Mitigating Numerical Inconsistencies and Exceptions in Heterogeneous HPC Systems

Ignacio Laguna

ZIH-COLLOQUIUM
Technische Universität Dresden
May 26, 2023

LLNL-PRES-845366

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Numerical Reproducibility and Numerical Consistency Is Crucial

Code

System 1

x = 1.0001
y = 2.0001
z = 3.0001

→

Code

System 2

x = 1.0001
y = 2.0001
z = 3.0001

→

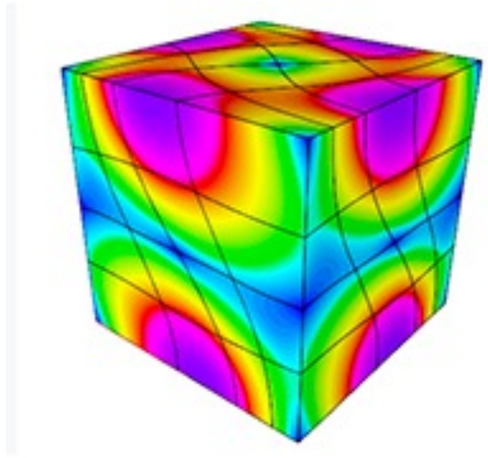
Code

System 3

x = 1.0001
y = 5.3746
z = ∞

Real example of a numerical inconsistency

Hydrodynamics mini application



<u>Early development and porting to new system (IBM Power8, NVIDIA GPUs)</u>

clang –O1: |e| = 129941.1064990107
clang –O2: |e| = 129941.1064990107
clang –O3: |e| = 129941.1064990107

gcc –O1: |e| = 129941.1064990107
gcc –O2: |e| = 129941.1064990107
gcc –O3: |e| = 129941.1064990107

xlc –O1: |e| = 129941.1064990107
xlc –O2: |e| = 129941.1064990107
xlc –O3: **|e| = 144174.9336610391**

It took several weeks of effort and many methods to debug it

Sources of Numerical Inconsistencies in Numerical Software

- Floating-point error — 1.23xxxxx…
- New Hardware (e.g., GPU)
- New Compiler — →01011
- Optimizations
- Exceptions
- …

Focus of the talk

Strategy to Mitigate Numerical Inconsistencies and Exceptions

1

Detecting **Numerical Exceptions**

- Exceptions cause inconsistencies
- Detection is crucial
- Compiler and dynamic instrumentation

2

Finding Inputs that Cause Exceptions

- Inputs induce exceptions
- Bayesian Optimization
- Mitigate bad inputs in testing

3

Isolating **Lines of Code**

- Isolate code that is impacted
- Search by enhancing precision
- Isolate expressions

State of The Art in FP Exception Detection

- When a CPU exceptions occurs, it is signaled
  - Status flag FPSCR (floating-point status and control register) is set by default
  - Tools can read such registers
    - Peter Dinda, Alex Bernat, and Conor Hetland. ***Spying on the Floating-Point Behavior of Existing, Unmodified Scientific Applications***. In Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (**HPDC**), 2020

NVIDIA GPUs have no mechanism to detect floating-point exceptions,
set a status register or raise a signal when an exception occurs

Printf Helps but It’s Not Enough

NaN and **Infinity** propagate quickly:

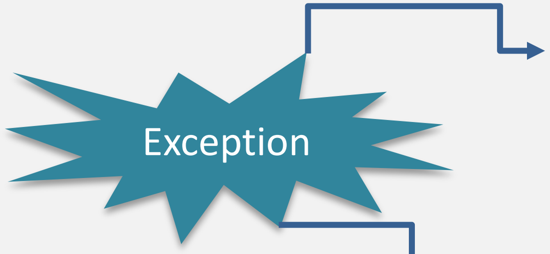
$2 \times \infty = \infty$

Code

```
printf(“Energy = %f\n”, energy);
```

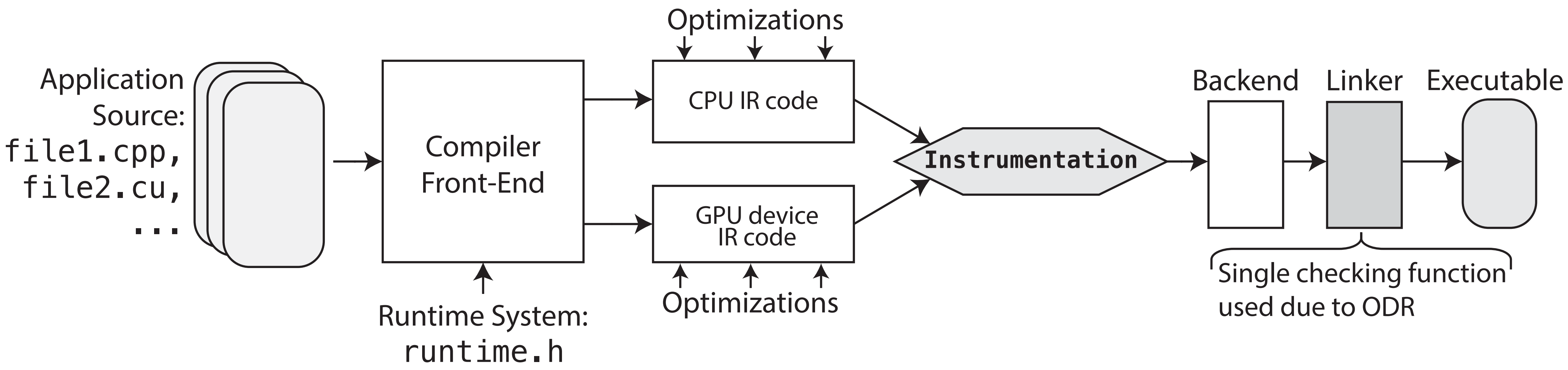
Output

```
Energy = Inf
```




```
double x = a/b;
if (x < …) {

  //Branch 1

} else {

  //Branch 2

}
```

Compile-time Instrumentation Workflow of FPChecker



FPChecker

https://fpchecker.org/

Laguna, Ignacio, et al. "FPChecker: Floating-Point Exception Detection Tool and Benchmark for Parallel and Distributed HPC." *2022 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2022.

