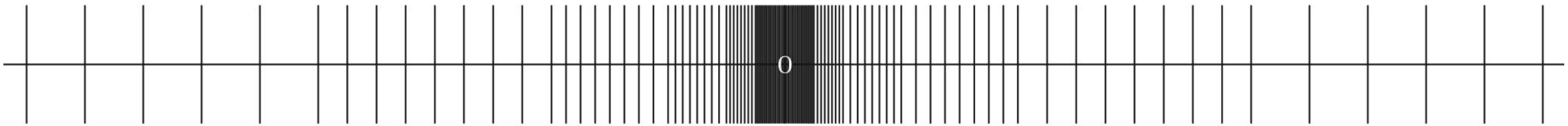
Tools to Detect and Diagnose Floating-Point Errors in Heterogeneous Computing Hardware and Software

A Half-Day Tutorial at Supercomputing 2025

Tut 122: Sunday, 16 November 2025, 8:30am - 12:00pm CST : **Location 121**



Lawrence Livermore National Laboratory

KAHLERT SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

UC DAVIS
UNIVERSITY OF CALIFORNIA

Tools to Detect and Diagnose Floating-Point Errors in Heterogeneous Computing Hardware and Software

A Half-Day Tutorial at Supercomputing 2025

Presenters

Ignacio Laguna

Lawrence Livermore National Laboratory

Mark Baranowski and
Ganesh Gopalakrishnan

KAHLERT SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

Dolores Miao and
Cindy Rubio-González

UC DAVIS
UNIVERSITY OF CALIFORNIA

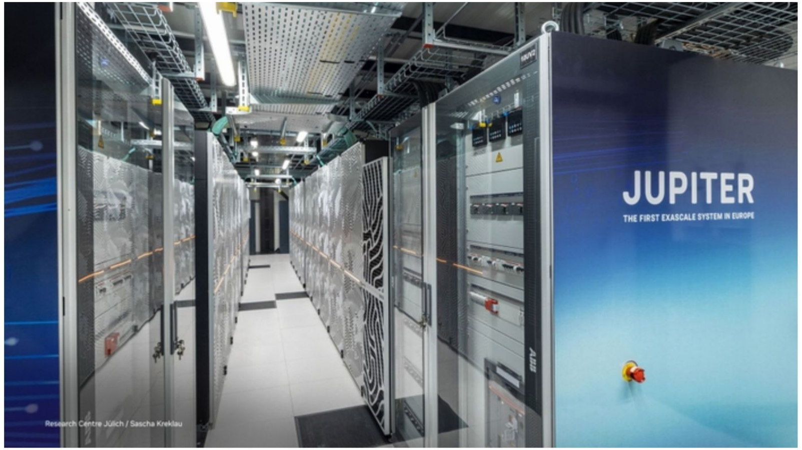
Agenda, and a single QR code to access all our material

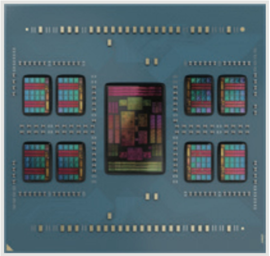
- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Plz Fill Survey
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Plz Fill Survey

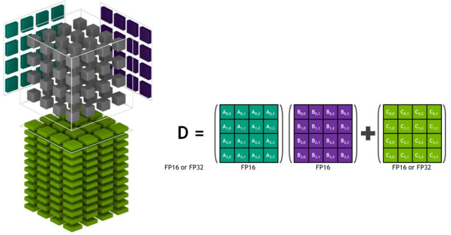
SC 25 Floating-Point Tutorial : All Tools and Benchmarks

fpanalysistools.org

Floating-Point Computations are fundamental to CPUs and GPUs



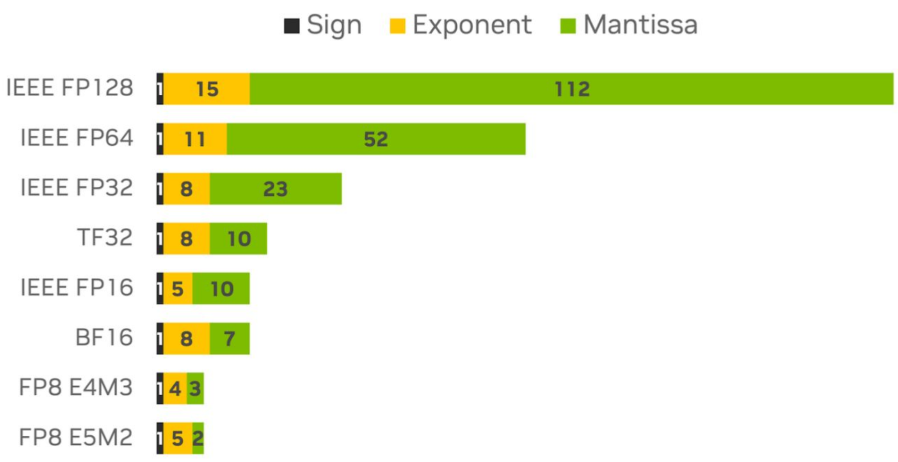




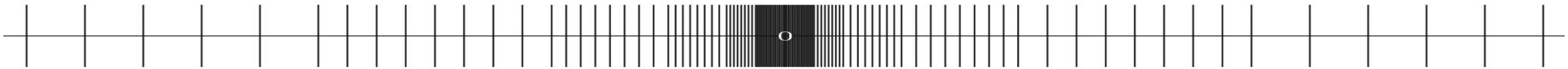
Floating-Point Numbers : A compromise, with many good properties, that can span a large range with a small number of bits

Half the representable numbers lie between -1 and +1

Format	Prec.	Exp.	Sm. norm.	Lrgst. Norm.	ULP @ emax	**eql. 2'c bits**	Redn. factor
FP16 16 bits	10	5	6.10 E-5	6.55 E4	5	41	2.5
BF16 16 bits	7	8	1.18 E-38	3.39 E38	120	262	16.3
TF32 19 bits	10	8	1.18 E-38	3.40 E38	117	265	13.9
FP32 32 bits	23	8	1.18 E-38	3.40 E38	104	254	7.9
FP64 **64 bits**	52	11	2.23 E-308	1.80 E308	**971**	**2099 bits**	32.7



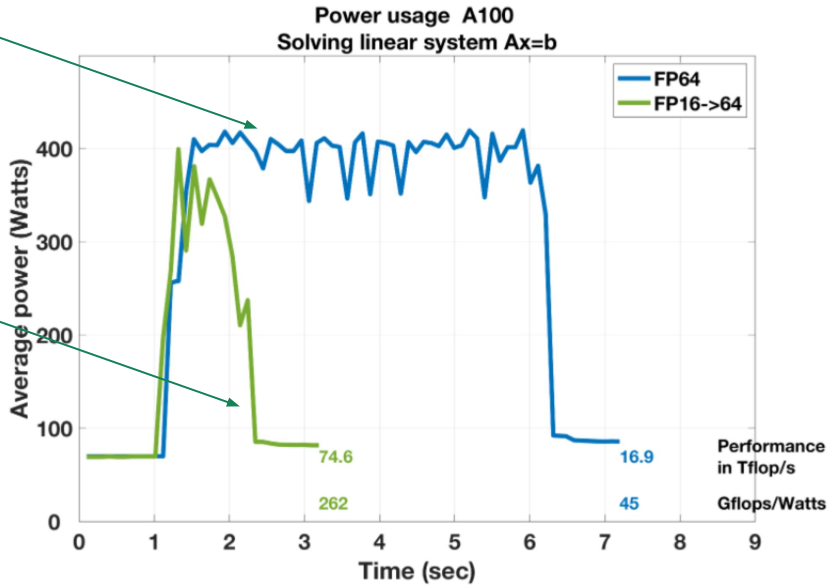
FP allows us to get away moving **64 bits** …. instead of moving **2099 bits (in 2's comp.)**



Lower precision Floating-Point saves **energy** (and **time**)

High-Precision
LU Solving

Low-Precision
LU Solving



From "Hardware Trends Impacting Floating-Point Computations In Scientific Applications," Dongarra et al.

While increasing precision almost always improves accuracy, there are rare cases where this does not happen.

- The loop in FP32 iterates 10 times (as per real semantics)

- In FP64, it iterates 11 times

```
#include <stdio.h>

int main() {
  int cnt=0;
  for (float i = 0.0f; i < 1.0f; i += 0.1) {
    cnt++;
    printf("%.1f ", i);
  }
  printf("\n");
  printf("cnt=%d\n",cnt);
  return 0;
-UU-:---   F1  floatCnt.c      Top   L5     (C/*l Abbrev) ---------------------------
#include <stdio.h>

int main() {
  int cnt=0;
  for (double i = 0.0f; i < 1.0f; i += 0.1) {
    cnt++;
    printf("%.1f ", i);
  }
  printf("\n");
  printf("cnt=%d\n",cnt);
  return 0;
-UU-:---   F1  doubCnt.c       Top   L1     (C/*l Abbrev) ---------------------------
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ gcc -o fc floatCnt.c
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ gcc -o dc doubCnt.c
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ ./fc
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
cnt=10
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ ./dc
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
cnt=11
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
```

Non-intuitive behavior caused by Rounding!

```
File Edit Options Buffers Tools C Help
#include <stdio.h>

int main() {
  int cnt=0;
  for (float i = 0.0f; i < 1.0f; i += 0.1) {
    cnt++;
    printf("%.1f ", i);
  }
  printf("\n");
  printf("cnt=%d\n",cnt);
  return 0;
-UU-:---  F1  floatCnt.c     Top   L5     (C/*l Abbrev) ----------------------------
#include <stdio.h>

int main() {
  int cnt=0;
  for (double i = 0.0f; i < 1.0f; i += 0.1) {
    cnt++;
    printf("%.1f ", i);
  }
  printf("\n");
  printf("cnt=%d\n",cnt);
  return 0;
-UU-:---  F1  doubCnt.c      Top   L1     (C/*l Abbrev) ----------------------------
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ gcc -o fc floatCnt.c
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ gcc -o dc doubCnt.c
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ ./fc
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
cnt=10
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ ./dc
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
cnt=11
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
```

```
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ gc
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ ./
Value: 0.00000000000000000000
(sign: 0, exp: 0, mantissa: 0x000000)
Value: 0.10000000149011611938
(sign: 0, exp: 123, mantissa: 0x4CCCCD)
Value: 0.20000000298023223877
(sign: 0, exp: 124, mantissa: 0x4CCCCD)
Value: 0.30000001192092895508
(sign: 0, exp: 125, mantissa: 0x19999A)
Value: 0.40000000596046447754
(sign: 0, exp: 125, mantissa: 0x4CCCCD)
Value: 0.50000000000000000000
(sign: 0, exp: 126, mantissa: 0x000000)
Value: 0.60000002384185791016
(sign: 0, exp: 126, mantissa: 0x19999A)
Value: 0.70000004768371582031
(sign: 0, exp: 126, mantissa: 0x333334)
Value: 0.80000007152557373047
(sign: 0, exp: 126, mantissa: 0x4CCCCE)
Value: 0.90000009536743164062
(sign: 0, exp: 126, mantissa: 0x666668)
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$ ./
Value: 0.00000000000000000000
(sign: 0, exp: 0, mantissa: 0x0000000000000)
Value: 0.10000000000000000555
(sign: 0, exp: 1019, mantissa: 0x999999999999A)
Value: 0.20000000000000001110
(sign: 0, exp: 1020, mantissa: 0x999999999999A)
Value: 0.30000000000000004441
(sign: 0, exp: 1021, mantissa: 0x3333333333334)
Value: 0.40000000000000002220
(sign: 0, exp: 1021, mantissa: 0x999999999999A)
Value: 0.50000000000000000000
(sign: 0, exp: 1022, mantissa: 0x0000000000000)
Value: 0.59999999999999997780
(sign: 0, exp: 1022, mantissa: 0x3333333333333)
Value: 0.69999999999999995559
(sign: 0, exp: 1022, mantissa: 0x6666666666666)
Value: 0.79999999999999993339
(sign: 0, exp: 1022, mantissa: 0x9999999999999)
Value: 0.89999999999999991118
(sign: 0, exp: 1022, mantissa: 0xCCCCCCCCCCCCC)
Value: 0.99999999999999988898
(sign: 0, exp: 1022, mantissa: 0xFFFFFFFFFFFFF)
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
(base) ganesh@ganesh-XPS-17-9710:~/parfloat-cs6961/c-prec-expts$
```

