Mitigating Numerical Inconsistencies and Exceptions in Heterogeneous HPC Systems

Ignacio Laguna

ZIH-COLLOQUIUM
Technische Universität Dresden
May 26, 2023
Numerical Reproducibility and Numerical Consistency Is Crucial

System 1

\[ x = 1.0001 \]
\[ y = 2.0001 \]
\[ z = 3.0001 \]

System 2

\[ x = 1.0001 \]
\[ y = 2.0001 \]
\[ z = 3.0001 \]

System 3

\[ x = 1.0001 \]
\[ y = 5.3746 \]
\[ z = \infty \]

Code

Code

Code
Real example of a numerical inconsistency

Hydrodynamics mini application

Early development and porting to new system (IBM Power8, NVIDIA GPUs)

- 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: $|e| = 144174.9336610391$
- 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
  \[1.23xxxxx\ldots\]

• New Hardware (e.g., GPU)

• New Compiler

• Optimizations

• Exceptions

• …

Focus of the talk
Strategy to Mitigate Numerical Inconsistencies and Exceptions

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

Finding Inputs that Cause Exceptions
- Inputs induce exceptions
- Bayesian Optimization
- Mitigate bad inputs in testing

Isolating Lines of Code
- Isolate code that is impacted
- Search by enhancing precision
- Isolate expressions
When a CPU exception 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

```c
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

Application Source: file1.cpp, file2.cu, ...

Compiler Front-End

Runtime System: runtime.h

CPU IR code

GPU device IR code

Optimizations

Instrumentation

Backend

Linker

Executable

Single checking function used due to ODR

https://fpchecker.org/

## Floating Point Exceptions and Events Detected by FPChecker

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflow</td>
<td>Result did not fit and it is an infinity</td>
<td>∞</td>
</tr>
<tr>
<td>Underflow</td>
<td>Result could not be represented as normal</td>
<td>0, subnormal</td>
</tr>
<tr>
<td>Divide by Zero</td>
<td>Divide-by-zero operation</td>
<td>∞</td>
</tr>
<tr>
<td>Invalid</td>
<td>Operation operand is not a number (NaN)</td>
<td>NaN</td>
</tr>
<tr>
<td>Inexact</td>
<td>Result is produced after rounding</td>
<td>normal</td>
</tr>
</tbody>
</table>

### Other events

<table>
<thead>
<tr>
<th>Other events</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison</td>
<td>Two numbers are compared for equality</td>
</tr>
<tr>
<td>Cancellation</td>
<td>Cancellation in addition or subtraction</td>
</tr>
<tr>
<td>Latent Infinity+</td>
<td>Large normal, close to positive infinity</td>
</tr>
<tr>
<td>Latent Infinity-</td>
<td>Large normal, close to negative infinity</td>
</tr>
<tr>
<td>Latent underflow</td>
<td>Small normal, close to subnormal</td>
</tr>
</tbody>
</table>
Example of Reproducibility Problem: *Subnormal Numbers and Optimizations*

- **Subnormal numbers**: very small quantities, \( 1 \text{e-309} \)