Floating Point Exceptions and Events Detected by FPChecker

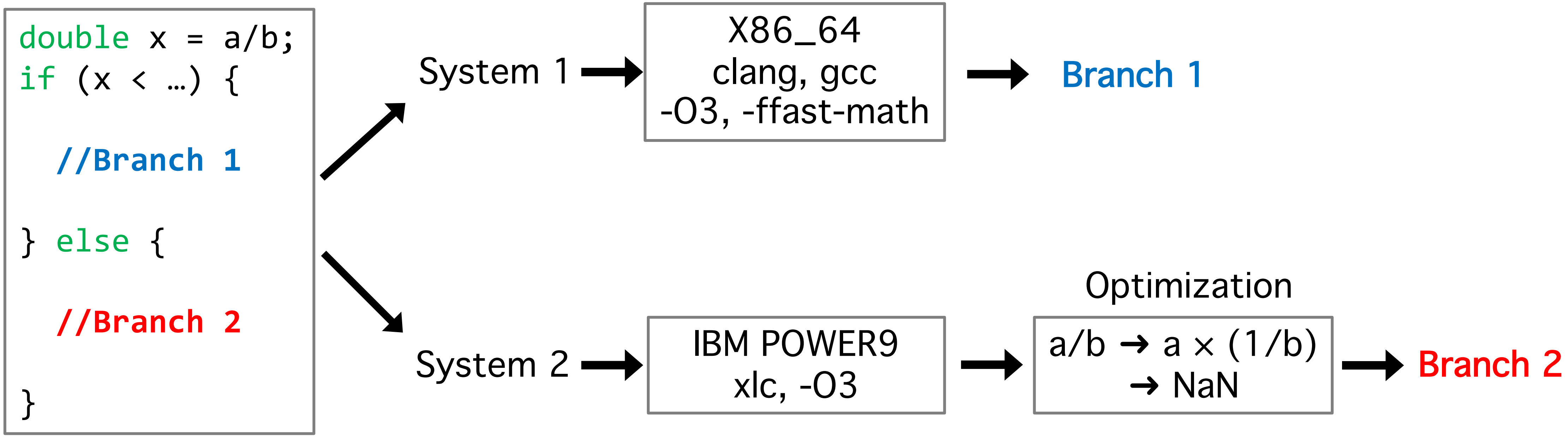
IEEE 754-2019 Standard

Exception	Description	Result	
`Overflow`	Result did not fit and it is an infinity	∞	✓
`Underflow`	Result could not be represented as normal	0, subnormal	✓
`Divide by Zero`	Divide-by-zero operation	∞	✓
`Invalid`	Operation operand is not a number (NaN)	NaN	✓
`Inexact`	Result is produced after rounding	normal	✗

Other events	Description	
`Comparison`	Two numbers are compared for equality	✓
`Cancellation`	Cancellation in addition or subtraction	✓
`Latent Infinity+`	Large normal, close to positive infinity	✓
`Latent Infinity-`	Large normal, close to negative infinity	✓
`Latent underflow`	Small normal, close to subnormal	✓

Example of Reproducibility Problem:
Subnormal Numbers and Optimizations

- **Subnormal numbers:** very small quantities, 1e-309



FPChecker Report

Main Report

	Events	Severity
Infinity (+)	0	High
Infinity (–)	0	High
NaN	860	High
Division by zero	0	High
Underflow (subnormal)	0	Medium
Comparison	0	Medium
Cancellation	478141486	Low
Latent Infinity (+)	0	Low
Latent Infinity (–)	0	Low
Latent underflow	0	Low

FPChecker Evaluation Results

Program	Model	∞+	∞−	NaN	DivByZero	Cancell.	Comp.	Underf.	Lat. ∞+	Lat. ∞−	Lat. Underf.
LULESH	MPI, OpenMP	0	0	430	0	123,068,684	0	0	0	0	0
Kripke	MPI, OpenMP, *RAJA*	0	0	0	0	256	4,096	0	0	0	0
Quiksilver	MPI	0	0	0	0	32,786,872	1,153,703,064	0	0	0	0
RAJAPerf	*RAJA*, serial	0	0	0	0	3,306,835,400	0	0	0	0	0
AMG	MPI	14	0	0	14	7,885	31,385	0	14	0	0
IS	serial	2	0	0	1	1,052	0	0	2	0	0
EP	serial	0	0	0	0	134,163	0	0	0	0	0
CG	serial	22,505	0	0	0	22,505	0	0	0	0	0
MG	serial	0	0	0	0	10,376	0	0	0	0	0
FT	serial	0	0	0	0	4,222	0	0	0	0	0
BT	serial	0	0	0	0	739	0	0	0	0	0
SP	serial	0	0	0	0	752	0	0	0	0	0
LU	serial	0	0	0	0	12	0	0	0	0	0

- Cancellation is a common event in HPC workloads
- FPChecker reported NaN in a few applications
- Several applications compare floating-point numbers for equality
  - This can be dangerous

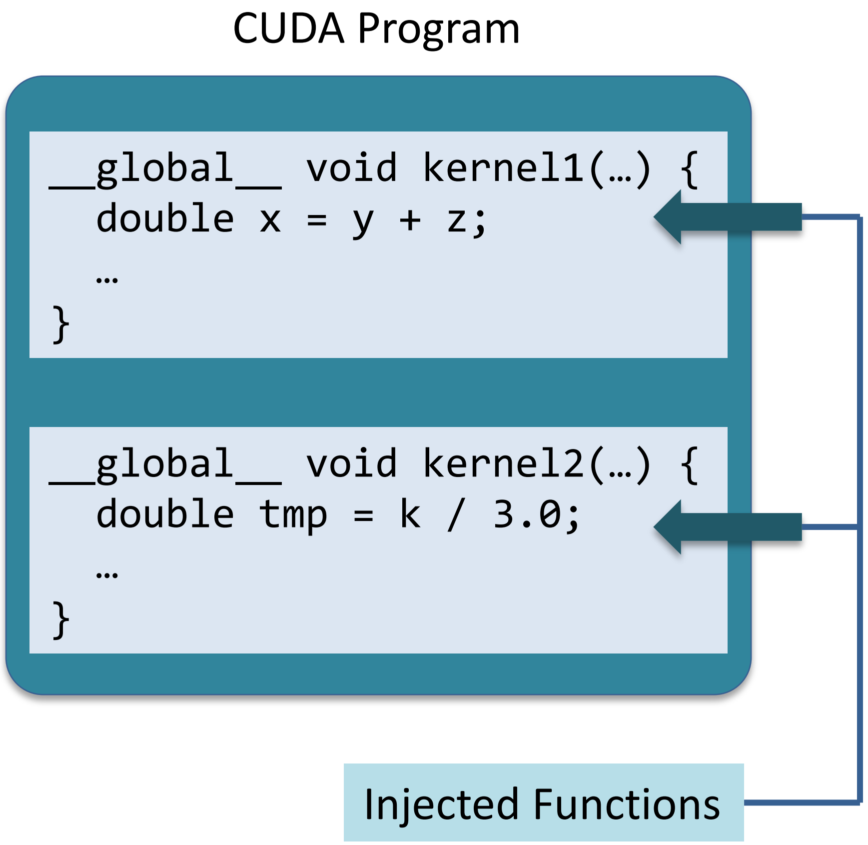
BinFPE: Dynamic Analysis via Binary Instrumentation

Why Dynamic Analysis?