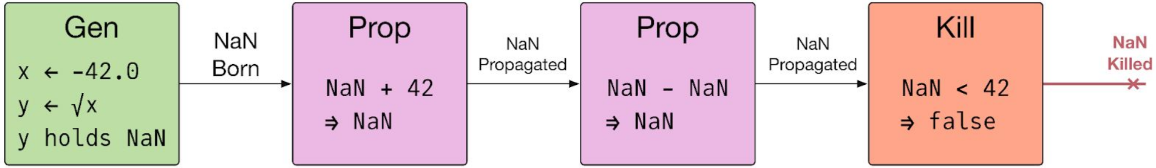
Focus of this tutorial

- Excessive Rounding Error (not addressed in this tutorial)
- Exhausting the range **(**addressed**)**
    - **Detection of Exceptions (main focus)**

FP Exceptions (Exceptional Values) are Undesirable (*often* are bugs)

- FP32: (1.9×10^{19})^2 $\rightarrow$ INF … (in E4M3, 16^2 = INF) ⇐ a popular ML low-prec format
  - Easily generated during dot-product
- Of the FP Exceptions, we focus on INF and NaN
  - NaNs can skew control-flow
- This macro is buggy
  - #define MAX(x,y) ((x >= y) ? x : y)

  hint : consider x == NaN or y == NaN
- Exceptions often vanish harmlessly (e.g. in ML codes)
  - But it is difficult to know which control-flows they have affected meanwhile



Tools Presented

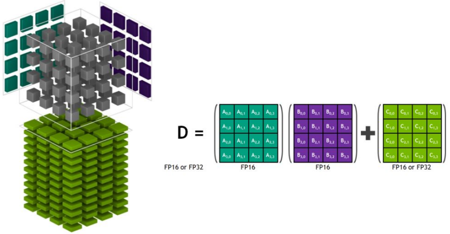
FPChecker

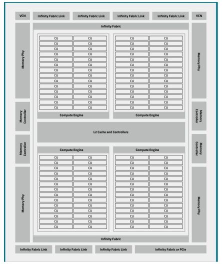
GPU-FPX + NixNaN

FloatGuard

NVIDIA GPUs

LLVM





AMD GPUs

Lawrence Livermore National Laboratory

KAHLERT SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

UC DAVIS
UNIVERSITY OF CALIFORNIA

Agenda

- 08:30 : Introduction
- **08:35 : FPChecker**
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

FPChecker

Tutorial

FPChecker:

Floating-Point Profiling Through Compiler-Instrumentation

https://fpchecker.org

Ignacio Laguna

LLNL-PRES-2012776
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

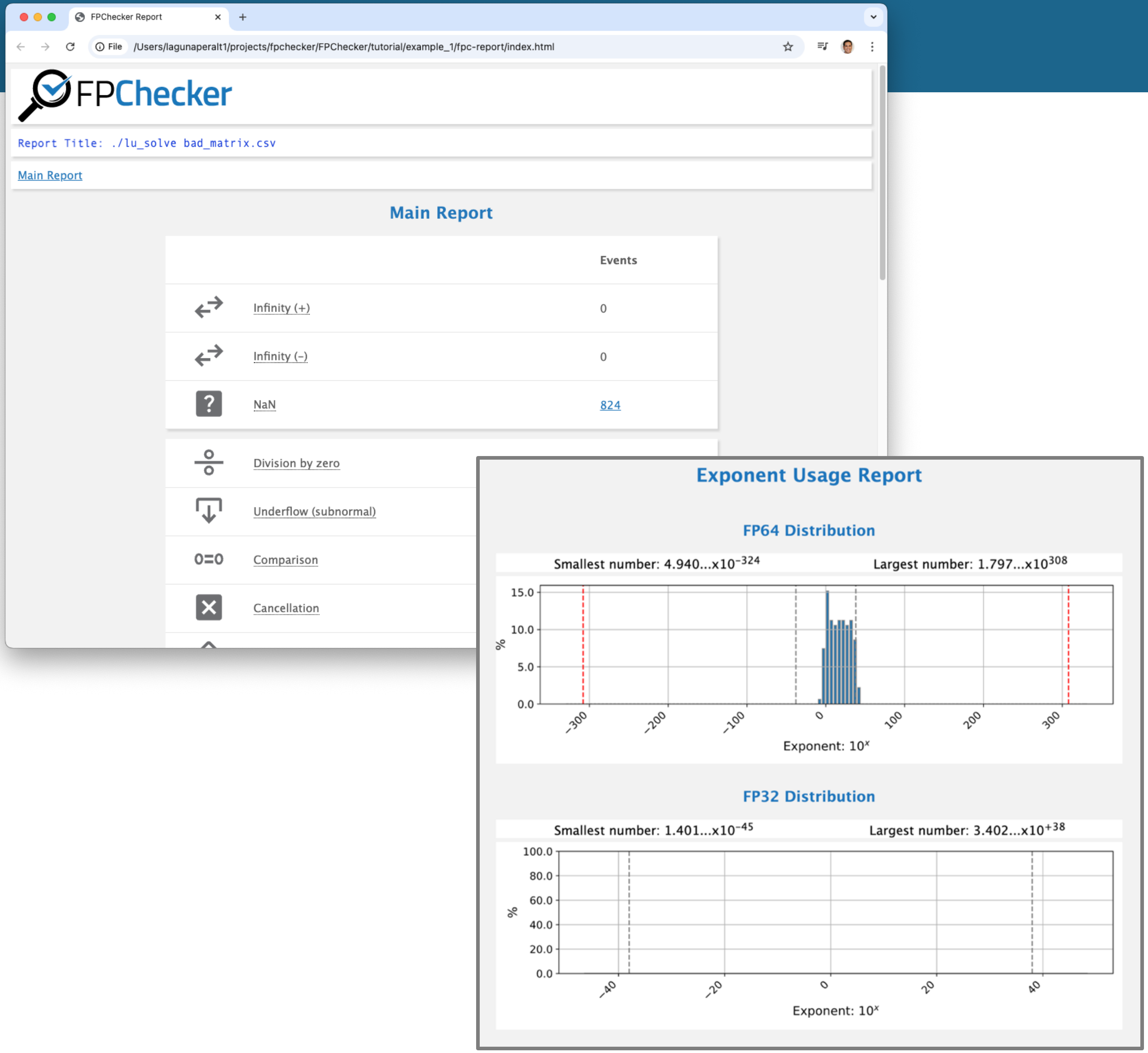
Lawrence Livermore National Laboratory

Agenda

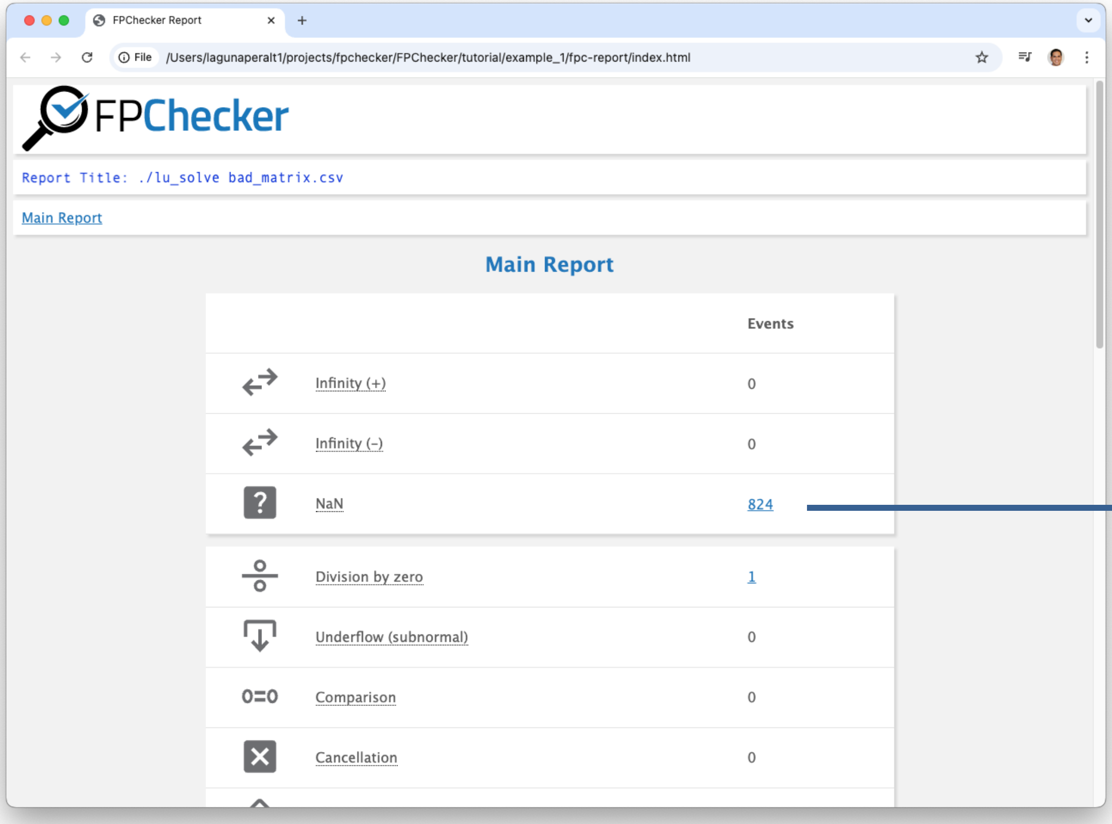
Intro	5 min
Tool's Overview	15 min
Installation & Exercises (hands-on)	35 min
Q&A	5 min

