FPChecker

Detecting Floating-Point Exceptions in GPUs

Ignacio Laguna, Harshitha Menon, Tristan Vanderbruggen
Lawrence Livermore National Laboratory

Michael Bentley, Ian Briggs, Ganesh Gopalakrishnan
University of Utah

Cindy Rubio González
University of California at Davis

http://fpanalysistools.org/

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-PRES-780623).

Trapping Floating-Point Exceptions in CPU Code

Floating-Point Arithmetic Standard (IEEE 754)

1. Invalid operation
2. Division by zero
3. Overflow
4. Underflow
5. Inexact calculation

- When an exceptions occurs, it is signaled
  - System sets a flag or takes a trap
  - Status flag FPSCR set by default
- The system (e.g., Linux) can also cause the floating-point exception signal to be raised
  - SIGFPE

Source: https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.genprogc/floating-point_except.htm

CUDA has Limited Support for Detecting Floating-Point Exceptions

- CUDA: programming language of NVIDIA GPUs
- CUDA has no mechanism to detect exceptions
  - As of CUDA version: 10
- All operations behave as if exceptions are masked

You may have **“hidden”** exceptions in your CUDA program

Detecting the Result of Exceptions in a CUDA Program

- Place **printf** statements in the code (as many a possible)

```
double x = 0;
x = x/x;
printf("res = %e\n", x);
```

- Programming checks are available in CUDA:

```
__device__  int isnan ( float  a );
__device__  int isnan ( double  a );
```

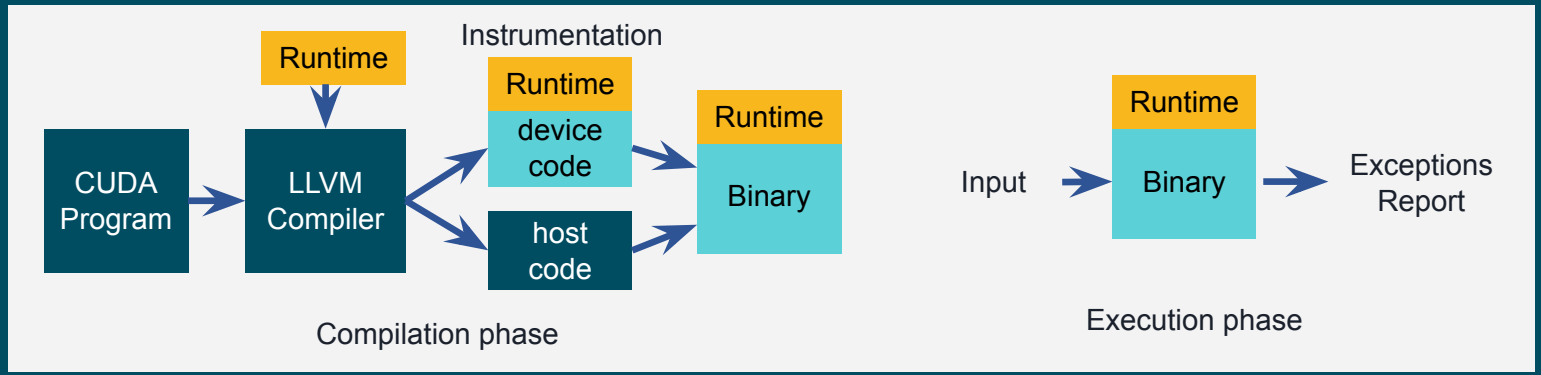
  - Also available isinf

These solutions are not ideal; they require significant programming effort

Goals of FPChecker

- Automatically detect the location of FP exceptions in NVIDIA GPUs
  - Report file & line number
  - No extra programming efforts required
- Report input operands
- Use software-based approach (compiler)
- Analyze optimized code

Workflow of FPChecker



How to Use FPChecker

1. Use clang as compiler for CUDA

2. Include path of FPChecker runtime system

3. Tell clang to load the instrumentation library

Example of Compilation Configuration

Use clang instead of NVCC

```
#CXX = nvcc
CXX = /path/to/clang++
CUFLAGS = -std=c++11 --cuda-gpu-arch=sm_60 -g
FPCHECK_FLAGS = -Xclang -load -Xclang /path/libfpchecker.so \
  -include Runtime.h -I/path/fpchecker/src
CXXFLAGS += $(FPCHECK_FLAGS)
```

- Load instrumentation library
- Include runtime header file

What Happens At Runtime?