- Previous work (FPChecker) uses static and dynamic analysis in LLVM
- Most HPC applications use `nvcc` (NVIDIA compiler); not LLVM
- Source is not available for some GPU libraries:
  — cuDNN, cuBLAS, cuFFT, cuSOLVER, CUDA Math API, ….

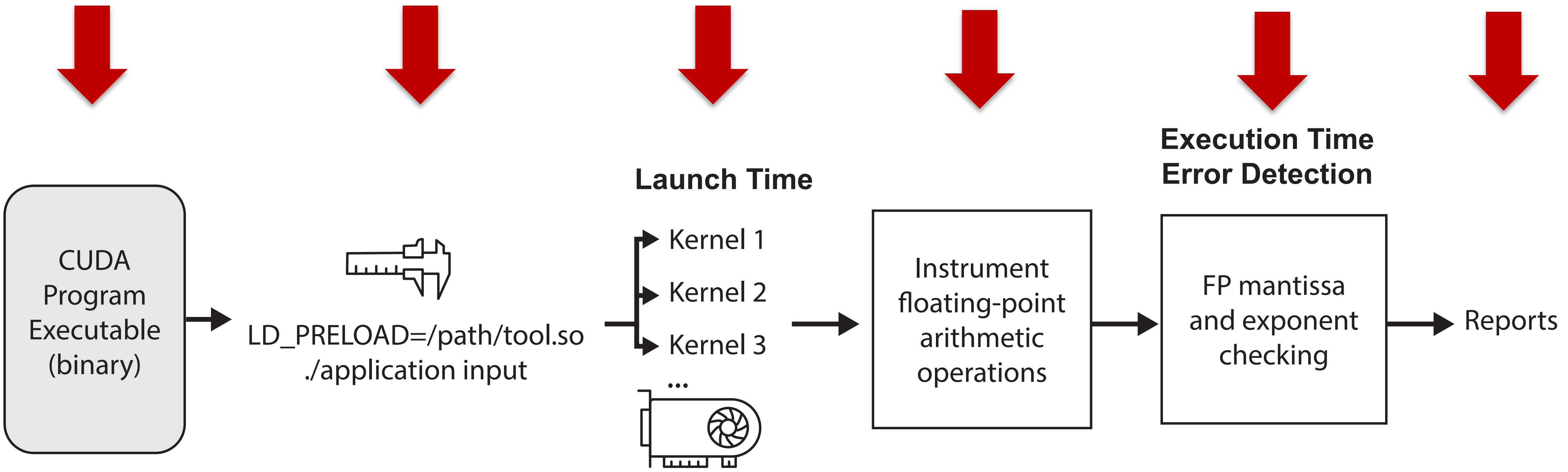
NVBit Overview

- Dynamic binary instrumentation framework for NVIDIA GPUs
- Provides APIs that allows:
  - Instruction inspection
  - Callbacks to CUDA driver APIs
  - Injection of arbitrary CUDA functions into any application before kernel launch
- The injected analysis functions are executed in the GPU
  - BinFPE: to monitor exceptions



Oreste Villa, Mark Stephenson, David Nellans, and Stephen W Keckler. 2019. ***NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs***. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. 372–383.

BinFPE's Workflow



- **Channel** between GPU-and-CPU
- Detection is performed in CPU

Value-Based Exception Detection

PTX: high-level language (ISA)
SASS: low-level architecture dependent assembly

Instruction format (SASS)

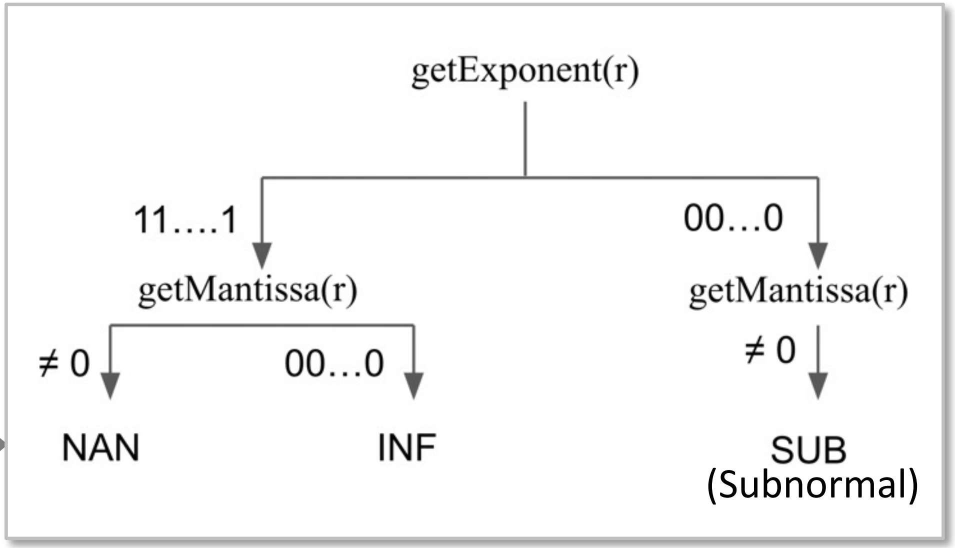
```
(Instr.) (Destination), (Param1), (Param2) ...
```

↓

Get a copy of Destination Value (register ***r***)

