FloatGuard: Efficient Whole-Program Detection of Floating-Point Exceptions in AMD GPUs

Dolores Miao (UC Davis)
Ignacio Laguna (LLNL)
Cindy Rubio-González (UC Davis)

HPDC 2025
Notre Dame, IN, USA, 07.21.2025

AMD GPUs Gaining Traction in HPC

- Supercomputers like El Capitan and Frontier use AMD GPUs
- AMD GPU computing toolchain is maturing: ROCm
  - HIP kernel language with Clang compiler
  - Debugging tools such as ROCgdb
- Arising need in debugging numerical code, incl. FP Exceptions

AMD ROCm

Automated FP Exception Detection

1. Dinda et al. 2020. Spying on the Floating Point Behavior of Existing, Unmodified Scientific Applications. In HPDC. ACM, 5–16.
2. Laguna et al. 2022. FPChecker: Floating-Point Exception Detection Tool and Benchmark for Parallel and Distributed HPC. In IISWC. IEEE, 39–50.
3. Li et al. 2023. Design and Evaluation of GPU-FPX: A Low-Overhead tool for Floating-Point Exception Detection in NVIDIA GPUs. In HPDC. ACM, 59–71.

Platform	FP Exception Hardware	Tools / Approach	Mechanism & Notes
CPUs (x86-64)	✅ registers and traps	**FPSpy [1]**	Uses FP control/status register and signal-based trap-and-emulate to detect exceptions in unmodified binaries.
NVIDIA GPUs (CUDA)	❌ No hardware	FPChecker [2], GPU-FPX [3]	Compiler or binary instrumentation; high overhead; no native FP exception trapping.
AMD GPUs	✅ registers and traps	**???**	**(How can we leverage AMD's exception registers to natively track exceptions in GPU kernels?)**

Floating-Point Exceptions on AMD GPUs

Exception types not in IEEE 754

Exception Type	Abbr.	Trap	Mode	Descriptions
invalid operation	**NAN**	0	12	NaN as result, i.e. 0/0
input denormal	**IN_SUB**	1	13	Subnormal number in operand
divide by zero	**DIV0**	2	14	Division by zero, i.e. 10.0/0.0
overflow	**INF**	3	15	Result outside of range expressed by FP type
underflow	**OUT_SUB**	4	16	Subnormal number in result
inexact	**5**	5	17	Result not precisely represented, rounding is involved
int. divide by zero	**INT_DIV0**	6	18	Integer division by zero, i.e. 10/0

Floating-Point Exception Registers on AMD GPUs

- Mode register
  - Individually enable/disable types of exceptions
  - Reset at the beginning of every GPU kernel
- Trap status register
  - Accumulate exception state after they are encountered
  - Can be cleared at any point

Challenges using FP Exception Registers

Naïve thought: use ROCgdb manually to track exceptions

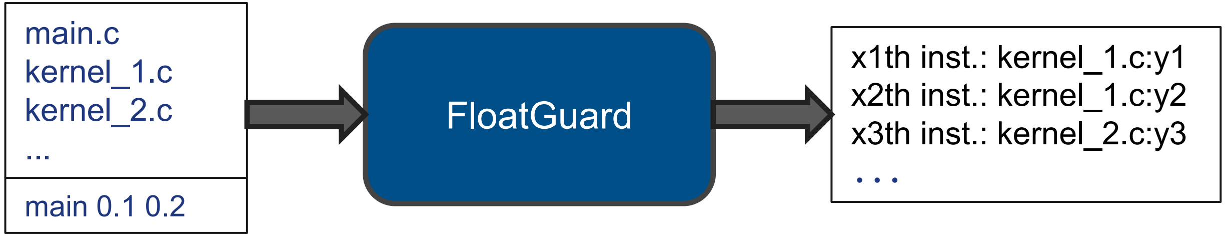
1. Exception trapping is off by default in kernels
   - Need to manually enable in each kernel thread
2. Program counter after a trap may be delayed
3. Program state unrecoverable with trapped exception
   - Difficult to track exception after the first

Conclusion: debugging manually is too time-consuming and thus calls for an automated approach