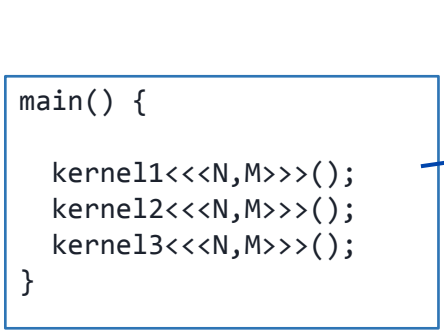
Mode 1: Errors abort

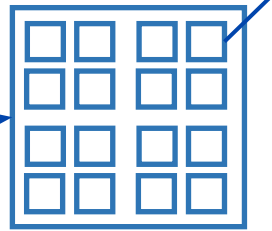
- If exception is detected, we signal a trap
- Kernel aborts execution

Mode 2: Errors don't abort

- If exception is detected, we store the location in global memory
- At the end of kernels, we check if exception occurred
- If so, it prints report
- Slightly higher overhead than mode 1

Errors Abort Mode



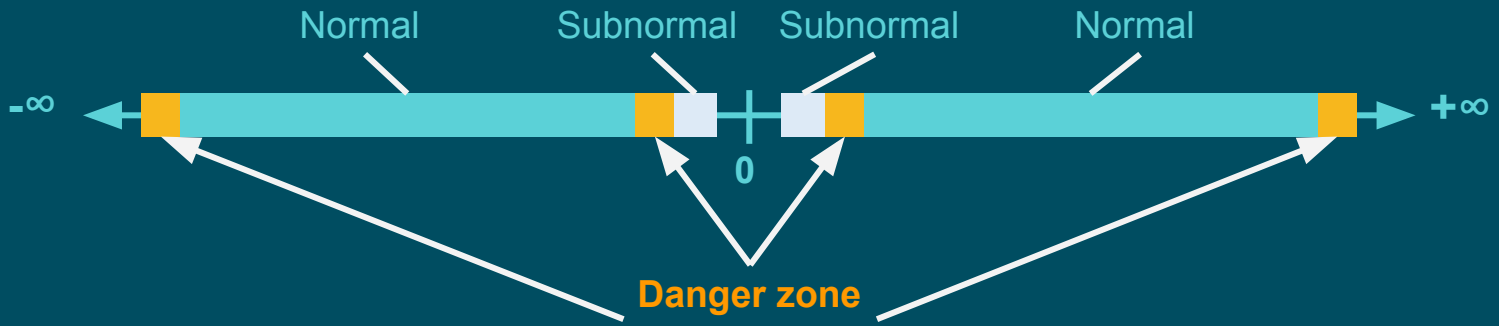


Interrupt routine:
- Threads (in block) get a lock
- First thread signals trap instruction

Given a floating-point operation
- Resulted in +INF or -INF?
- Resulted in NaN?
- Is an underflow?
- Is an overflow?
- Is latent underflow/overflow?

No synchronization when checking

We report **Warnings** for Latent Underflows/Overflows



- -D **FPC_DANGER_ZONE_PERCENT**=x.x:
  a. Changes the size of the danger zone.
  b. By default, x.x is 0.10, and it should be a number between 0.0 and 1.0.

Example of Error Report

```
+------------------------- FPChecker Error Report -------------------------+
Error          : Underflow
Operation      : MUL (9.999888672e-321)
File           : dot_product_raja.cpp
Line           : 32
+--------------------------------------------------------------------------+
```

Source code available:
https://github.com/LLNL/FPChecker

Questions?

Exercises

Exercises with FPChecker

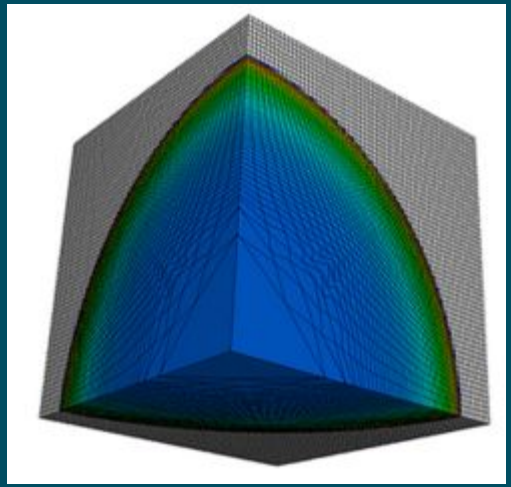
1. Compile and run CUDA application with Clang
2. Compile application with Clang & FPChecker
3. ERRORS_ABORT: NaN exception
4. ERRORS_DONT_ABORT: INF exception

Directory Structure

```
/Module-FPChecker
     |---/exercise-1
     |---/exercise-2
     |---/exercise-3
     |---/exercise-4
```

Application: LULESH

- Proxy application developed at LLNL
- Models a shock hydrodynamics problem
- LULESH version 2.0.2 for CUDA
  - Input: -s N
  - N: integer
  - Example: ./lulesh -s 10
    - Runs a 10x10x10 problem
- https://computation.llnl.gov/projects/co-design/lulesh