↓

Check *r* is FP32 or FP64 → Analyze exponent and mantissa



BinFPE Results from CUDA Programs

				FP32			FP64			Time (sec)		
Group	Program Name	Kernels	Instructions	NaN	INF	Under.	NaN	INF	Underf.	Normal	Instrum.	Slowdown
Rodinia	backprop	2	28421	0	0	0	0	0	0	0.481	1.218	2.53
	bfs	1	0	0	0	0	0	0	0	0.636	1.312	2.06
	cfd	5	–	**10**	0	**1**	**59**	**24**	**49**	–	–	–
	gaussian	3	33	0	0	0	0	0	0	0.483	1.165	2.41
	heartwall	2	795516513	0	0	0	0	0	0	1.195	652.782	546.26
	hotspot	1	592540	0	0	0	0	0	0	0.86	1.871	2.16
	lavaMD	1	289774400	0	0	0	0	0	0	0.584	226.918	388.56
	leukocyte	12	1293579493	0	0	0	0	0	0	0.797	1108.525	1390.87
	lud	3	205360	0	0	0	0	0	0	0.558	1.923	3.45
	nn	1	13370	0	0	0	0	0	0	0.488	0.943	1.93
	nw	2	0	0	0	0	0	0	0	0.58	1.354	2.339
	srad/srad_v1	9	60549150	0	0	0	0	0	0	0.606	64.075	105.73
	srad/srad_v2	2	18087936	0	0	0	0	0	0	0.814	20.258	24.89
	streamcluster	301	1704346338	0	0	0	0	0	0	4.971	1277.952	257.08
	pathfinder	1	0	0	0	0	0	0	0	0.972	1.394	1.43
Proxy apps.	Kripke	9	647310240	0	0	0	0	0	0	5.305	544.925	102.72
	LULESH	100	119537586	0	0	0	0	0	0	0.437	101.537	232.35
	SW4Lite	149	2817345320	0	0	0	0	0	**1**	3.643	2340	642.33
	CoMD	20	46164	0	0	0	0	0	0	0.307	3.932	12.81
NPB-GPU	BT	27	33500406	0	0	0	0	0	0	0.417	33.78	81.01
	LU	28	13398491	0	0	0	0	0	0	0.452	53.193	117.68
	SP	34	5744336	0	0	0	0	0	0	0.444	17.037	38.37
PolyBench-GPU	2DCONV	1	4716288	0	0	0	0	0	0	1.621	7.021	4.33
	2MM	2	536870912	0	0	0	0	0	0	52.618	672.222	12.78
	3DCONV	1	7741920	0	0	0	0	0	0	0.828	7.683	9.279
	3MM	3	12582912	0	0	0	0	0	0	1.588	15.91	10.02
	ATAX	2	1048576	0	0	0	0	0	**2**	0.559	2.132	3.81
	BICG	2	1048576	0	0	0	0	0	0	0.599	2.221	3.71
	CORR	4	137889344	0	0	0	0	0	0	11.878	315.372	26.55
	COVAR	3	136479232	0	0	0	0	0	0	11.927	313.643	26.30
	FDTD-2D	4	524224000	0	0	0	0	0	0	7.616	529.822	69.57
	GEMM	1	8396800	0	0	0	0	0	0	0.889	10.29	11.57
	GESUMMV	1	1048832	0	0	0	0	0	**3**	0.653	1.938	2.97
	GRAMSCHM	3	277628940	0	0	0	**1**	0	**3**	57.078	790.6	13.85
	MVT	2	1048576	0	0	0	0	0	0	0.641	2.26	3.52
	SYR2K	1	1342308352	0	0	0	0	0	0	32.417	1571.062	48.46
	SYRK	1	67141632	0	0	0	0	0	0	2.448	78.282	31.98

Strategy to Mitigate Numerical Inconsistencies and Exceptions

1

Detecting **Numerical Exceptions**

- Exceptions cause inconsistencies
- Detection is crucial
- Compiler and dynamic instrumentation

2

Finding Inputs that Cause Exceptions

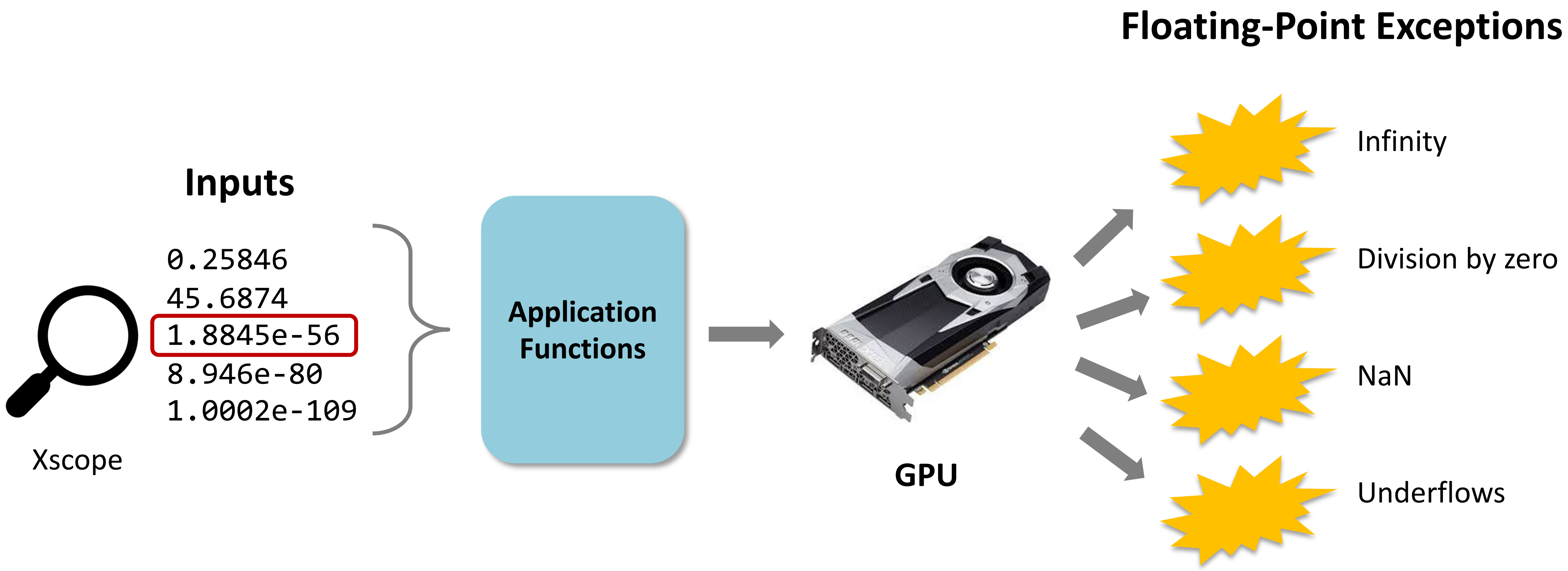
- Inputs induce exceptions
- Bayesian Optimization
- Mitigate bad inputs in testing

3

Isolating **Lines of Code**

- Isolate code that is impacted
- Search by enhancing precision
- Isolate expressions

