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

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

Tutorial @ SC25
St. Louis, MO, USA, 11.16.2025



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

Automated FP Exception Detection

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

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.

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

Live Demo 1 - Detecting FP Exceptions Manually

Prerequisite: AMD GPU + ROCm environment

1. Compile sample program
2. Run ROCgdb with the sample program; no exceptions
3. Use “b [kernel name]” to add breakpoints, then run the program again
4. When program is stopped, change mode register value
5. Exception occurs

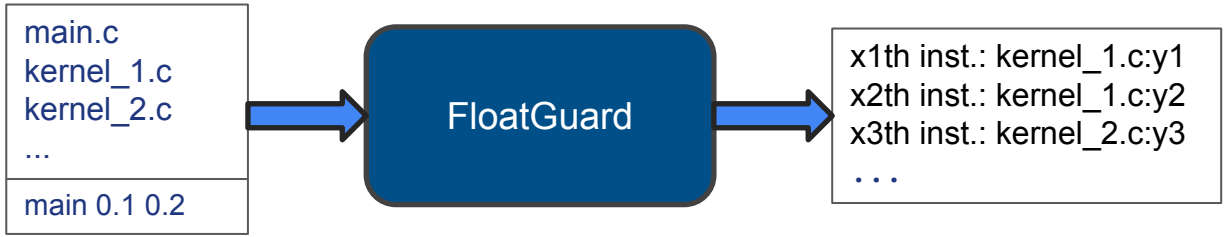
Challenges using FP Exception Registers

What can we conclude from our demo?

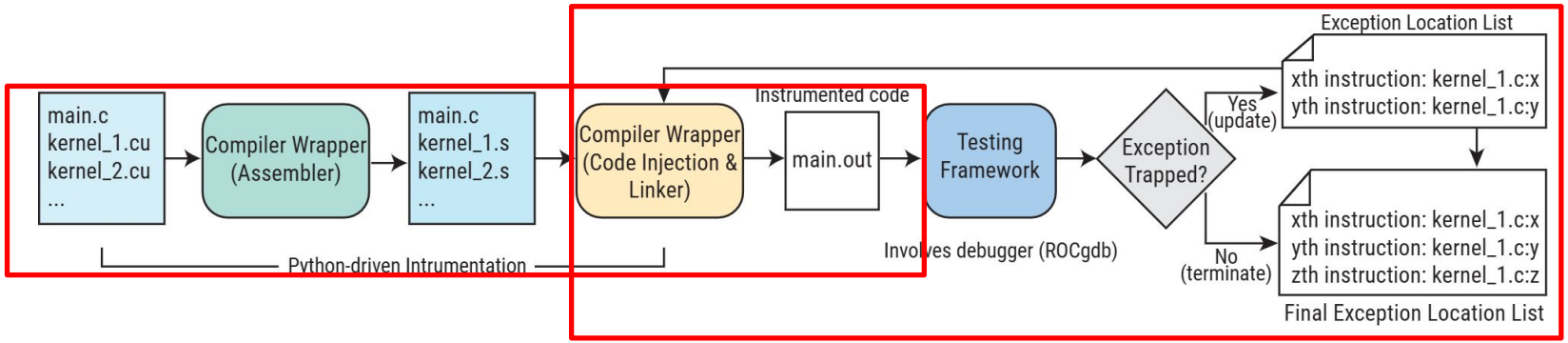
1. Exception trapping is off by default in kernels
   Need to manually enable in each kernel thread
2. Program counter after a trap may be delayed
3. Program state unrecoverable with trapped exception
   Difficult to track exception after the first

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

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



FloatGuard Workflow



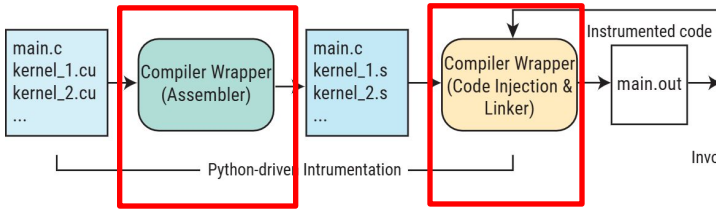
Code Instrumentation

Main steps:

- 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



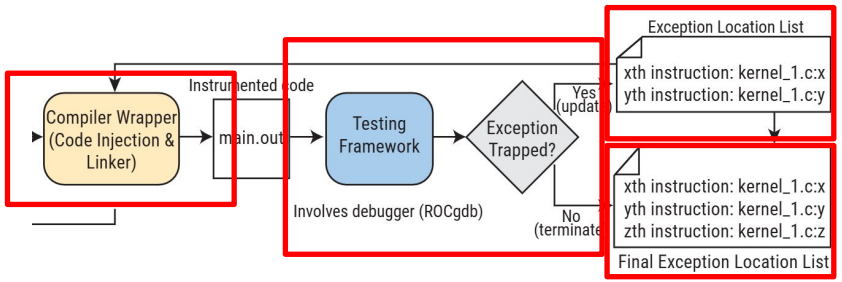
Code Instrumentation - Assembly Injection

- 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



Live Demo 2 - Running FloatGuard on Sample

Prerequisite: AMD GPU + Linux + ROCm environment (rocm + rocm-llvm-dev)
Clone code from here: https://github.com/LLNL/FloatGuard (sc25 branch)

1. Go to sample directory
2. Run: **`python3 [FloatGuard dir]/gdb_script/time_measure.py`**
3. Inspect results in the **`results/`** directory

Setup Your Code for FloatGuard

- Replace the compiler in your Makefile with our wrapper script

```
HIPCC = [FloatGuard Directory]/gdb_script/hipcc_wrapper.sh
```

- Some Makefile projects are small and only has one source file, and one command to compile and link the program. For those, make sure your compile and link commands are separate:

```
${HIPCC} ${HIPFLAGS} -c main.cpp -o main.o
${HIPCC} ${HIPFLAGS} -c other.cpp -o other.o
${HIPCC} ${LINKFLAGS} main.o other.o -o main
```

Setup Your Code for FloatGuard

- For CMake projects, replace the compiler in CMakeLists.txt

```
set(CMAKE_CXX_COMPILER [FloatGuard Dir]/gdb_script/hipcc_wrapper.sh)
```

- Create a **setup.ini** file in the root directory of your code project. For CMake projects, put the CMake command that creates project and compile here.

```
[DEFAULT]
compile = # the command line to compile the executable
run =     # the command line to run the executable
clean =   # the command line to clean the executable
```

Live Demo 3 - Running FloatGuard on Benchmarks

Prerequisite: AMD GPU + Linux + ROCm environment (rocm + rocm-llvm-dev)

1. Go to [benchmark directory]
2. Run: **`python3 [FloatGuard dir]/gdb_script/time_measure.py`**
3. Inspect results in the **`results/`** directory

Benchmark shown:

- rodinia/cfd
- PolyBench-ACC/lu

Thank you!

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

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