Exercise 1

Exercise 1: Compiling CUDA with Clang

- Open Makefile file
- Take a look at this compilation options:
  - NVCC = clang++
    - Indicates to use clang as the CUDA compiler
  - FLAGS = -g --cuda-gpu-arch=sm_35
    - Use debug information (-g)
    - Use CUDA compute capability (architecture) sm_35
- Execute:
  - $ make clean
  - $ make

Exercise 1: Output

```
$ make
clang++ -g --cuda-gpu-arch=sm_35  -Wno-mismatched-new-delete -Wno-format-extra-args -O3 -DNDEBUG
allocator.cu -I ./ -c -o allocator.o
clang++ -g --cuda-gpu-arch=sm_35  -Wno-mismatched-new-delete -Wno-format-extra-args -O3 -DNDEBUG
lulesh.cu -I ./   -c -o lulesh.o
clang++ -g --cuda-gpu-arch=sm_35  -Wno-mismatched-new-delete -Wno-format-extra-args -O3 -DNDEBUG
lulesh-comms.cu -I ./ -c -o lulesh-comms.o
clang++ -g --cuda-gpu-arch=sm_35  -Wno-mismatched-new-delete -Wno-format-extra-args -O3 -DNDEBUG
lulesh-comms-gpu.cu -I ./ -c -o lulesh-comms-gpu.o
clang++ -L/usr/local/cuda-8.0/lib64/ -lcuda -lcudart allocator.o lulesh.o lulesh-comms.o
lulesh-comms-gpu.o -o lulesh
```

Exercise 1: Running LULESH

- Run LULESH:
  - ./run_lulesh.sh
- Internally the scripts runs:
  - ./lulesh -s 10

```
$ ./run_lulesh.sh
Host ip-172-31-37-229 using GPU 0: Tesla K80
Running until t=0.010000, Problem size=10x10x10
cycle = 1, time = 6.042222e-05, dt=6.042222e-05
cycle = 2, time = 1.329289e-04, dt=7.250667e-05
cycle = 3, time = 1.577814e-04, dt=2.485252e-05
cycle = 4, time = 1.785352e-04, dt=2.075378e-05
...
...
cycle = 231, time = 1.000000e-02, dt=3.744566e-05
Run completed:
   Problem size        =  10
   MPI tasks           =  1
   Iteration count     =  231
   Final Origin Energy = 2.720531e+04
   Testing Plane 0 of Energy Array on rank 0:
        MaxAbsDiff   = 5.456968e-12
        TotalAbsDiff = 2.286042e-11
        MaxRelDiff   = 3.296482e-14

Elapsed time         =       0.05 (s)
Grind time (us/z/c)  = 0.21277922 (per dom)  (0.21277922 overall)
FOM                  =   4699.707 (z/s)
```

Exercise 2

Exercise 2: Compile Application with FPChecker

1. Open Makefile
2. Take a look at FPChecker flags

```
FPCHECKER_PATH  = /opt/fpchecker/install
LLVM_PASS       = -Xclang -load -Xclang $(FPCHECKER_PATH)/lib/libfpchecker.so \
-include Runtime.h -I$(FPCHECKER_PATH)/src

OTHER_FLAGS = $(LLVM_PASS) -Wno-mismatched-new-delete -Wno-format-extra-args

NVCC            = clang++
FLAGS           = -g --cuda-gpu-arch=sm_35
DFLAGS  = $(OTHER_FLAGS) -lineinfo
RFLAGS  = $(OTHER_FLAGS) -O3 -DNDEBUG
```

Exercise 2: Compile Application with FPChecker

- Run make:
  - make

FPChecker output

Some instructions are instrumented

```
$ make
clang++ -g --cuda-gpu-arch=sm_35  -Xclang -load -Xclang
/opt/fpchecker/install/lib/libfpchecker.so -include Runtime.h
-I/opt/fpchecker/install/src -Wno-mismatched-new-delete -Wno-format-extra-args -O3
-DNDEBUG  allocator.cu -I ./ -c -o allocator.o
#FPCHECKER: Initializing instrumentation
#FPCHECKER: Pointer value (fp32_check_add_function): 0
...
clang++ -g --cuda-gpu-arch=sm_35  -Xclang -load -Xclang
/opt/fpchecker/install/lib/libfpchecker.so -include Runtime.h
-I/opt/fpchecker/install/src -Wno-mismatched-new-delete -Wno-format-extra-args -O3
-DNDEBUG  lulesh.cu -I ./   -c -o lulesh.o
#FPCHECKER: Initializing instrumentation
#FPCHECKER: Pointer value (fp32_check_add_function): 0
#FPCHECKER: Found _FPC_DEVICE_CODE_FUNC_
#FPCHECKER: Found _FPC_PRINT_ERRORS_
...
#FPCHECKER: Entering main loop in instrumentFunction
#FPCHECKER: Instrumented operations: 15
#FPCHECKER: Leaving main loop in instrumentFunction
#FPCHECKER: Instrumenting function: _Z31CalcAccelerationForNodes_kerneliPdS_S_S_S_S_S_
#FPCHECKER: Entering main loop in instrumentFunction
#FPCHECKER: Instrumented operations: 4
```