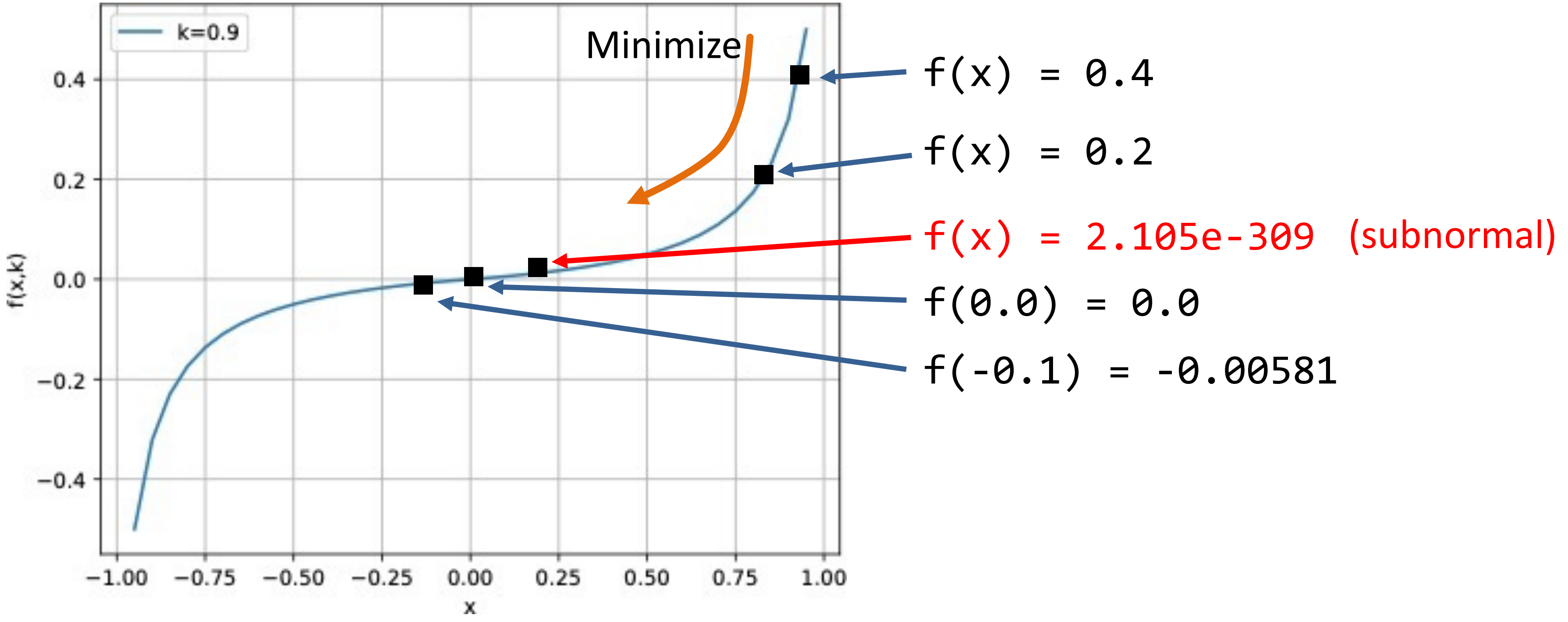
XScope in a Nutshell



Problem with BO Finding Underflows

Sigmoid function

```
__device__
double sigmoid(double x, double k) {
  double d = x - (k*x);
  double n = k - 2.0*k*abs(x) + 1.0;
  return d/n;
}
```



Problem:

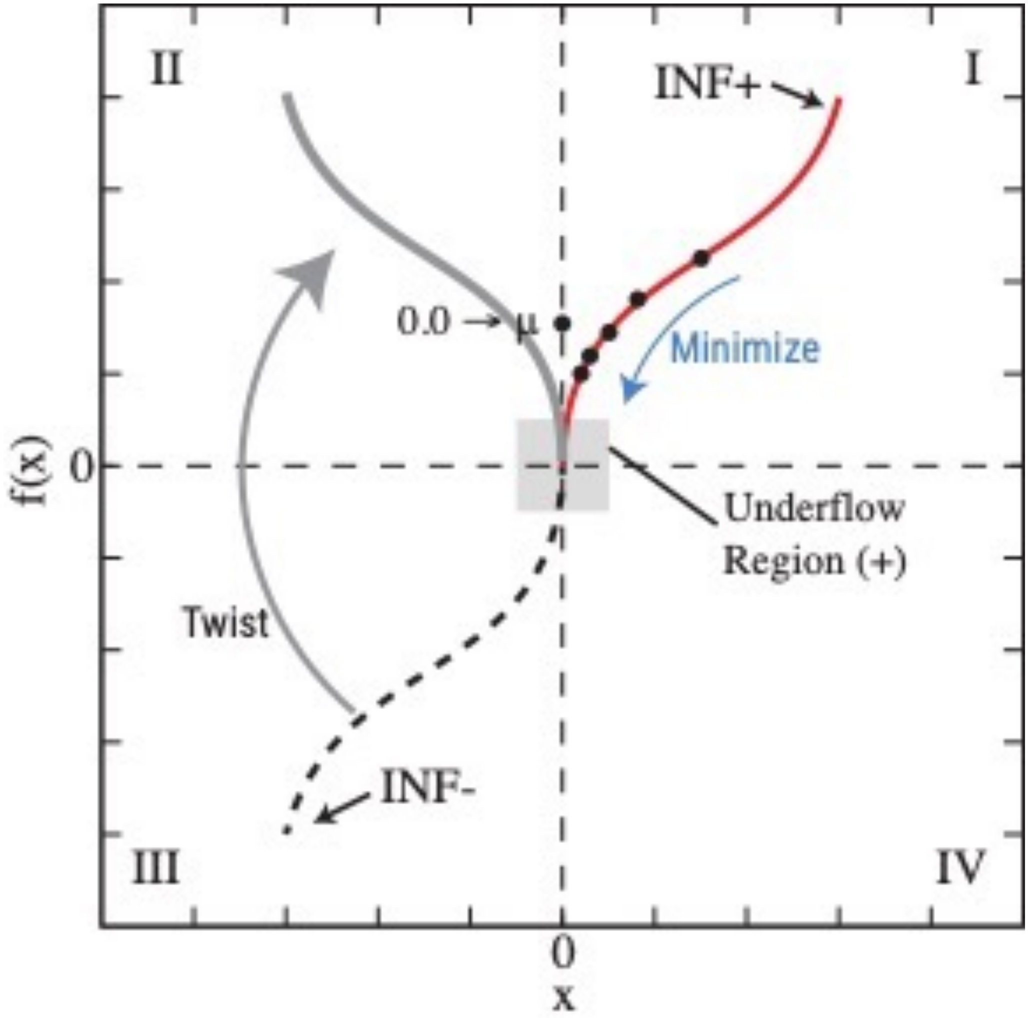
- BO can't stop when an underflow occurs
- We need to guide BO to identify underflows

Function Twisting

- Transform f in a new function f'
- Ask BO to minimize f'
- BO's output should be a subnormal number (underflow)

S_{min}^{+} : min subnormal (positive)

$$f'(x) = \begin{cases} \mu & \text{if } f(x) = 0.0 \text{ or } f(x) = -0.0 \\ \text{f(x)} & \text{if } f(x) \geq S_{min}^{+} \\ \text{-f(x)} & \text{if } f(x) < S_{min}^{+}. \end{cases}$$