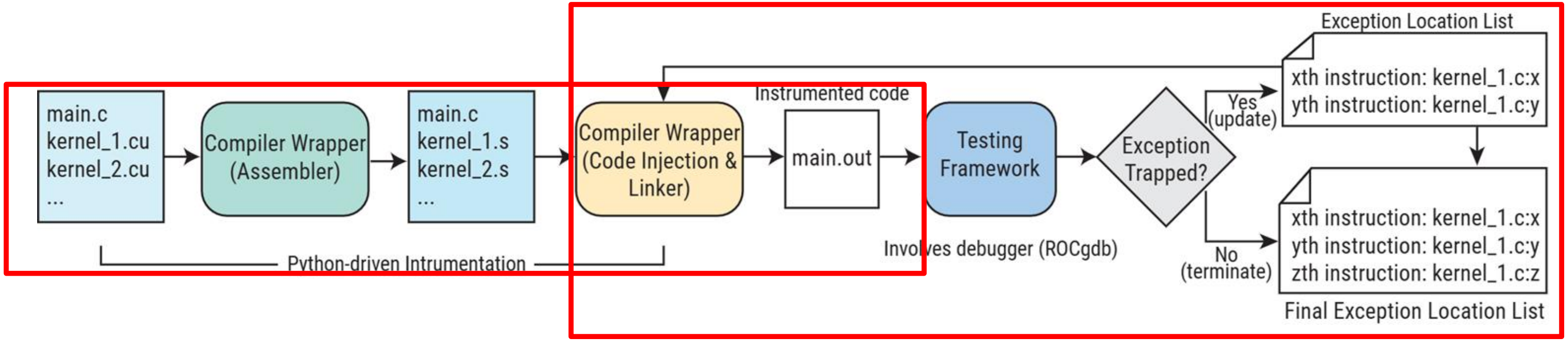
```
__global__ void
kernel_fp(int *gm, float a, float b) {
  int fret = (float)a / (float)b;
  *gm = fret;
}

__global__ void
kernel_mixed(int *gm, int a, int b) {
  int fret = a / b;
  fret = fret + (float)a / (float)b;
  *gm = fret;
}
```

A sample program with exceptions in 2 kernels

FloatGuard: first tool to detect floating-point exceptions on AMD GPUs

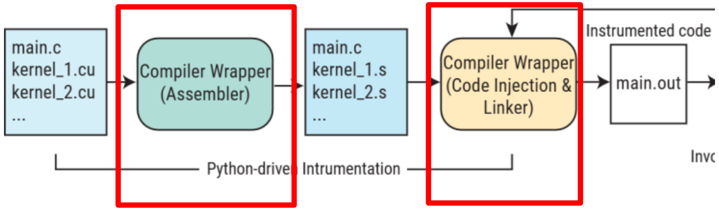


FloatGuard Workflow



Python-driven Code Instrumentation

- Compile source files to assembly (*.s) instead of objects (*.o)
- Inject instrumentation code into assembly
- Link to generate executables with code instrumentation



Our method has several advantages

- Inject code after all optimization passes in both frontend and backend are finished
- Compiler agnostic
- Only requires changing compiler in build scripts

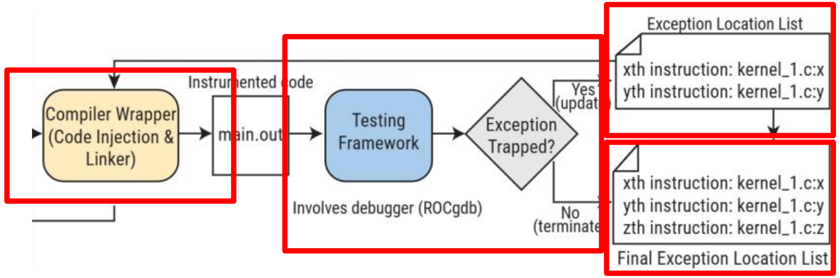
Python-driven Code Instrumentation – Assembly Injection