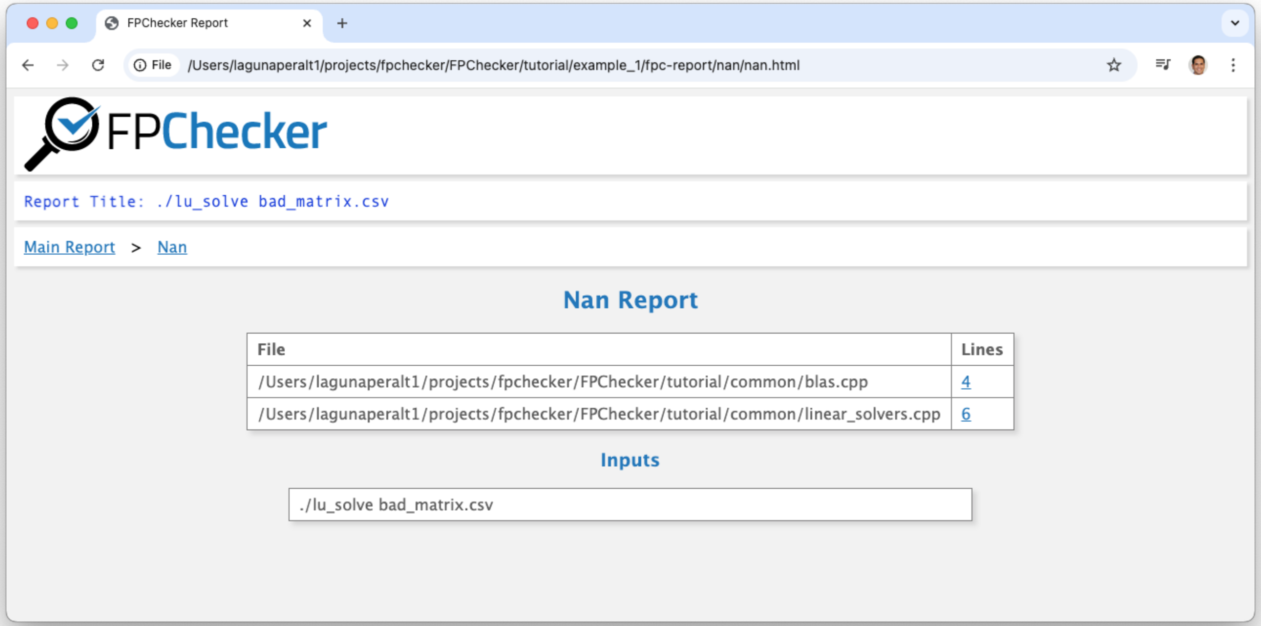
FPChecker Overview

- Detects floating-point exceptions
  - NaN, Infinity
- Shows impacted *lines of code*
- Shows other “code smells”
  - Cancellations, underflows
- Analyzes dynamic range
  - Is FP32 or FP64 enough?
- Works by compiler instrumentation
  - Relies on Clang/LLVM
- Documentation: https://fpchecker.org/

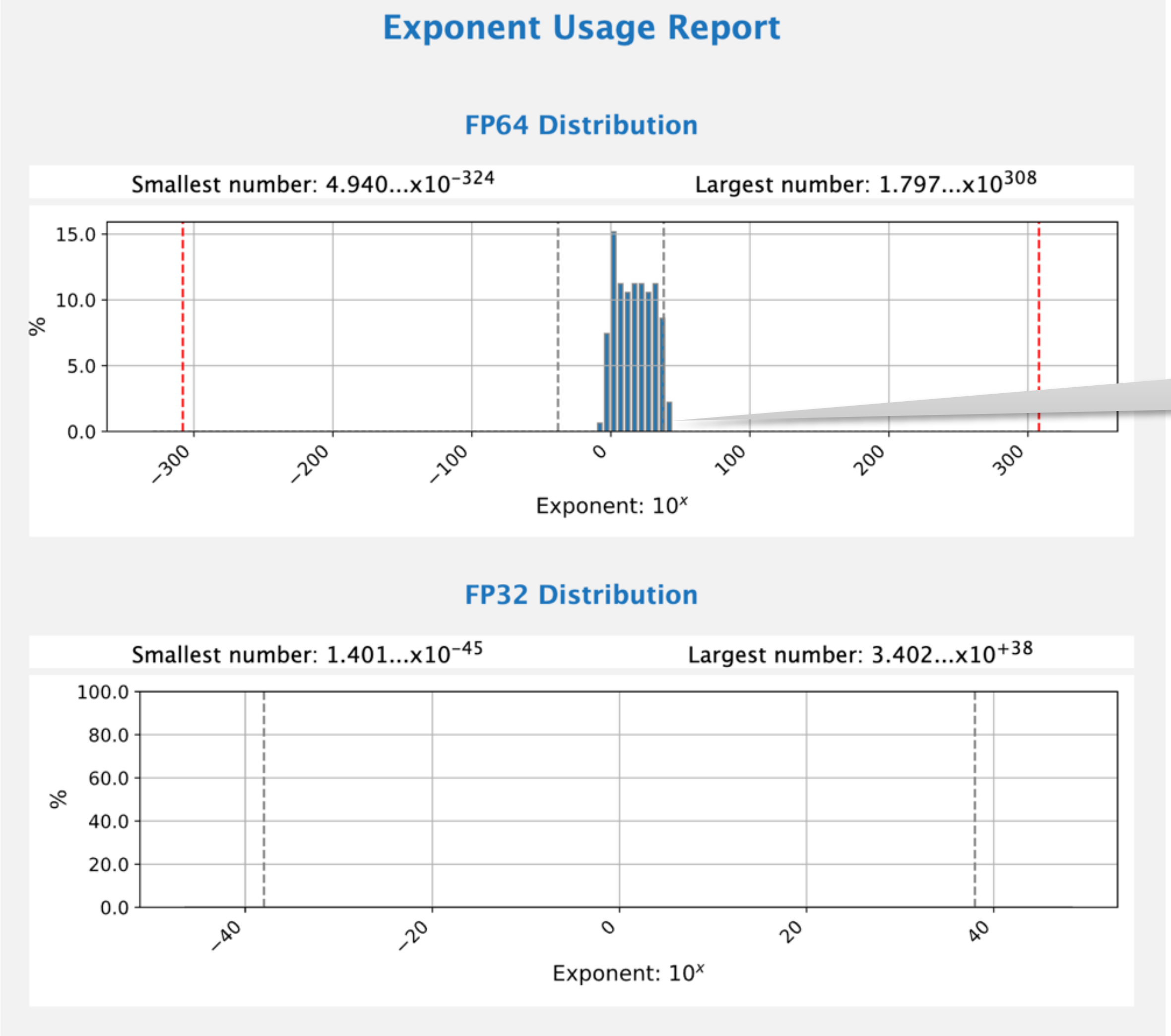


Report Example





Exponent Usage (Dynamic Range)



Out of range for FP32

LLVM Instrumentation Process

Source → Clang & LLVM IR → Optimizations (-O2) → Instrumentation → Executable

1. Use `clang++-fpchecker` wrapper in your Makefile
   - Or add instrumentation pass to your CFLAGS and/or CXXFLAGS
2. Enable `FPC_INSTRUMENT` environment variable when compiling

   ```
   $ FPC_INSTRUMENT=1 make
   ```
3. Run executable
4. Create report

Two Classes of Env Variables: *Compile-time & Run-time*

✓ Covered in the Tutorial

Compile-time Variables

	Variable	Type	Description
✓	FPC_INTRUMENT	Compile-time	Instruments the application
✓	FPC_ANNOTATED	Compile-time	Indicates that the program is annotated

Run-time Variables

	Variable	Type	Description
✓	FPC_EXPONENT_USAGE	Run-time	Profiles exponent usage for FP32/FP64
	FPC_TRAP_INFINITY_POS	Run-time	Program exits when Infinity positive is found
	FPC_TRAP_INFINITY_NEG	Run-time	Program exits when Infinity negative is found
	FPC_TRAP_NAN	Run-time	Program exits when NaN is found
	FPC_TRAP_DIVISION_ZERO	Run-time	Program exits when division-by-zero is found
	FPC_TRAP_CANCELLATION	Run-time	Program exits when cancellation is found
	FPC_TRAP_COMPARISON	Run-time	Program exits when Comparison is found
	FPC_TRAP_UNDERFLOW	Run-time	Program exits when underflow is found
	FPC_TRAP_LATENT_INF_POS	Run-time	Program exits when Latent Infinity positive is found
	FPC_TRAP_LATENT_INF_NEG	Run-time	Program exits when Latent Infinity negative is found
	FPC_TRAP_LATENT_UNDERFLOW	Run-time	Program exits when Latent Underflow is found

Software Requirements

- Linux or Mac OS
  - Windows not supported
- LLVM/Clang 19
- Cmake
- Python 3.12
- Matplotlib
- Optional for parallel code:
  - MPI
  - OpenMP

Installation Process

1. Install **Conda** (this allow us to install LLVM easily)
2. Install FPChecker

How to install Anaconda on Mac or Linux

For Mac, watch this YouTube video:

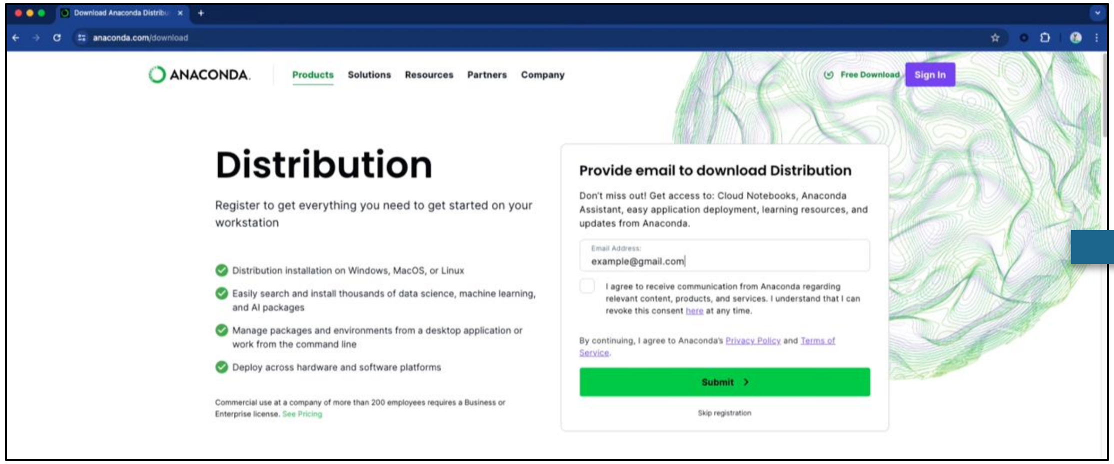
https://youtu.be/DNu8pQOYRGg

For Linux:

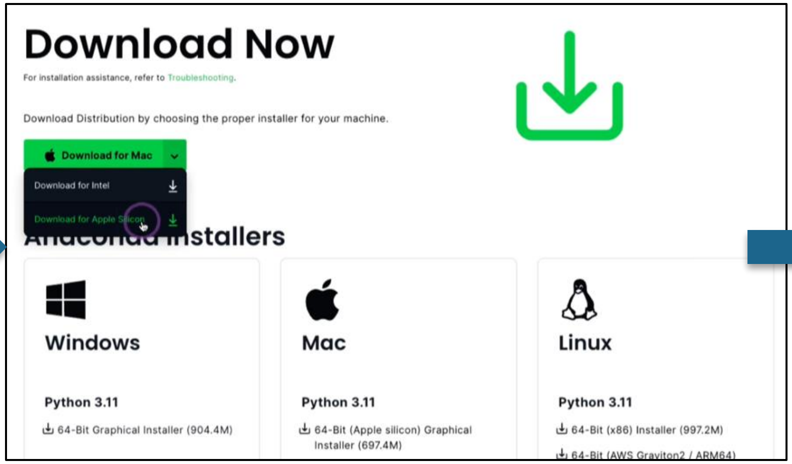
https://docs.conda.io/projects/conda/en/stable/user-guide/install/linux.html

Steps to Install Conda in Mac

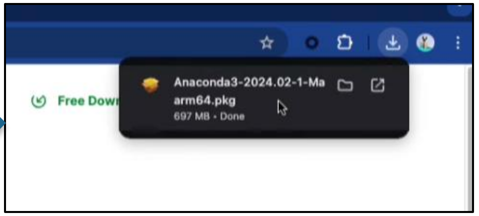
Got https://www.anaconda.com/ -> Download



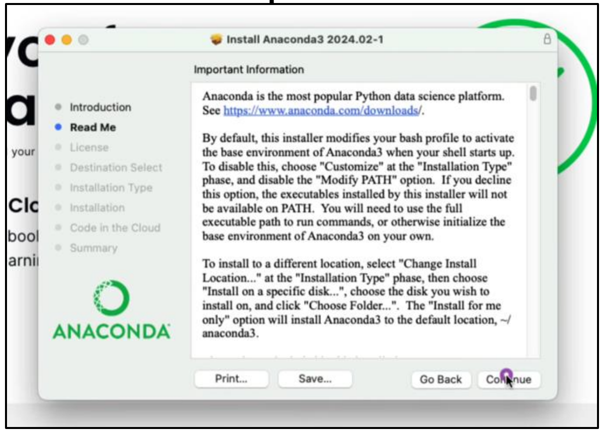
Download



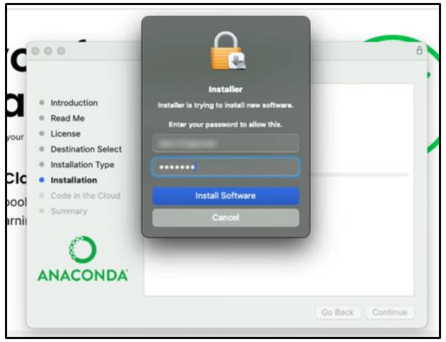
Open Installer



Follow the steps



Provide password if asked



In the terminal, you should see the word **base**:

```
(base) [user@system] $
```

Get FPChecker

This tutorial is based on release v0.5

```
   # We will install all in /tmp
$  cd /tmp
$  mkdir tutorial
$  cd tutorial

   # Clone FPChecker
$  git clone https://github.com/LLNL/FPChecker.git
$  cd FPChecker
```

Install FPChecker Dependencies with Conda (Option 1)
Manual Installation

```
   # Create a conda env
$  conda create --name tutorial_env
$  conda activate tutorial_env

   # Install dependencies: cmake, LLVM, clang++, python
$  conda install cmake
$  conda install llvmdev=19.1.7 -c conda-forge
$  conda install clangxx=19.1.7 -c conda-forge
$  conda install python=3.12.9

   # Install matplotlib. Not required to build FPChecker, but needed for reports
$  pip3 install matplotlib

   # MPI for some examples, but not required to build for FPChecker
$  conda install openmpi=5.0.7 -c conda-forge
```

Install FPChecker Dependencies with Conda (Option 2)
With environment file for Mac (ARM)

```
   # Create a conda env and dependencies
$  cd tutorial
$  conda create --name tutorial_env --file conda_environment.txt -c conda-forge
$  conda activate tutorial_env
```

- The `conda_environment.txt` file provides the packages for Mac (ARM 64-bit)
- This installs all the dependencies:
  - LLVM 19
  - Clang++ 19
  - Cmake

Install FPChecker

```
$ cd ..
$ mkdir build
$ cd build/
$ cmake -DCMAKE_INSTALL_PREFIX=../../install ..
$ make && make install

  # Export installation path
$ export PATH=/tmp/tutorial/install/bin:$PATH
```

```
# If the right python3 is not found, set the DPython3_ROOT_DIR
$ cmake -DCMAKE_INSTALL_PREFIX=../../install -DPython3_ROOT_DIR=/opt/anaconda3 ..
```

Tutorial Exercises

Tutorial Examples

	Topics	Example Program
1	NaN and Infinity exceptions	Linear system (Ax=b) solver with LU decomposition + partial pivoting
2	Exponent usage – from FP64 and FP32 precision	Finite differences + 1D Reaction-Diffusion PDE
3	Controlling slowdown with code annotations	Finite Elements + 2D Heat Conduction PDE solver
4	Analyzing parallel code: MPI/OpenMP	MPI-based Parallel Heat PDE solver

First, build the *common* library
It will be used in all examples

- Common library:
  - BLAS operations
  - Linear solvers (LU, CG)
  - Matrix & vector printing
  - Other functionalities

```
   # Compile library
$  cd /tmp/tutorial/FPChecker/tutorial/common/
$  FPC_INSTRUMENT=1 make
```

Example 1:
NaN & Infinity exception in linear solver

- Location:

  `tutorial/example_1/lu_solve.cpp`

- Program description
  - Linear solver $Ax = b$
  - Solves $Ax = 1$
  - LU decomposition: $PA = LU$
  - Partial pivoting
  - Solve by forward/backward substitution

FPChecker Use Case

- Use an ill-condition problem (matrix)
- Produces NaN and Infinity
  - U factor ends up with zero diagonal
  - Division by zero
- FPChecker locates the exceptions

Example 1: Script (Good Matrix)

```
   # Compile and run problem
$ FPC_INSTRUMENT=1 make
$ ./lu_solve matrix.csv

   # List trace files (there is one)
$ ls -l .fpc_logs/

   # Create report
$ fpc-create-report -t "./lu_solve matrix.csv"
$ open fpc-report/index.html

   # Clear logs and remove report
$ fpc-create-report -rc
```

Nothing interesting in the report

Example 1: Script (Bad Matrix)

```
  # Run problem
$ ./lu_solve bad_matrix.csv

  # Create report
$ fpc-create-report -t "./lu_solve bad_matrix.csv"
$ open fpc-report/index.html
```

- NaN
- Division by zero

Report of Example 1

FPChecker Report

File /Users/lagunaperalt1/projects/fpchecker/FPChecker/tutorial/example_1/fpc-report/index.html

FPChecker

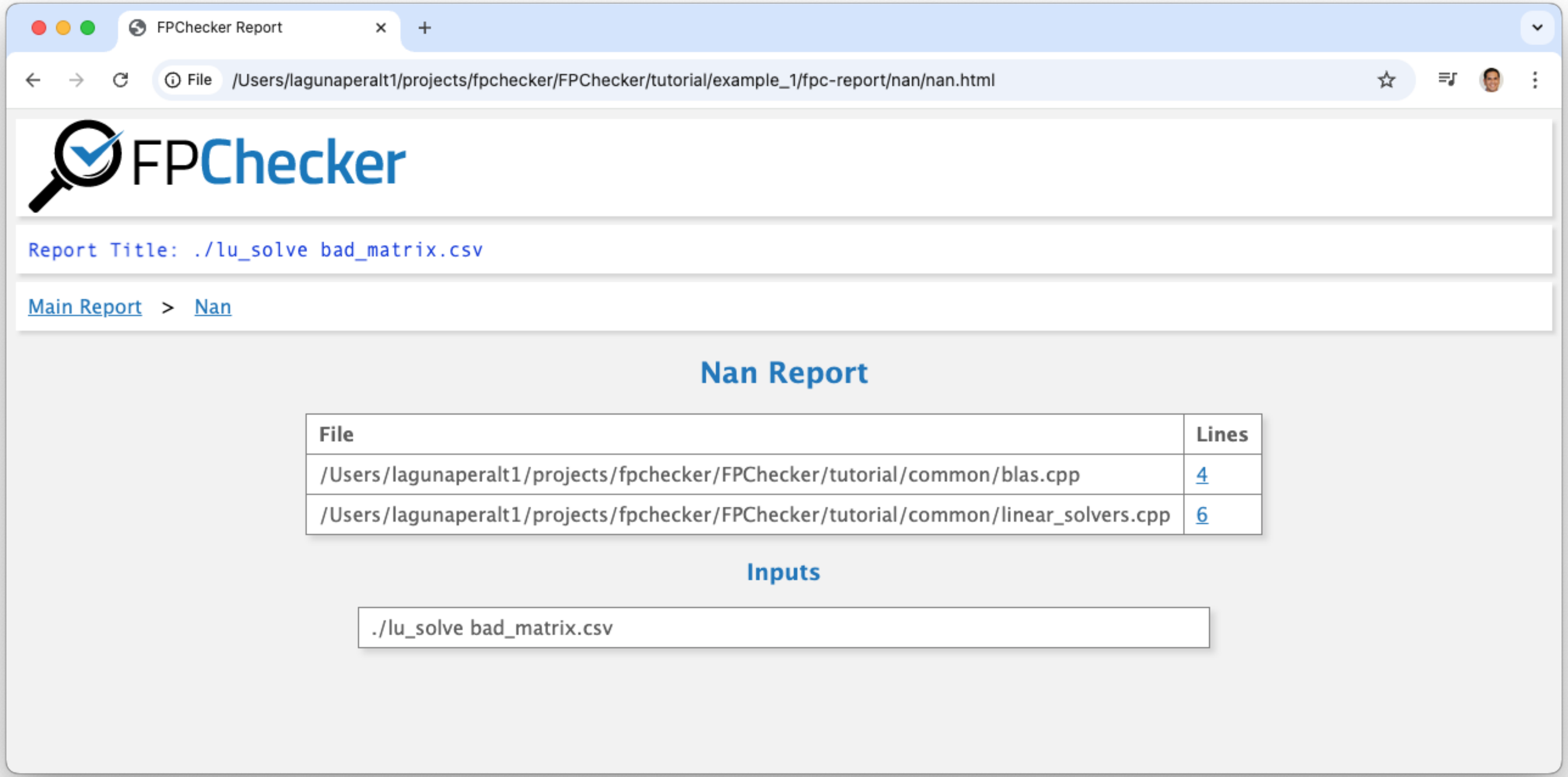
Report Title: ./lu_solve bad_matrix.csv

Main Report

Main Report

	Events
Infinity (+)	0
Infinity (-)	0
NaN	824
Division by zero	1
Underflow (subnormal)	0
Comparison	0
Cancellation	0