```c
double x = a/b;
if (x < …) {
    //Branch 1
} else {
    //Branch 2
}
```

**System 1**
- X86_64
- clang, gcc
- `-O3`, `-ffast-math`

**Branch 1**
- Optimization
- \( a/b \rightarrow a \times (1/b) \rightarrow \text{NaN} \)

**System 2**
- IBM POWER9
- xlc, `-O3`

**Branch 2**
## Main Report

<table>
<thead>
<tr>
<th>Event</th>
<th>Events</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinity (+)</td>
<td>0</td>
<td>High</td>
</tr>
<tr>
<td>Infinity (-)</td>
<td>0</td>
<td>High</td>
</tr>
<tr>
<td>NaN</td>
<td>860</td>
<td>High</td>
</tr>
<tr>
<td>Division by zero</td>
<td>0</td>
<td>High</td>
</tr>
<tr>
<td>Underflow (subnormal)</td>
<td>0</td>
<td>Medium</td>
</tr>
<tr>
<td>Comparison</td>
<td>0</td>
<td>Medium</td>
</tr>
<tr>
<td>Cancellation</td>
<td>478141486</td>
<td>Low</td>
</tr>
<tr>
<td>Latent Infinity (+)</td>
<td>0</td>
<td>Low</td>
</tr>
<tr>
<td>Latent Infinity (-)</td>
<td>0</td>
<td>Low</td>
</tr>
<tr>
<td>Latent underflow</td>
<td>0</td>
<td>Low</td>
</tr>
</tbody>
</table>
FPChecker Evaluation Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Model</th>
<th>$\infty+$</th>
<th>$\infty-$</th>
<th>NaN</th>
<th>DivByZero</th>
<th>Cancell.</th>
<th>Comp.</th>
<th>Underf.</th>
<th>Lat. $\infty+$</th>
<th>Lat. $\infty-$</th>
<th>Lat. Underf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LULESH</td>
<td>MPI, OpenMP</td>
<td>0</td>
<td>0</td>
<td>430</td>
<td>0</td>
<td>123,068,684</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kripke</td>
<td>MPI, OpenMP, RAJA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>256</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Quicksilver</td>
<td>MPI</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32,786,872</td>
<td>1,153,703,064</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RAJAPerf</td>
<td>RAJA, serial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3,306,835,400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AMG</td>
<td>MPI</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>7,885</td>
<td>31,385</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IS</td>
<td>serial</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1,052</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EP</td>
<td>serial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>134,163</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CG</td>
<td>serial</td>
<td>22,505</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22,505</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MG</td>
<td>serial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10,376</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FT</td>
<td>serial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4,222</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BT</td>
<td>serial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>739</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SP</td>
<td>serial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>752</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LU</td>
<td>serial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- 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, ...
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

BinFPE’s Workflow

CUDA Program Executable (binary)

LD_PRELOAD=/path/tool.so ./application input

Launch Time

Instrument floating-point arithmetic operations

Execution Time Error Detection

FP mantissa and exponent checking

Reports

- 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

- getExponent(\( r \))
  - 11...1
    - getMantissa(\( r \))
      - \( \neq 0 \)
        - INF
      - 00...0
    - \( \neq 0 \)
      - SUB (Subnormal)
  - 00...0
    - getMantissa(\( r \))
      - \( \neq 0 \)
Table 3.

<table>
<thead>
<tr>
<th>Group</th>
<th>Program Name</th>
<th>Kernels</th>
<th>Instructions</th>
<th>FP32 NaN</th>
<th>FP32 INF</th>
<th>FP32 Underl.</th>
<th>FP64 NaN</th>
<th>FP64 INF</th>
<th>FP64 Underl.</th>
<th>Normal</th>
<th>Instrum.</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>backprop</td>
<td>2</td>
<td>28421</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.481</td>
<td>1.218</td>
<td>2.53</td>
</tr>
<tr>
<td>bfs</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.636</td>
<td>1.312</td>
<td>2.06</td>
</tr>
<tr>
<td>cfd</td>
<td>5</td>
<td>–</td>
<td></td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>59</td>
<td>24</td>
<td>49</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>gaussian</td>
<td>3</td>
<td>33</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.483</td>
<td>1.165</td>
<td>2.41</td>
</tr>
<tr>
<td>heartwall</td>
<td>2</td>
<td>795516513</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.195</td>
<td>652.782</td>
<td>546.26</td>
</tr>
<tr>
<td>hotspot</td>
<td>1</td>
<td>592540</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.86</td>
<td>1.871</td>
<td>2.16</td>
</tr>
<tr>
<td>lavaMD</td>
<td>1</td>
<td>289774400</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.584</td>
<td>226.918</td>
<td>388.56</td>
</tr>
<tr>
<td>Rodinia</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>leukocyte</td>
<td>12</td>
<td>1293579493</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.797</td>
<td>1108.525</td>
<td>1390.87</td>
</tr>
<tr>
<td>lud</td>
<td>3</td>
<td>205360</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.558</td>
<td>1.923</td>
<td>3.45</td>
</tr>
<tr>
<td>nn</td>
<td>1</td>
<td>13370</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.488</td>
<td>0.943</td>
<td>1.93</td>
</tr>
<tr>
<td>nw</td>
<td>2</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.58</td>
<td>1.354</td>
<td>2.339</td>
</tr>
<tr>
<td>srad/srad_v1</td>
<td>9</td>
<td>60549150</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.606</td>
<td>64.075</td>
<td>105.73</td>
</tr>
<tr>
<td>srad/srad_v2</td>
<td>2</td>
<td>18087936</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.814</td>
<td>20.258</td>
<td>24.89</td>
</tr>
<tr>
<td>streamcluster</td>
<td>301</td>
<td>1704346338</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.971</td>
<td>1277.952</td>
<td>257.08</td>
</tr>
<tr>
<td>pathfinder</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.972</td>
<td>1.394</td>
<td>1.43</td>
</tr>
<tr>
<td>Proxy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kripke</td>
<td>9</td>
<td>647310240</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.305</td>
<td>544.925</td>
<td>102.72</td>
</tr>
<tr>
<td>LULESH</td>
<td>100</td>
<td>119537586</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.437</td>
<td>101.537</td>
<td>232.35</td>
</tr>
<tr>
<td>SW4Lite</td>
<td>149</td>
<td>2817345320</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.643</td>
<td>2340</td>
<td>642.33</td>
</tr>
<tr>
<td>CoMD</td>
<td>20</td>
<td>46164</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.307</td>
<td>3.952</td>
<td>12.81</td>
</tr>
<tr>
<td>NPB-GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>27</td>
<td>33500406</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.417</td>
<td>33.78</td>
<td>81.01</td>
</tr>
<tr>
<td>LU</td>
<td>28</td>
<td>13998491</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.452</td>
<td>53.193</td>
<td>117.68</td>
</tr>
<tr>
<td>SP</td>
<td>34</td>
<td>5744336</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.444</td>
<td>17.037</td>
<td>38.37</td>
</tr>
<tr>
<td>PolyBench-GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2DCONV</td>
<td>1</td>
<td>4716288</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.621</td>
<td>7.021</td>
<td>4.33</td>
</tr>
<tr>
<td>2MM</td>
<td>2</td>
<td>536870912</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>52.618</td>
<td>672.222</td>
<td>12.78</td>
</tr>
<tr>
<td>3DCONV</td>
<td>1</td>
<td>7741920</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.828</td>
<td>7.683</td>
<td>9.279</td>
</tr>
<tr>
<td>3MM</td>
<td>3</td>
<td>12582912</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.588</td>
<td>15.91</td>
<td>10.02</td>
</tr>
<tr>
<td>ATAX</td>
<td>2</td>
<td>1048576</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.559</td>
<td>2.132</td>
<td>3.81</td>
</tr>
<tr>
<td>BICG</td>
<td>2</td>
<td>1048576</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.599</td>
<td>2.221</td>
<td>3.71</td>
</tr>
<tr>
<td>CORR</td>
<td>4</td>
<td>137889344</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11.878</td>
<td>315.372</td>
<td>26.55</td>
</tr>
<tr>
<td>COVAR</td>
<td>3</td>
<td>136479232</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11.927</td>
<td>313.643</td>
<td>26.30</td>
</tr>
<tr>
<td>FDTD-2D</td>
<td>4</td>
<td>524224000</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7.616</td>
<td>529.822</td>
<td>69.57</td>
</tr>
<tr>
<td>GEMM</td>
<td>1</td>
<td>8396800</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.889</td>
<td>10.29</td>
<td>11.57</td>
</tr>
<tr>
<td>GESUMMV</td>
<td>1</td>
<td>1048832</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.653</td>
<td>1.938</td>
<td>2.97</td>
</tr>
<tr>
<td>GRAMSCHM</td>
<td>3</td>
<td>277628940</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>57.078</td>
<td>790.6</td>
<td>13.85</td>
</tr>
<tr>
<td>MVT</td>
<td>2</td>
<td>1048576</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.641</td>
<td>2.26</td>
<td>3.52</td>
</tr>
<tr>
<td>SYR2K</td>
<td>1</td>
<td>1342308352</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32.417</td>
<td>78.282</td>
<td>48.46</td>
</tr>
<tr>
<td>SYRK</td>
<td>1</td>
<td>67141632</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.448</td>
<td>78.282</td>
<td>31.98</td>
</tr>
</tbody>
</table>

BinFPE

Results from CUDA Programs
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

**Inputs**

0.25846
45.6874
1.8845e-56
8.946e-80
1.0002e-109

**Application Functions**

**Floating-Point Exceptions**

- Infinity
- Division by zero
- NaN
- Underflows

**GPU**
Problem with BO Finding Underflows

Sigmoid function