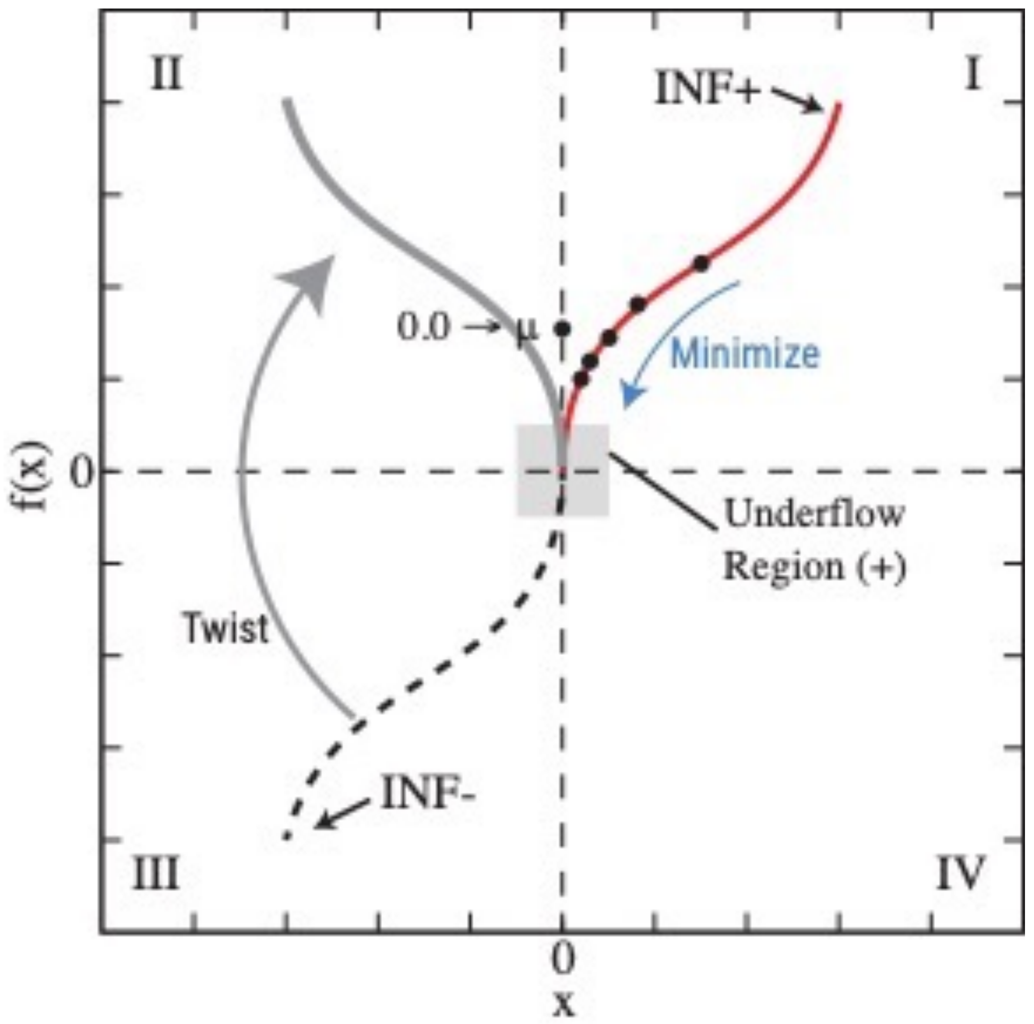


Function Twisting: Zero is a Special Case

- Zero is smaller than the minimum subnormal number
- BO could think zero is the smallest point
- BO may not stop at a subnormal number
- Zero gets a special value μ
- **Condition:**
  - μ must be greater than the largest subnormal
  - μ = 1 worked in practice

S^{+}_{min} : min subnormal (positive)

$$f'(x) = \begin{cases} \mu & \text{if } f(x) = 0.0 \text{ or } f(x) = -0.0 \\ \text{f(x)} & \text{if } f(x) \geq S^{+}_{min} \\ \text{-f(x)} & \text{if } f(x) < S^{+}_{min}. \end{cases}$$



Example: `cosh(double x)`

- Calculates the hyperbolic cosine of input argument *x*
- It's an increasing function
  ➔ it will produce INF as input increases
- Library documentation is not clear on specific inputs that trigger INF
- Xscope found inputs triggering INF:
  - 4.35e+3
  - 1e+47
  - 4.17+306

Documentation

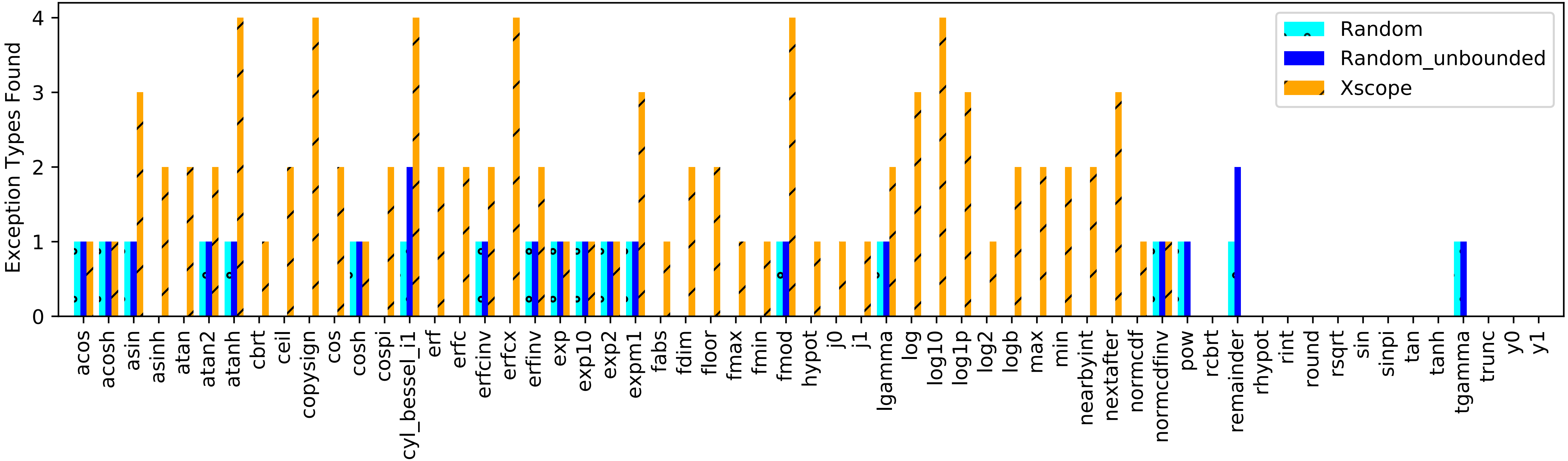
__device__ double cosh (double x)

Calculate the hyperbolic cosine of the input argument.

Returns

- cosh(± 0) returns 1.
- cosh($\pm\infty$) returns $+\infty$.