Report of Example 1



Tutorial Examples

	Topics	Example Program
1	NaN and Infinity exceptions	Linear system (Ax=b) solver with LU decomposition + partial pivoting
2	Exponent usage – from FP64 and FP32 precision	Finite differences + 1D Reaction-Diffusion PDE
3	Controlling slowdown with code annotations	Finite Elements + 2D Heat Conduction PDE solver
4	Analyzing parallel code: MPI/OpenMP	MPI-based Parallel Heat PDE solver

Example 2:
Exponent Usage on FP64-to-FP32 porting

- Location:

`tutorial/example_2/reaction_diffusion.cpp`

- Program description
  - 1D linear **reaction-diffusion equation**
  - $PDE: \dfrac{\partial u}{\partial t} = D\dfrac{\partial^2 u}{\partial x^2} + \lambda u$
  - Explicit finite difference method
    - Forward Euler in time, Central Difference in space
  - Large λ provides positive feedback
    - Over time, it leads to **exponential growth**

- Code parameters:

$$D = 0.01$$

$$\lambda = 25$$

FPChecker Use Case

- Run simulation in FP64 and FP32
- Vizualize exponent usage
- In FP32, values are ”out-of-range”
  - Produce exceptions

Example 2: Script

FP64 Version

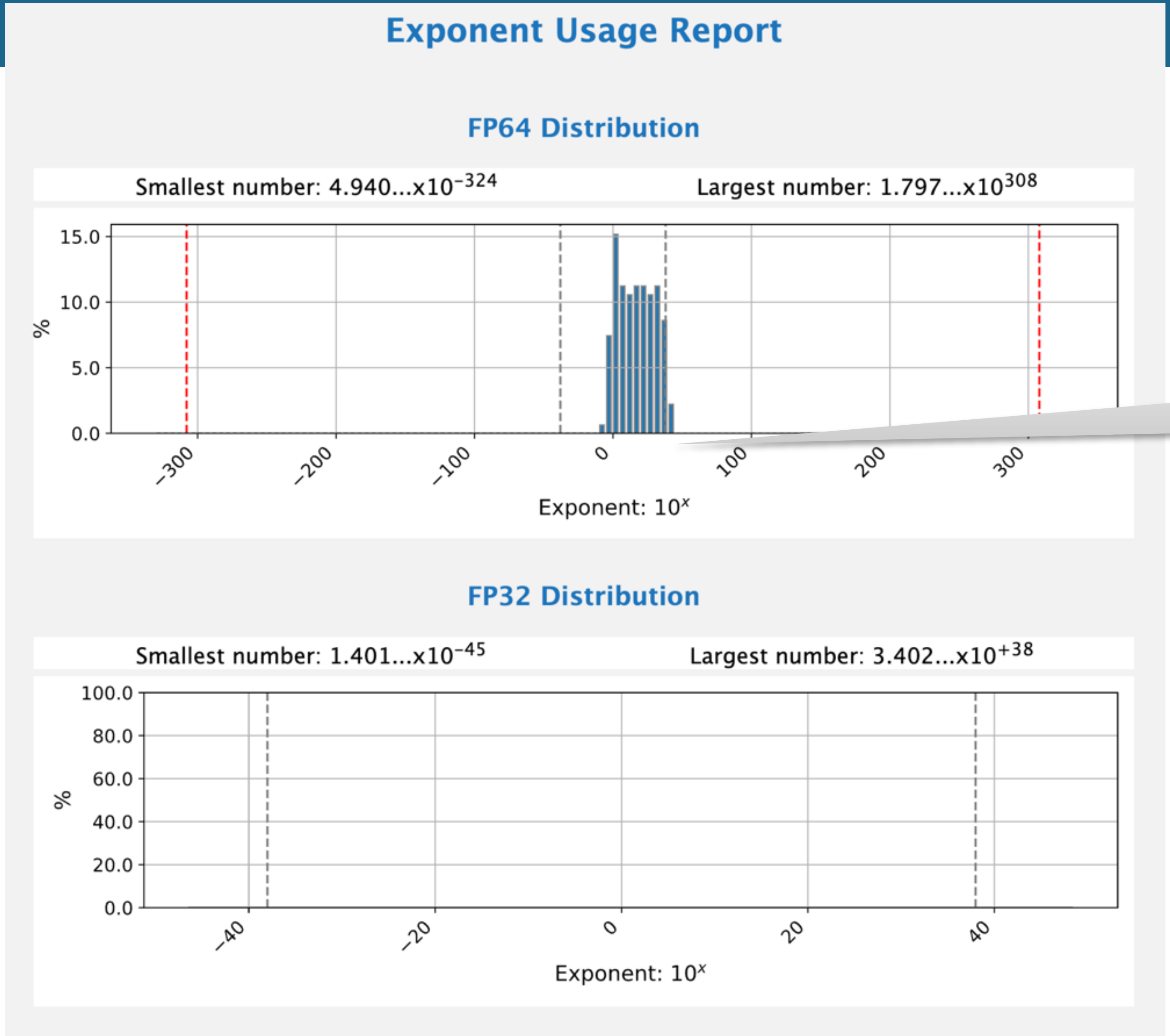
```
  # Compile and run problem
$ FPC_INSTRUMENT=1 make
$ FPC_EXPONENT_USAGE=1 ./reaction_diffusion

  # List trace files (there are two)
$ ls -l .fpc_logs/

  # Create report
$ fpc-create-report
$ open fpc-report/index.html

  # Clear logs and remove report
$ fpc-create-report -rc
```

Report (Example 1, FP64)



Out of range for FP32

Example 2: Script

FP32 Version

<u>First Step:</u>

- Open `reaction_diffusion.cpp`
- Modify lines 7-8 to use FP32

```
typedef double Real_t;
// typedef float Real_t;
```

```
   # Compile and run problem
$  FPC_INSTRUMENT=1 make
$  FPC_EXPONENT_USAGE=1 ./reaction_diffusion

   # List traces files (there are two)
$  ls -l .fpc_logs/

   # Create report
$  fpc-create-report
$  open fpc-report/index.html

   # Clear logs and remove report
$  fpc-create-report -rc
```

Report (Example 2, FP32)

FPChecker Report

File /Users/lagunaperalt1/projects/fpchecker/FPChecker/tutorial/example_2/fpc-report/index.html

FPChecker

Report Title:

Main Report

Main Report

	Events
Infinity (+)	15
Infinity (-)	3
NaN	155658
Division by zero	0
Underflow (subnormal)	0
Comparison	0
Cancellation	0

Tutorial Examples

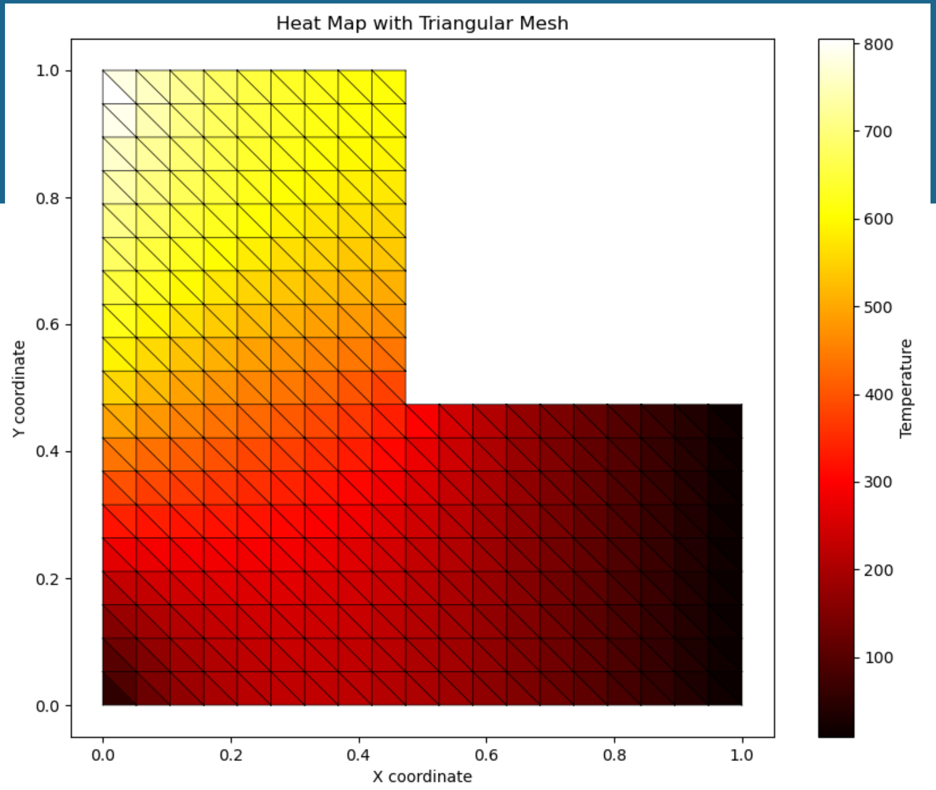
Topics	Example Program
1 NaN and Infinity exceptions	Linear system (Ax=b) solver with LU decomposition + partial pivoting
2 Exponent usage – from FP64 and FP32 precision	Finite differences + 1D Reaction-Diffusion PDE
3 Controlling slowdown with code annotations	Finite Elements + 2D Heat Conduction PDE solver
4 Analyzing parallel code: MPI/OpenMP	MPI-based Parallel Heat PDE solver

Example 3:
Annotations to Control Slowdown

- Location:

  `tutorial/example_3/heat_PDE_finite_elements.cpp`

- Program description
  - **2D Heat conduction equation** with a source term
    - Steady state
  - PDE: $\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = s(x,y)$
  - Finite elements method
    - Triangular shapes
  - $T(x,y)$: temperature distribution
  - Domain: L shape
  - Source $s(x,y) = 0$
  - Boundary conditions: temp applied on the sides



FPChecker Use Case

- Slowdown can be high (for a large problem)
- Reduce slowdown by annotating the code

Example 3: Script

- Run small problem (10 nodes)
- Should take less than 1 second

```
   # Compile and run problem
$  FPC_INSTRUMENT=1 make
$  time FPC_EXPONENT_USAGE=1
$  ./heat_PDE_finite_elements 10

   # Optional: Vizualize heat map
$  python3 plot.py
```

- Run a larger problem (50 nodes)
- It takes about 20 seconds in my laptop

```
   # Compile and run problem
$  time FPC_EXPONENT_USAGE=1 ./heat_PDE_finite_elements 50
```

Code Annotations

- Let's annotate the matrix-multiply function
  - That is: we only instrument and analyze that function
  - Should reduce overhead significantly