- At the beginning of kernels, enable exceptions
- Around code locations with previously reported exception
  - Disable before entering, enable after exiting

```
# enable exception; set to 0x2F0 to disable exception
s_mov_b32 s31, 0x5F2F0
s_setreg_b32 hwreg(HW_REG_MODE), s31
# clear trap status flags to report exception types correctly
s_setreg_imm32_b32 hwreg(HW_REG_TRAPSTS, 0, 7), 0
```

Testing Framework

- Run program until exception occur, record location
- Rerun assembly code instrumentation with updated info
- Link and run program again
- Rinse and repeat until no further exception is triggered



Evaluation Setup

- **56** benchmark programs
  - Detecting exceptions in real scientific codes
  - Rodinia, PolyBench-ACC, Parboil, SHOC, GPGPU-Sim, HPCG
  - Compiled with default flags and run on provided inputs
- **500** synthetic GPU programs generated by Varity [1]
  - Cases with existing compiler-induced inconsistencies
  - Compiled with –O3
- Test Machine: Ryzen 5 7500F + 32GiB RAM + RX 6650 XT
  - Also tested on CDNA2/RDNA3/etc. architectures
- ROCm 6.1.2 + Clang 17.0.0

1. Ignacio Laguna. 2020. Varity: Quantifying Floating-Point Variations in HPC Systems Through Randomized Testing. In IPDPS. IEEE, 622–633.

RQ1: Exception Detection Effectiveness

Exceptions in 9/56 benchmarks with all provided inputs

Benchmark Set	Benchmark	Total Exceptions	NAN	IN_SUB	DIV0	INF	OUT_SUB
Rodinia	cfd	12	6	0	6	0	0
Rodinia	myocyte	50	26	3	0	15	9
PolyBench-ACC	correlation	1	0	0	1	0	0
PolyBench-ACC	gramschmidt	2	1	0	1	0	0
PolyBench-ACC	lu	1	1	0	0	0	0
PolyBench-ACC	adi	4	4	0	0	0	0
SHOC	s3d	823	6	681	0	7	264
Parboil	stencil	2	0	1	0	0	1
GPGPU-Sim	wp	68	2	50	0	5	18

RQ1: Exception Detection Effectiveness

Exceptions found in 498/500 synthetic programs

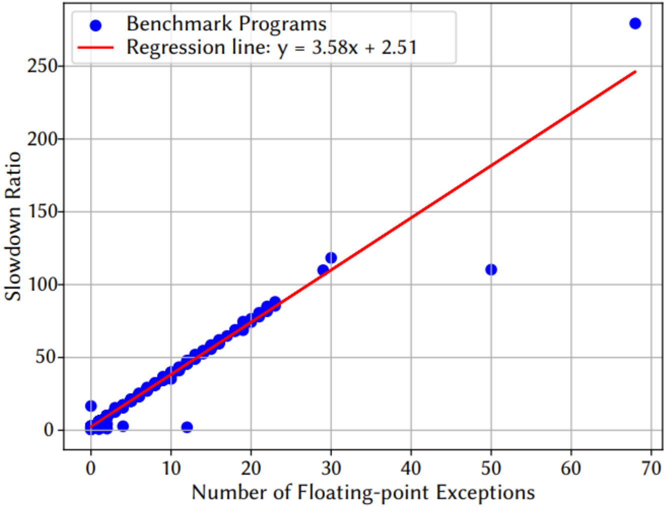
- The remaining 2 GPU programs have exceptions not reported under –O3
  - One **precalculates** all exception-occurring operations in compile time
  - The other has exception-occurring **dead code** that is removed in compile time

```
for (int i = 0; i < var_1; ++i) {
    comp = var_2 * 1.5271E-42f * (var_3 / floorf(
           -1.5532E35f - 1.8528E35f - var_4 * -1.4807E36f));
    comp = var_5 + 1.1129E35f + var_6 / powf(expf(1.4350E-36f),
           var_7 / -1.7503E11f * var_8 / 1.5272E-19f);
}
```

RQ1: Exception Detection Effectiveness

What about slowdown?

