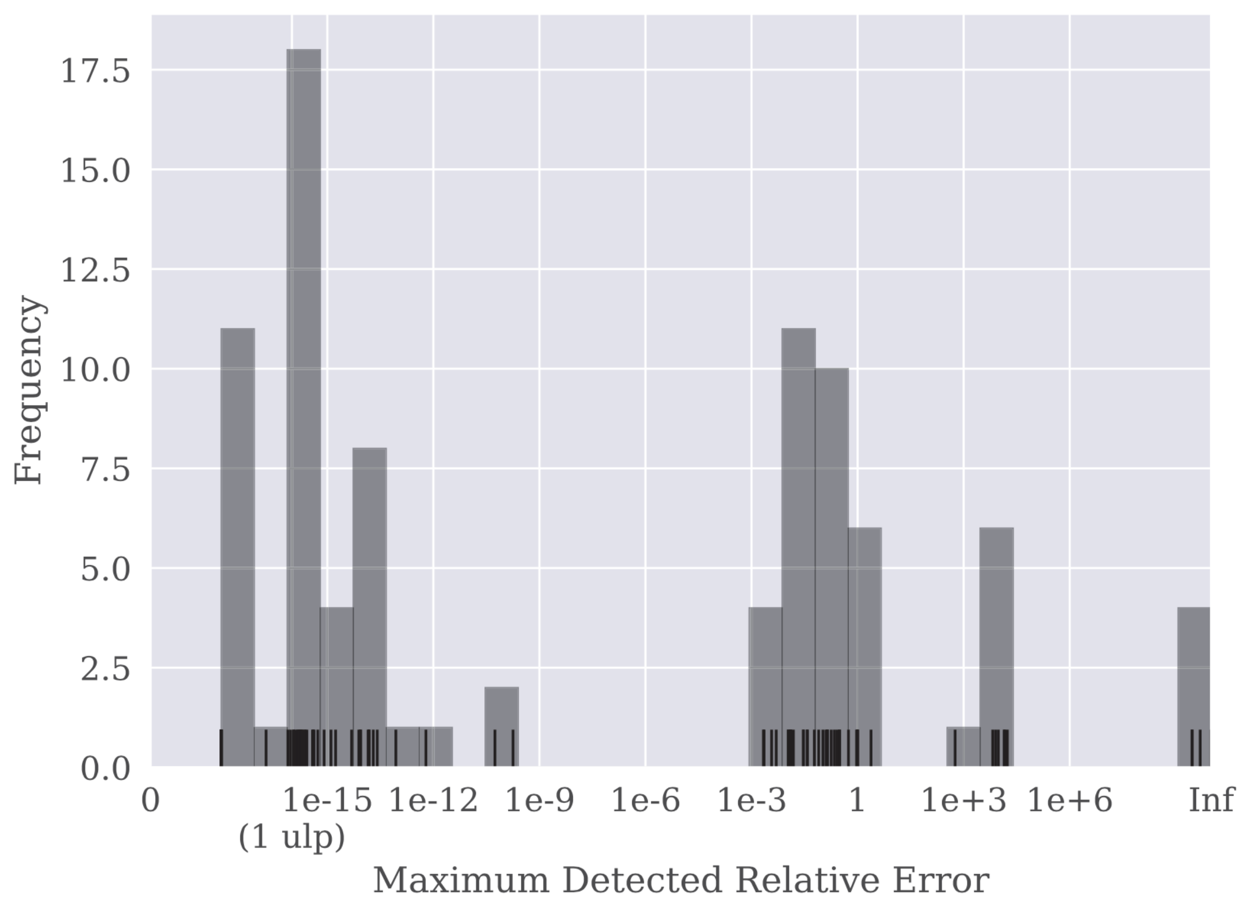
Evaluation

- Subjects: 88 functions from GNU Scientific Library
- Definition of significant error: relative error $\geq 10^{-3}$

On 88 GSL Functions	FP Operations	Potential Unstable Operations	Unstable Operations
#operations	90	40	12

Evaluation – Effectiveness

ATOMU finds significant errors in 42 of the 88 GSL functions



Evaluation – Effectiveness

- Compared with the state-of-the-art technique, ATOMU
  - Finds significant errors in 8 more functions (28 vs. 20)
  - Incurs no false negatives

```
gsl_sf_sin
gsl_sf_cos
gsl_sf_sinc
gsl_sf_dilog
gsl_sf_expint_E1
gsl_sf_expint_E2
gsl_sf_lngamma
gsl_sf_lambert_W0
```

Evaluation – Runtime Cost

- Avg. cost per GSL Function
  - ATOMU + oracle (validation): 0.34+0.09 seconds
  - 1000+x faster than DEMC [POPL 2019]
  - 100+x faster than LSGA [ICSE 2015]

- ATOMU achieves *orders of speedups* over the state-of-the-art
  - Much more practical

Take-Home Messages

- ATOMU: Super fast / effective technique for detecting FP errors
- Atomic condition: Powerful tool for analyzing FP programs
  - Oracle-free
  - Native
  - Informative
- Expected broader applications based on atomic condition
  - Debugging, Repair, Synthesis, etc.

https://github.com/FP-Analysis/atomic-condition