- Location:

  `tutorial/common/blas.cpp`

- Search for:

  `vector<vector<double>>` **`matrix_multiply`**`(…`

- Add (or uncomment):

  `FPC_INSTRUMENT_FUNC`

blas.cpp

blas.cpp ⟩ No Selection

```cpp
// Matrix multiplication
FPC_INSTRUMENT_FUNC
vector<vector<double>> matrix_multiply(const vector<vector<double>> &A, const
    vector<vector<double>> &B)
{
    int rows_A = A.size();
    if (rows_A == 0)
    {
        return {};
    }
    int cols_A = A[0].size();

    int rows_B = B.size();
    if (rows_B == 0)
    {
        return {};
    }
    int cols_B = B[0].size();

    if (cols_A != rows_B)
    {
        cerr << "Error: Number of columns must be equal to the number of rows for
            multiplication." << endl;
        return {};
    }

    vector<vector<double>> result(rows_A, vector<double>(cols_B, 0.0));
```

Line: 1 Col: 1

Example 3: Script

Recompile the common library

```
$  cd ../common/
$  make clean
$  FPC_INSTRUMENT=1 FPC_ANNOTATED=1 make

$  cd ../example_3/
$  FPC_INSTRUMENT=1 FPC_ANNOTATED=1 make
$  time FPC_EXPONENT_USAGE=1 ./heat_PDE_finite_elements 50
   ...
   ...
   real    0m1.049s
   user    0m0.729s
   sys     0m0.071s
```

Lower run time

Tutorial Examples

	Topics	Example Program
1	NaN and Infinity exceptions	Linear system (Ax=b) solver with LU decomposition + partial pivoting
2	Exponent usage – from FP64 and FP32 precision	Finite differences + 1D Reaction-Diffusion PDE
3	Controlling slowdown with code annotations	Finite Elements + 2D Heat Conduction PDE solver
4	Analyzing parallel code: MPI/OpenMP	MPI-based Parallel Heat PDE solver

Example 4:
Analyzing MPI code

- Location:

  `tutorial/example_4/heat_mpi.cpp`

- Program description
  - 1D heat equation
  - $PDE: \ \frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$
  - Explicit finite difference method
- MPI domain decomposition:
  - 1D spatial grid divided into NPROC contiguous segments
  - NPROC: number of MPI processes
  - Each process computes temperature in its segment

FPChecker Use Case

- Generates traces for MPI programs
- Combine traces into a single report

Example 4: Script

```
  # Show Makefile uses CXX = mpic++-fpchecker
  # Compile and run problem
  # OMPI_CXX indicates to Open MPI which conda compiler to use
$ OMPI_CXX=clang++ FPC_INSTRUMENT=1 make
$ FPC_EXPONENT_USAGE=1 mpiexec -n 4 ./heat_mpi

  # List trace files (there are 4)
$ ls .fpc_logs/
  fpc_king01_95250.json fpc_king01_95251.json
  fpc_king01_95252.json fpc_king01_95253.json

  # Create report
$ fpc-create-report
$ open fpc-report/index.html
```

Showing the Configuration of the Installation

```
$ fpchecker-show
========================================
         FPChecker Configuration
========================================

Installation path: /tmp/tutorial/install

Add this to CFLAGS and/or CXXFLAGS:
-g -include /tmp/tutorial/install/src/Runtime_cpu.h -fpass-plugin=/tmp/tutorial/install/lib/libfpchecker_cpu.dylib

Wrappers are located here:
/tmp/tutorial/install/bin/clang-fpchecker
/tmp/tutorial/install/bin/clang++-fpchecker
/tmp/tutorial/install/bin/mpicc-fpchecker
/tmp/tutorial/install/bin/mpicxx-fpchecker
```

Conda Commands:

- Removing an environment:

```
conda env remove --name <environment_name>
```

- Adding an environment:

```
conda env --name <environment_name>
```

- List environments:

```
conda env list
```

Demo FPChecker in Some Packages

Package		Build Model
SuperLU	Linear solver	C, cmake
Hypre	Linear solver	C, cmake, MPI
LAMMPS	Molecular dynamics	C, C++, cmake
FFTW	Fourier transform	C, autotools

Example: SuperLU

```
$  git clone git@github.com:xiaoyeli/superlu.git
$  git clone https://github.com/xiaoyeli/superlu.git
$  cd superlu
$  git checkout 4ef39075e029927e8c959b22c8e7052dcb40c995
$  mkdir build
$  cd build
$  CC=clang-fpchecker cmake -DCMAKE_INSTALL_PREFIX=./install -Denable_internal_blaslib=ON ..
$  FPC_INSTRUMENT=1 make -j
   make install
```

- Configure to build internal BLAS
  - There seems to be an error with fpchecker-clang and Cmake finding BLAS in Mac

Example: FFTW

```
$ wget https://www.fftw.org/fftw-3.3.10.tar.gz
$ tar -zxvf fftw-3.3.10.tar.gz
$ cd fftw-3.3.10
$ CC=clang-fpchecker ./configure --disable-fortran --prefix=/tmp/tutorial/examples/fftw/fftw-3.3.10/fftw-install
$ FPC_INSTRUMENT=1 make -j
$ make install
```

- See example at:
  - FPChecker/tutorial/other_examples/fftw:
    - fftw_test.c

Example: LAMMPS

```
$  wget https://github.com/lammps/lammps/archive/refs/tags/stable_29Aug2024_update2.tar.gz
$  tar -xvf stable_29Aug2024_update2.tar.gz
$  cd lammps-stable_29Aug2024_update2/
$  cd cmake/
$  mkdir build
$  cd build
$  CC=clang CXX=clang++ cmake \
   -DCMAKE_CXX_FLAGS="-include /tmp/tutorial/install/src/Runtime_cpu.h -fpass-
   plugin=/tmp/tutorial/install/lib/libfpchecker_cpu.dylib -g" \
   -DBUILD_MPI=OFF -DBUILD_OMP=OFF -DBUILD_FORTRAN=OFF \
   -DBUILD_SHARED_LIBS=OFF -DENABLE_TESTING=OFF ..
$  FPC_INSTRUMENT=1 make -j
$  ./lmp -in ../../examples/melt/in.melt
```

Example: Hypre

```
$  git clone git@github.com:hypre-space/hypre.git
$  cd hypre
$  git checkout be52325a3ed8923fb93af348b1262ecfe44ab5d2
$  cd src
$  mkdir build
$  cd build
$  CC=clang cmake -DHYPRE_ENABLE_MPI=ON \
   -DHYPRE_WITH_EXTRA_CFLAGS="-include /tmp/tutorial/install/src/Runtime_cpu.h -fpass-
   plugin=/tmp/tutorial/install/lib/libfpchecker_cpu.dylib -g" ..
$  FPC_INTRUMENT=1 make –j
$  make install
```

- HYPRE is installed in:
  - `/tmp/tutorial/examples/hypre/src/hypre`
- `Test in tutorial/other_examples/hypre:`
  - `To compile example:`

    ```
    FPC_INSTRUMENT=1 OMPI_CC=clang make
    HYPRE_MATRIX=matrix.csv ./hypre_test 0
    ```

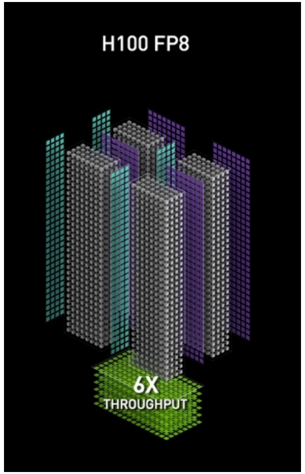
Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- **09:35 : NixNan (Part-1)**
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

NixNan : Detecting Exceptions in NVIDIA GPUs

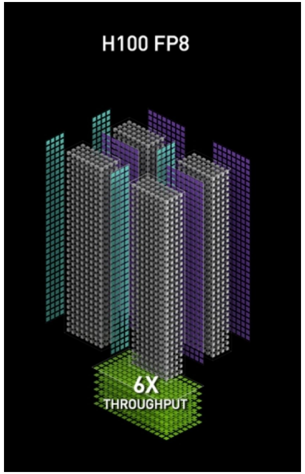
Detects exceptions in

SIMT Cores as well as Tensor Cores



NixNan : Detecting Exceptions in NVIDIA GPUs

Detects exceptions in
SIMT Cores as well as Tensor Cores



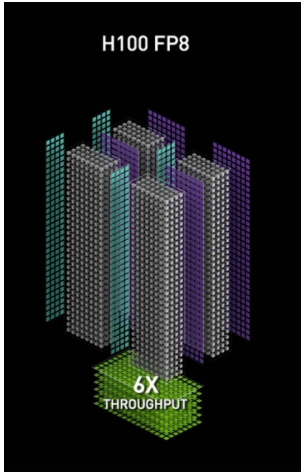
- NVIDIA GPUs do not generate any traps when an FP exception occurs
- Thus, we have to detect exceptions in software
    - By decoding the exponent and mantissa of the computed answer

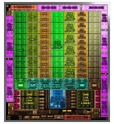
S	E	M

FF	INF (if 0) or NaN (if not)
0	non-zero => subnormal

NixNan Detects Exceptions in NVIDIA GPUs **per Kernel Launch**

Detects exceptions in
SIMT Cores as well as Tensor Cores





> LD_PRELOAD = nixnan.so
Pre-Compiled Application Binary

Intercept all "Kernel Launch" CUDA Driver API Calls

For Each FP instruction :

Inject_instrumentation();

CUDA Driver APIs

```
#nixnan: ------------ nixnan Report

#nixnan: --- FP16 Operations ---
#nixnan: NaN: 4
#nixnan: Infinity: 3
#nixnan: Subnormal: 2
#nixnan: Division by 0: 0

#nixnan: --- FP32 Operations ---
#nixnan: NaN: 0
#nixnan: Infinity: 0
#nixnan: Subnormal: 0
#nixnan: Division by 0: 0

#nixnan: --- FP64 Operations ---
#nixnan: NaN: 0
#nixnan: Infinity: 0
#nixnan: Subnormal: 0
#nixnan: Division by 0: 0
```

```
----- Test NaN: A[0,0]=inf, B[0,0]=1, C[0,0]=-inf -----
A[0,0] = inf (0x7c00), B[0,0] = 1.000000 (0x3c00), C[0,0] = -inf (0xfc00)
Computing D = A * B + C with Tensor Cores...
#nixnan: error [NaN,infinity] detected in instruction HMMA.16816.F16 R20, R4.reuse, R16, RZ ;
#nixnan: error [NaN] detected in instruction HMMA.16816.F16 R22, R4, R18, RZ ; in function WMM
D[0,0]=nan (0x7fff)
```