- Slowdown ratios approx. linearly related to the number of exceptions
- Sublinearly when running time is longer
- s3d has higher slowdown ratio due to longer linker time



RQ2: Exceptions & Floating-Point Optimization Flags

- Tested flags: **`-ffast-math -fdenormal-fp-math=preserve-sign`**
- Overall, fewer exceptions in general
  - Total exceptions in 56 benchmark programs dropped by 37.3%
  - Total exceptions in 500 synthetic programs dropped by 47%
- But there are cases with different or more exceptions, for example case_387 where var_2 is subnormal:

```
if( comp == cosf(var_1 - 1.8906E35f - (1.1216E14f / var_2 ) ) )
```

- If you use these flags in production, test your program with/without these optimizations, and take note of exceptions

RQ3: HIP and CUDA exception behavior comparison

Why test across platforms?

- Floating-point behavior varies with compilation and execution, especially across platforms
- Many GPU programs are written for CUDA and ported to HIP, leading to potential differences
- Studying exception handling between CUDA and HIP reveals platform-specific compliance and optimizations
- Aids debugging and ensures code portability
- Use FloatGuard on HIP; GPU-FPX [1] on CUDA

1. Li et al. 2023. Design and Evaluation of GPU-FPX: A Low-Overhead tool for Floating-Point Exception Detection in NVIDIA GPUs. In HPDC. ACM, 59–71.

RQ3: HIP and CUDA exception behavior comparison

Example synthetic programs that show differences in behavior

- case_450: division too small, results in zero; FloatGuard reports as an exception because it still has underflow

```
if ( comp < (-1.2964E-35f/var_2 ) )
```

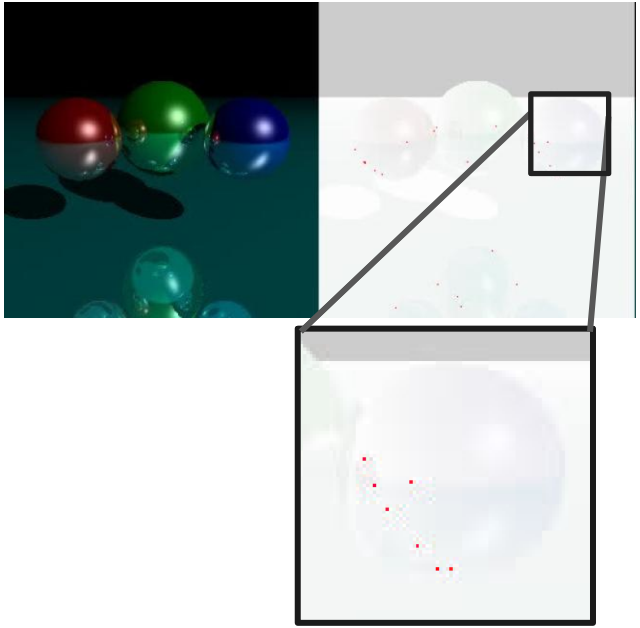
- case_350: **comp** variable was subnormal, seen as selection instruction result, triggers exception on GPU-FPX, but not on FloatGuard

```
if (comp <= (-0.0f - var_1 - 1.9945E-44f/-1.8945E36f)) {comp = …}
```

RQ3: HIP and CUDA exception behavior comparison

4 benchmark programs show different numerical behaviors

- For example, **GPGPU-Sim/rayTracing**: exceptions in CUDA due to different __saturatef() implementations, minor color difference in output
- Or **HPCG** where exceptions in CUDA only, due to underlying cuBLAS/hipBLAS library behaviors



FloatGuard Contributions

- The **first** tool to detect all floating-point exceptions in AMD HIP programs, utlizing registers and code instrumentation
- Implemented as FloatGuard, requires **minimal** build system change, and detects exceptions with **linear** slowdown relative to exception count
- Same GPU program shows **varying** numerical behaviors over FP optimizations, or between HIP and CUDA builds

Thank you!

Correspondence: Dolores Miao (wjmiao@ucdavis.edu / captainmieu@gmail.com)
Code repository: https://github.com/LLNL/FloatGuard

QR code for CV

I am currently seeking postdoc/academic/industry research opportunities—feel free to connect!