Comparison to Random Sampling



(1) Xscope: fp sampling + many-ranges

(2) Random: stops sampling inputs when the first exception is found

- This is how Xscope operates as well

(3) Random unbounded: does not stop sampling when an exception is found

All methods have the same number of trials (samples)

Strategy to Mitigate Numerical Inconsistencies and Exceptions

1

Detecting **Numerical Exceptions**

- Exceptions cause inconsistencies
- Detection is crucial
- Compiler and dynamic instrumentation

2

Finding Inputs that Cause Exceptions

- Inputs induce exceptions
- Bayesian Optimization
- Mitigate bad inputs in testing

3

Isolating **Lines of Code**

- Isolate code that is impacted
- Search by enhancing precision
- Isolate expressions

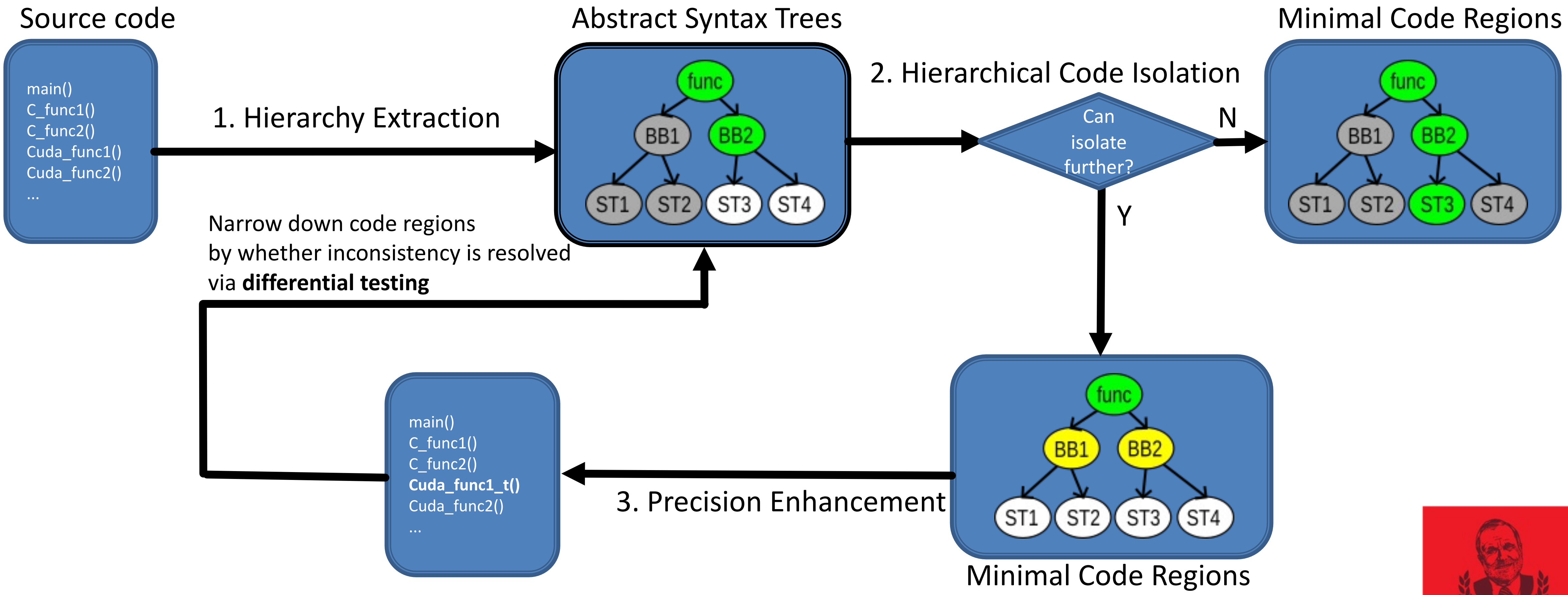
Compiler-induced Numerical Inconsistencies

- Motivational example 1: example from NMSE 3.3.4/FPBench[1]
  pow((x + 1.0), (1.0 / 3.0)) - pow(x, (1.0 / 3.0)); where x = 8291454011552366.0

Command line	Platform	Results
`nvcc -O0`	**CUDA**	**0**
`nvcc –O3 –use_fast_math`	**CUDA**	**0**
`gcc -O0`	**x64**	**2.9103830456733704e-11**
`gcc –O3 –ffast-math`	**x64**	**-5.8207660913467407e-11**

1. Toward a Standard Benchmark Format and Suite for Floating-Point Analysis
NSV'16: N. Damouche, M. Martel, P. Panchekha, C. Qiu, A. Sanchez-Stern, and Z. Tatlock

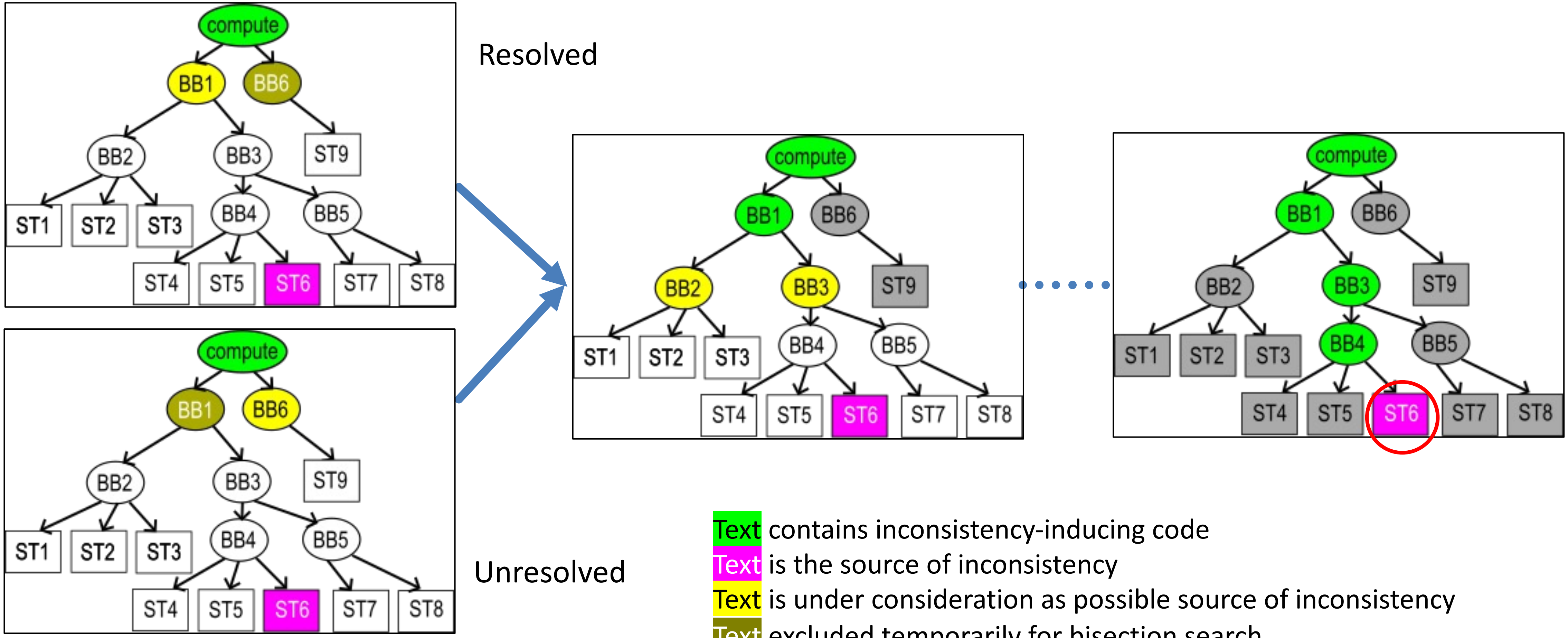
Ciel (Compiler-induced Inconsistency Expression Locator)