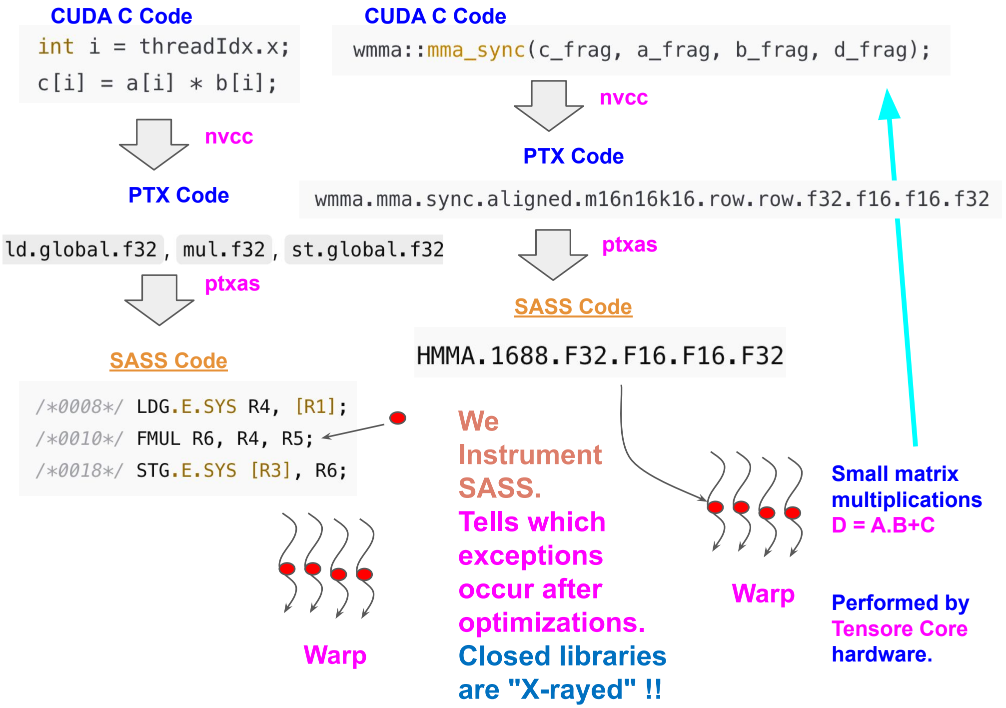
History: GPU-FPX ⟶ Evolved to become NixNan

First demonstrated during HPDC 2023

Has a detector and an analyzer component that does flow analysis

Exits early when a thread detects exceptions

No support for FP16 or Tensor Cores



Continuing to refine it in response to tests (discovering missing cases each time…)

Has a single component that detects as well as flow-analyzes

Finds all exceptions (without quitting early)

Support for FP16 and Tensor Cores

Real-world Issue: NixNan usage

- Link: https://github.com/asappresearch/sru/issues/193

hoagy-davis-digges commented on Nov 2, 2021 • edited

I have run the example code in the readme on both 2.6.0 and 3.0.0-dev and both have nan values in both the output and state objects using pytorch 1.9, I've tried this on my computer using a Titan X and also with a fresh install on a cloud T4, this doesn't seem to relate to the other nan issue raised here https://github.com/asappresearch/sru/issues/185 because this problem appears immediately.

This case-study was done by Dr. Xinyi Li who developed GPU-FPX, the precursor to NixNan

(the latter is a refined version with bug-fixes, and Tensor-Core Support)

We have adapted Dr. Li's test and fix to NixNan.

Issue

- Link: https://github.com/asappresearch/sru/issues/193

taolei87 commented on Nov 5, 2021 Contributor ...

hi @hoagy-davis-digges , did you mean you tried the following example and got NaN?

```
import torch
from sru import SRU, SRUCell

# input has length 20, batch size 32 and dimension 128
x = torch.FloatTensor(20, 32, 128).cuda()

input_size, hidden_size = 128, 128

rnn = SRU(input_size, hidden_size,
    num_layers = 2,          # number of stacking RNN layers
    dropout = 0.0,           # dropout applied between RNN layers
    bidirectional = False,   # bidirectional RNN
    layer_norm = False,      # apply layer normalization on the output of each layer
    highway_bias = -2,        # initial bias of highway gate (<= 0)
)
rnn.cuda()

output_states, c_states = rnn(x)      # forward pass
```

Tool Usage

```
1999  python run_sru.py
2000  LD_PRELOAD=~/nixnan.so python run_sru.py
2001  LD_PRELOAD=~/nixnan.so python run_sru_fixed.py
```

Seeing NaNs early, we examined the source-code…

```
#nixnan: running kernel [void at::native::vectorized_elementwise_kernel] ...
#nixnan: running kernel [ampere_sgemm_32x128_nn] ...
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R1, R32.reu
se, R40.reuse, R1 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 0 of instruction FFMA R0, R32, R4
1.reuse, R0 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R10, R34, R
43.reuse, R10 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R5, R33.reu
se, R40.reuse, R5 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R3, R32.reu
se, R42, R3 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 0 of instruction FFMA R4, R33, R4
1.reuse, R4 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R0, R32, R4
1.reuse, R0 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 0 of instruction FFMA R1, R36.reu
se, R44.reuse, R1 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 3 of instruction FFMA R8, R38.reu
se, R45.reuse, R8 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [NaN,subnormal] detected in operand 0 of instruction FFMA R5, R37
.reuse, R44.reuse, R5 ; in function ampere_sgemm_32x128_nn of type f32
```

Found uninitialized the tensor; setting that fixes NaNs

taolei87 commented on Nov 5, 2021 · Contributor · ···

hi @hoagy-davis-digges , did you mean you tried the following example and got NaN?

```
import torch
from sru import SRU, SRUCell

# input has length 20, batch size 32 and dimension 128
x = torch.FloatTensor(20, 32, 128).cuda()

input_size, hidden_size = 128, 128

rnn = SRU(input_size, hidden_size,
    num_layers = 2,          # number of stacking RNN layers
    dropout = 0.0,           # dropout applied between RNN layers
    bidirectional = False,   # bidirectional RNN
    layer_norm = False,      # apply layer normalization on the output of each layer
    highway_bias = -2,        # initial bias of highway gate (<= 0)
)
rnn.cuda()

output_states, c_states = rnn(x)      # forward pass
```

torch.randn(20,32,128).cuda()

Before Fixing SRU

```
#nixnan: --- FP16 Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: Infinity:                   0 (0 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                  0 (0 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: Infinity:                   0 (0 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                  0 (0 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                      843 (2418829 repeats)
#nixnan: Infinity:                  13 (15635 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                746 (251293 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: Infinity:                   0 (0 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                  0 (0 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                        3 (5373 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                        0 (0 repeats)
```

Exceptions Written Into Memory

Our fix was applied

After Fixing SRU

```
#nixnan: --- FP16 Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: Infinity:                   0 (0 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                  0 (0 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: Infinity:                   0 (0 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                  0 (0 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: Infinity:                   0 (0 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                  0 (0 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: Infinity:                   0 (0 repeats)
#nixnan: -Infinity:                  0 (0 repeats)
#nixnan: Subnormal:                  0 (0 repeats)
#nixnan: Division by 0:              0 (0 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                        0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                        0 (0 repeats)
```

Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- **09:55 : Plz Fill Survey**
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- **10:00 : Coffee Break**
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

Agenda + Single QR Code for those coming post-Coffee

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- **10:30 : Recap of First Half**
- **10:35 : NixNan (Part-2)**
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

SC 25 Floating-Point Tutorial : All Tools and Benchmarks

fpanalysistools.org

Demo of NixNan : HPC

- **We show two cases of solving a linear system (next slide)**
    - Matrix with a low condition number : **matrix_0.csv**
    - Matrix with a high condition number: **matrix_5.csv**

FP Exceptions in HPC : Ill-conditioned Matrix

```
#nixnan: ------------ nixnan Report -----------

#nixnan: --- FP16 Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: Infinity:                0 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: Infinity:                0 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: Infinity:                0 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: Infinity:                0 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           0 (0 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
```

```
--- Solution X Vector (10)
1.12338
0.799707
7.82785
-1954.76
-0.000778511
1.07278
-0.535868
-3.61902
8.83203
0.585185
```

```
LD_PRELOAD=~/nixnan.so
CUDA_MATRIX=matrix_0.csv ./LU_solver
```

```
LD_PRELOAD=~/nixnan.so
CUDA_MATRIX=matrix_5.csv ./LU_solver
```

```
--- Solution X Vector (10)
1.4665
inf
2.15945
0.895458
10.9171
1.11917
inf
24.0914
0.421027
-1.39342
```

```
#nixnan: ------------ nixnan Report -----------

#nixnan: --- FP16 Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: Infinity:                0 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: Infinity:                0 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                     4 (0 repeats)
#nixnan: Infinity:                0 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                    48 (0 repeats)
#nixnan: Infinity:                2 (0 repeats)
#nixnan: -Infinity:               0 (0 repeats)
#nixnan: Subnormal:               0 (0 repeats)
#nixnan: Division by 0:           2 (2 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                     0 (0 repeats)
```

Demo of NixNan : ML

- A simple GPT (due to Karpathy)
  - Observed many exceptions
    - They do not seem to affect the loss
- Also ran a medical agent (BioMistral)
  - Also observed many exceptions
    - Also does not affect the loss

Innocuous (?) FP Exceptions in BioMistral

```
ganesh@beast:~/repos/newmed$ source venv/bin/activate
(venv) ganesh@beast:~/repos/newmed$ python simple_medical_agent.py
============================================================
Medical Dialog Agent - Initializing...
============================================================
Model: BioMistral/BioMistral-7B
4-bit Quantization: True

Device: cuda
Loading tokenizer...
Loading model with 4-bit quantization (low VRAM mode)...
✓ Model loaded successfully!


============================================================
          MEDICAL DIALOG AGENT
============================================================

⚠️ DISCLAIMER: This is for educational/research purposes only.
   NOT a substitute for professional medical advice.

Commands:
  - Type your symptoms or questions
  - 'clear' - Clear conversation history
  - 'save' - Save conversation to file
  - 'quit' or 'exit' - Exit the program

============================================================

You: dizzy

Agent: what did you do to yourself? You're bleeding.

You:
```