```c
__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;
}
```

<table>
<thead>
<tr>
<th>Number Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Normal (negative)</td>
<td>-1.00000e+07</td>
</tr>
<tr>
<td>Min Normal (positive)</td>
<td>1.00000e+07</td>
</tr>
<tr>
<td>Max Normal (negative)</td>
<td>-2.00000e-07</td>
</tr>
<tr>
<td>Max Normal (positive)</td>
<td>2.00000e-07</td>
</tr>
<tr>
<td>Min Subnormal (negative)</td>
<td>-5.00000e-38</td>
</tr>
<tr>
<td>Min Subnormal (positive)</td>
<td>5.00000e-38</td>
</tr>
<tr>
<td>Max Subnormal (negative)</td>
<td>-1.00000e+38</td>
</tr>
<tr>
<td>Max Subnormal (positive)</td>
<td>1.00000e+38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• BO can’t stop when an underflow occurs</td>
</tr>
<tr>
<td>• We need to guide BO to identify underflows</td>
</tr>
</tbody>
</table>

Minimize $f(x) = 0.4$

$f(x) = 0.2$

$f(x) = 2.105e-309$ (subnormal)

$f(0.0) = 0.0$

$f(-0.1) = -0.00581$
Function Twisting

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

\[ S_{\text{min}}^{+} : \min \text{subnormal (positive)} \]

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

Note that function twisting is not simply function flipping.

Also note that for finding inputs that trigger exceptional cases, such as NaN, INF, or INF+ when the quadrant I is flipped down to quadrant IV, BO’s output should be a subnormal number (underflow). This allows BO to move towards the underflow region.

More formally:

\[ f(x) = \begin{cases} 
0 & \text{if } -f(x) > 0 \\
\text{f}(x) & \text{if } f(x) > 0 \\
-f(x) & \text{if } f(x) < 0
\end{cases} \]

For the negative underflow region and stay in that region until BO is asked to minimize.

What about negative subnormals?

In other words, we transform the smallest subnormal number is found, at which point BO calls 2 since it may jump quickly to smaller values of $f$. Note that function twisting is not simply function flipping.

### Figure 5

Illustration of the function twisting idea.
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 \( \mu \)

**Condition:**
- \( \mu \) must be greater than the largest subnormal
- \( \mu = 1 \) worked in practice

\[
S_{min}^+ : \text{min subnormal (positive)}
\]

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