Miao, Dolores, Ignacio Laguna, and Cindy Rubio-González. "Expression Isolation of Compiler-Induced Numerical Inconsistencies in Heterogeneous Code." *High Performance Computing: 38th International Conference, ISC High Performance 2023, Hamburg, Germany, May 21–25, 2023.*

HANS MEUER AWARD

Hierarchical Bisection Search



Text contains inconsistency-inducing code
Text is the source of inconsistency
Text is under consideration as possible source of inconsistency
Text excluded temporarily for bisection search
Text is discarded from bisection search

Precision Enhancement

- Key Idea: **infinite precision is best**, but not possible
- Second best is enhanced precision (increased precision)
  - Avoid conditions that cause inconsistencies
  - Minimize rounding error caused by inconsistencies
- Can enhance precision for a single statement/expression

```
a = b * 2.0f + c    →    a = (float)((double)b * 2.0) + c
```

The expression itself is cast back to original precision

Inconsistency Analysis in Synthetic GPU Programs

- 330 randomly generated CUDA programs with compiler-induced inconsistencies
- **Successfully isolated code in 328 out of 330**
- Examples

```
+1.8922E-42f + var_3 ==> 0.0f + var_3

sinf(+1.0195E25f) ==> sin.approx.ftz.f32

-16458 / 1.67329e-16 ==> div.approx.ftz.f32

powf(-inf, 1.5f) ==> inf or 0.0 or nan
```

Ciel Isolates Inconsistencies in Heterogeneous Applications

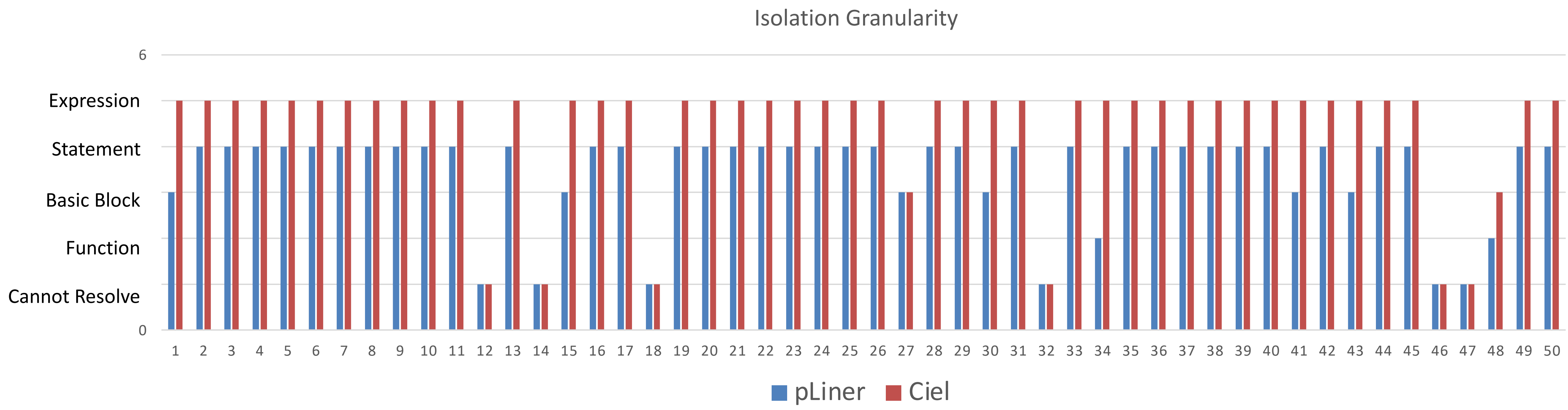
Program	Function	Lines	Expression Isolated?	#Configs	Time(m:s)
BT.S	exact_solution	1874-1886	Y	10	1:23
CG.S	sparse	1710,1722	Y	18	1:23
CG.W	sparse	1710,1713,1765	Y	19	1:34
LU.S	ssor_gpu_kernel_2	4023	Y	8	1:03
MG.W	rprj3_gpu_kernel	2045-2050	Y	14	1:16
CFD 097K	cuda_compute_step_factor	283	Y	14	6:01
CFD 193K	compute_speed_sqd	252,257	Y	10	7:29
LUD	lud_internal	---	N	17	1:16

- CLOUDSC: from leftover debug code, acknowledged by developers

```
zfallcorr = pow(yrecldp->rdensref/zrho[jl-1], (float)0.4);
```

Ciel Performs Better than State-of-the-art in Synthetic CPU Programs

- 50 programs generated by Varity, with inconsistency on x86 gcc 5.4.0
- Ciel uses 29.7% fewer searches to isolate statements
- Trade-off for isolating expressions: more searches (16.5 vs. 7.4)



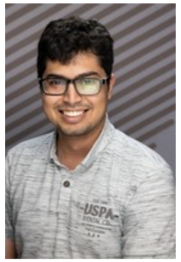
Collaborators and Contributors

University of Utah

Ganesh Gopalakrishnan

Xinyi Li

Tanmay Tirpankar



University of California, Davis

Cindy Rubio-González

Dolores Miao

Summary

1. Hidden floating-point exceptions can cause reproducibility issues
   - All exceptions must be addressed (to some degree) via testing
2. We provide tools to isolate exceptions in GPU programs
   - FPChecker: Clang/LLVM tool
   - BinFPE: binary instrumentation
   - Xscope: finds inputs that trigger exceptions
3. Identifying the source of inconsistencies is crucial
   - Ciel allows fine grained isolation of expressions

Thank you!

Contact: ilaguna@llnl.gov

Lawrence Livermore National Laboratory