Exercise 3

Exercise 3: NaN Exception & ERRORS_ABORT

- We inject a synthetic a NaN exception in LULESH
- FPChecker is run in ERRORS_ABORT mode
  - Detects the first exception
  - Reports the exception
  - Aborts

Exercise 3: Synthetic NaN Exception

- We inject a synthetic NaN exception in LULESH
  - See file: lulesh.cu
  - Line: 2868

```
2857 __global__
2858 void CalcAccelerationForNodes_kernel(int numNode,
2859                                       Real_t *xdd, Real_t *ydd, Real_t *zdd,
2860                                       Real_t *fx, Real_t *fy, Real_t *fz,
2861                                       Real_t *nodalMass)
2862 {
2863   int tid=blockDim.x*blockIdx.x+threadIdx.x;
2864   if (tid < numNode)
2865   {
2866       Real_t one_over_nMass = Real_t(1.)/nodalMass[tid];
2867       // NaN
2868       one_over_nMass = (one_over_nMass-one_over_nMass) / (one_over_nMass-one_over_nMass);
2869       xdd[tid]=fx[tid]*one_over_nMass;
2870       ydd[tid]=fy[tid]*one_over_nMass;
```

Exercise 3: FPChecker Detects NaN Exception

- Run lulesh:
  - ./run_lulesh.sh
- See FPChecker report
- Aborts after report is printed

```
$ ./run_lulesh.sh

========================================
 FPChecker (v0.1.0, Jun 23 2019)
========================================

Host ip-172-31-37-229 using GPU 0: Tesla K80
Running until t=0.010000, Problem size=10x10x10
+--------------------------- FPChecker Error Report ---------------------------+
 Error         : NaN
 Operation     : DIV
 File          : lulesh.cu
 Line          : 2868
+------------------------------------------------------------------------------+
terminate called after throwing an instance of 'thrust::system::detail::bad_alloc'
  what():  std::bad_alloc: an illegal instruction was encountered
./run_lulesh.sh: line 3:  3344 Aborted                 (core dumped) ./lulesh -s 10
```

Exercise 4

Exercise 4: INF Exception & ERRORS_DONT_ABORT

- We inject a synthetic a INF exception in LULESH
- FPChecker is run in ERRORS_DONT_ABORT mode
  - Reports the exception
  - It doesn't aborts on the first exception
  - Program continues running

Exercise 4: INF Exception & ERRORS_DONT_ABORT

Makefile

```
FPCHECKER_PATH  = /opt/fpchecker/install
LLVM_PASS       = -Xclang -load -Xclang $(FPCHECKER_PATH)/lib/libfpchecker.so \
-include Runtime.h -I$(FPCHECKER_PATH)/src -DFPC_ERRORS_DONT_ABORT

OTHER_FLAGS = $(LLVM_PASS) -Wno-mismatched-new-delete -Wno-format-extra-args

NVCC            = clang++
FLAGS           = -g --cuda-gpu-arch=sm_35
DFLAGS  = $(OTHER_FLAGS) -lineinfo
RFLAGS  = $(OTHER_FLAGS) -O3 -DNDEBUG
```

Exercise 4: FPChecker Detects INF Exception

- Run lulesh:
  - ./run_lulesh.sh
- FPChecker report is a single line
- Program continues to run after the error report
- A warning is also reported

```
$ ./run_lulesh.sh

=======================================
 FPChecker (v0.1.0, Jun 23 2019)
=======================================

Host ip-172-31-37-229 using GPU 0: Tesla K80
Running until t=0.010000, Problem size=10x10x10
cycle = 1, time = 6.042222e-05, dt=6.042222e-05
cycle = 2, time = 1.329289e-04, dt=7.250667e-05

#FPCHECKER: INF Error at lulesh.cu:2871 (code:#-2, tid:0)
cycle = 3, time = 1.577814e-04, dt=2.485252e-05
cycle = 4, time = 1.785352e-04, dt=2.075378e-05

#FPCHECKER: Warning at lulesh.cu:2871 (#-1.213789e+308, tid:0)
cycle = 5, time = 1.970364e-04, dt=1.850120e-05
cycle = 6, time = 2.142156e-04, dt=1.717920e-05
cycle = 7, time = 2.305486e-04, dt=1.633299e-05
cycle = 8, time = 2.463245e-04, dt=1.577590e-05
cycle = 9, time = 2.617391e-04, dt=1.541457e-05
cycle = 10, time = 2.769363e-04, dt=1.519719e-05
cycle = 11, time = 2.951729e-04, dt=1.823663e-05
...
```