```
__device__ double cosh ( double x )

Calculate the hyperbolic cosine of the input argument.

Returns
• cosh( ±0 ) returns 1.
• cosh( ±∞ ) returns +∞.
```
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\(^1\)
  \[\text{pow}((x + 1.0), (1.0 / 3.0)) - \text{pow}(x, (1.0 / 3.0)); \text{ where } x = 8291454011552366.0\]

<table>
<thead>
<tr>
<th>Command line</th>
<th>Platform</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvcc -00</td>
<td>CUDA</td>
<td>0</td>
</tr>
<tr>
<td>nvcc -O3 -use_fast_math</td>
<td>CUDA</td>
<td>0</td>
</tr>
<tr>
<td>gcc -00</td>
<td>x64</td>
<td>2.9103830456733704e-11</td>
</tr>
<tr>
<td>gcc -O3 -ffast-math</td>
<td>x64</td>
<td>-5.8207660913467407e-11</td>
</tr>
</tbody>
</table>

1. Toward a Standard Benchmark Format and Suite for Floating-Point Analysis
   NSV'16: N. Damouche, M. Martel, P. Panchehka, C. Qiu, A. Sanchez-Stern, and Z. Tatlock
Ciel (Compiler-induced Inconsistency Expression Locator)

1. Hierarchy Extraction

Source code

- `main()`
- `C_func1()`
- `C_func2()`
- `Cuda_func1()`
- `Cuda_func2()`

Abstract Syntax Trees

- `func`
- `BB1`
- `BB2`
- `ST1`
- `ST2`
- `ST3`
- `ST4`

Minimal Code Regions

- `func`
- `BB1`
- `BB2`
- `ST1`
- `ST2`
- `ST3`
- `ST4`

2. Hierarchical Code Isolation

Can isolate further?

N

Y

Narrow down code regions by whether inconsistency is resolved via differential testing

3. Precision Enhancement

Minimal Code Regions

- `BB1`
- `BB2`
- `ST1`
- `ST2`
- `ST3`
- `ST4`

Hierarchical Bisection Search

Resolved

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

Unresolved
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

\[
a = b \times 2.0f + c \quad \rightarrow \quad a = (\text{float})(\text{double})b \times 2.0 + c
\]

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

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

• CLOUDSC: from leftover debug code, acknowledged by developers

```c
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
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