LD_PRELOAD=~/nixnan.so
python simple_medical_agent.py

In case of ML routines, a huge number of NaNs and INF are generated. Some do leak into memory, but most seem to not.

```
#nixnan: error [NaN,subnormal] detected in operand 1 of instruction FADD R11, R14, R13 ; in function void pytorch_flash::flash_fwd_kernel of type f32
#nixnan: error [NaN,subnormal] detected in operand 2 of instruction FADD R11, R14, R13 ; in function void pytorch_flash::flash_fwd_kernel of type f32
#nixnan: error [subnormal] detected in operand 1 of instruction HADD2 R5, -R4.H0_H0, -RZ.H0_H0 ; in function void at::native::elementwise_kernel of type f16
#nixnan: error [subnormal] detected in operand 0 of instruction HADD2 R5, -R4.H0_H0, -RZ.H0_H0 ; in function void at::native::elementwise_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R44.reuse, R28, RZ ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R72.reuse, R48, R32 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [subnormal] detected in operand 2 of instruction HMMA.16816.F32 R40, R56.reuse, R60, R40 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R56.reuse, R44, R32 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R12.reuse, R72, R32 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [subnormal] detected in operand 2 of instruction HMMA.16816.F32 R40, R52.reuse, R68, R40 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [subnormal] detected in operand 0 of instruction @!P1 FMUL R103, R103, R103 ; in function void pytorch_flash::flash_fwd_kernel of type f32
#nixnan: error [NaN,subnormal] detected in operand 1 of instruction FADD R103, R103, R106 ; in function void pytorch_flash::flash_fwd_kernel of type f32
```

```
#nixnan: ------------ nixnan Report -----------
#nixnan: --- FP16 Operations ---
#nixnan: NaN:            1000 (10890400 repeats)
#nixnan: Infinity:        253 (8631 repeats)
#nixnan: -Infinity:       119 (8037 repeats)
#nixnan: Subnormal:      1195 (1868286503 repeats)
#nixnan: Division by 0:     0 (0 repeats)
#nixnan: --- BF16 Operations ---
#nixnan: NaN:               0 (0 repeats)
#nixnan: Infinity:          0 (0 repeats)
#nixnan: -Infinity:         0 (0 repeats)
#nixnan: Subnormal:         0 (0 repeats)
#nixnan: Division by 0:     0 (0 repeats)
#nixnan: --- FP32 Operations ---
#nixnan: NaN:            2550 (79727147 repeats)
#nixnan: Infinity:        548 (75451 repeats)
#nixnan: -Infinity:      1102 (2786872 repeats)
#nixnan: Subnormal:       263 (20232 repeats)
#nixnan: Division by 0:     0 (0 repeats)
#nixnan: --- FP64 Operations ---
#nixnan: NaN:               0 (0 repeats)
#nixnan: Infinity:          0 (0 repeats)
#nixnan: -Infinity:         0 (0 repeats)
#nixnan: Subnormal:         0 (0 repeats)
#nixnan: Division by 0:     0 (0 repeats)
#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:               0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:               0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:               0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:               0 (0 repeats)
```

One NixNan Use: Vary Model Parameters; then Observe Exception Diffs

Experimental Variation Studied (Claude-created)	What It Changes (Claude)	Expected NaN Reduction (Claude)	BF16 (as observed)	FP32 (as observed)
baseline	Nothing (reference)	0%		NaN: 4592 (22306080) inf: 1010 (666054) -inf: 1616 (9361693)
bfloat16	Compute dtype FP16→BF16	70-90%	NaN: 1298 (11852994) inf: 396 (189380) -inf: 586 (918390) subn: 1124 (11440312)	
eager	Disable Flash Attention	60-80%		NaN: 1336 (191306408) inf: 744 (863768) -inf: 824 (1226989)
layernqorm_eps	LayerNorm epsilon 1e-5→1e-3	20-40%		NaN: 4759 (218503344) inf: 1129 (730696) -inf: 1799 (9400170) subn: 1237 (42303)
attention_scale	Scale attention scores	30-50%	TBD	TBD
attention_clip	Clip attention logits	40-60%		Nan: 458 (4623) -inf: 248 (28452)

In ML, Frameworks such as PyTorch Silently "Fix" Exceptions, but this can be Slow / Imperfect (believable ChatGPT summary below)

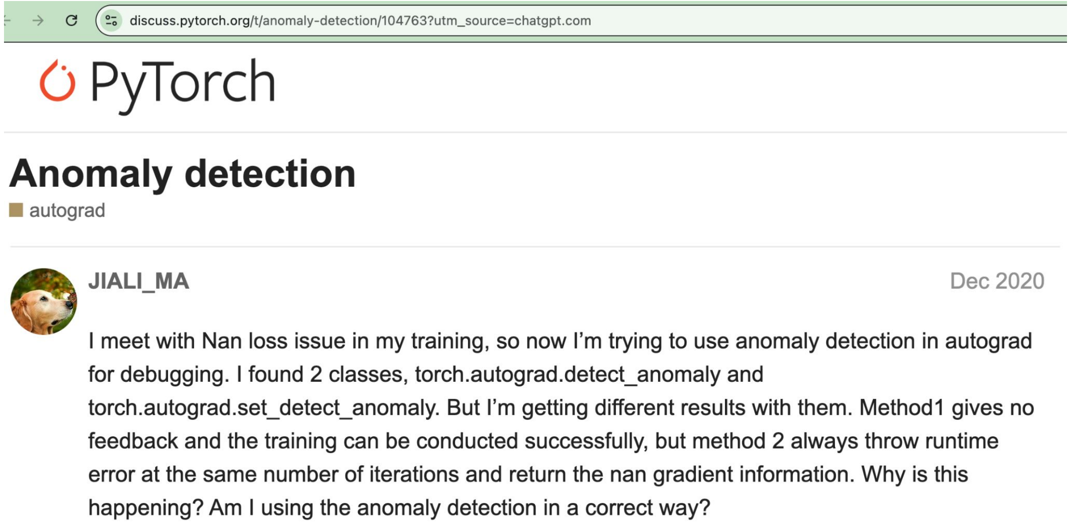
2 What PyTorch does automatically

Layer	What happens internally	Key API / behaviour
Forward kernels	Produces INF/NaN normally (IEEE "masked exceptions")—no trap.	`torch.isfinite`, `tensor.isnan()` etc. for explicit checks.
autograd anomaly detection	Wrap graph execution; throws when the *first* backward op creates a non-finite grad and prints the forward traceback.	`torch.autograd.detect_anomaly()` or `set_detect_anomaly(check_nan=True)` PyTorch Forums PyTorch
Mixed-precision scaler	`GradScaler` un-scales grads, then runs a fused `isfinite` reduction **on-GPU**. If any grad is non-finite the *optimizer step is skipped* and the loss-scale is halved.	`scaler.step(optim)` / `scaler.update()`; see AMP docs PyTorch PyTorch Forums
High-level trainers	Lightning, DeepSpeed, Accelerate all keep a "gradient overflow" flag; a skipped step just logs **OVERFLOW** and carries on.	DeepSpeed fp16 messages "Overflow detected. Skipping step." GitHub
Linalg ops	PyTorch's `torch.linalg` wrappers pass inputs to cuSOLVER/CPU LAPACK **without extra guards**; docs explicitly warn to pre-check with `torch.isfinite`. PyTorch	

PyTorch Issue 160016 tells that this can be "porous"

Framework-Recommended Solutions Don't Work Always

- Experts have told me how they deal with show-stopper exceptions
  - Try a bag of tricks (similar to those present in PyTorch - listed shortly)
- Thousands of reports that go like this



discuss.pytorch.org/t/anomaly-detection/104763?utm_source=chatgpt.com

PyTorch

Anomaly detection

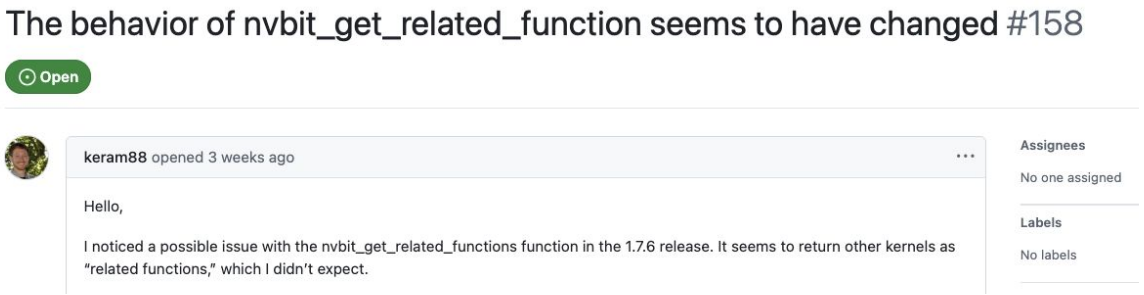
autograd

JIALI_MA Dec 2020

I meet with Nan loss issue in my training, so now I'm trying to use anomaly detection in autograd for debugging. I found 2 classes, torch.autograd.detect_anomaly and torch.autograd.set_detect_anomaly. But I'm getting different results with them. Method1 gives no feedback and the training can be conducted successfully, but method 2 always throw runtime error at the same number of iterations and return the nan gradient information. Why is this happening? Am I using the anomaly detection in a correct way?

The uphill battles we are waging… and path ahead

- We are dependent on NVIDIA's binary instrumentation framework
  - NVBit
  - Our work "merely" extends NVBit
- NVBit issues keep popping up
  - We are at present waiting for a fix (w/o which we can't use their latest release)
    - earlier releases have issues
- Without NVBit, we have (and the community has) **no tools whatsoever**
  - **Other issues: which instructions does nvcc generate?**
    - **and have we covered it?**



Exceptions are a menace in ML – NixNan helped below!

SRU

https://github.com/asappresearch/sru/issues/193

- Throws NaN
- User unable to diagnose

  We traced it to Misunderstood Pytorch allocation ; does not clear Memory as the user thought

Julia Bug

https://discourse.julialang.org/t/neuralpde-on-gpu-throws-nans-when-i-use-a-source-term-elevated-to-some-power/114048

- User has NaN issue during training in Julia that launches to GPUs; claims they have a fix

  We found the NaN still coming, (albeit much later)

BatchNorm1d

https://github.com/pytorch/pytorch/issues/162489

- User posted Sep'25
- Finds CPU / GPU differences wrt NaNs in BatchNorm1d

  GPT finds a fix when fed our Tool's traces.
  "need more precision"
  This is a "newbie bug"

This table lists the PyTorch issues we have examined using NixNan

NixNan is unique in that it can shed more light

At present, certain NVBit issues are making NixNan less effective

Original developers may have benefited a lot more, than us!

Selected Examples Tried	Type of Issue	Conclusions on Issue - following NixNan run
Karpathy's simple LLM	NaN during training	Does not affect training
Lossy compressor PyBlaz	NaN during compression	Does not affect operation
Issue 125674	Nan in grad. of scaled dotprod attention	Can produce traces that may help customer
Issue 156020	Value too high	Trace suggests NaN written into memory
Issue 156707	Issue with MPS	Tracing on GPUs possible
Issue 152737	NaN prop in Conv1d	Two NaN Stores into memory observed
Issue 157272	Reciprocal NaN issue	Bug in CPU; ran GPU code for comparison (traces)
Issue 159333	Foreach.copy bug	NaN store into memory present. Traces informative?
Issue 68425	Can run min instance	NaN due to predicated execution known
Issue 160016	PyTorch's own anomaly detection can be "porous" (does not report in the cases shown here)	Reproduces issues pointed out
BioMistral	None; runs	Lots of NaN/INF observed; harmless?

The Reality : In systems such as PyTorch, must use Tools such as NixNan **during Kernel Design-Time!**

(Else, it is too hopeless to find bugs later.)

Here are a few summary observations wrt PyTorch, to date

Easy bugs : can often spot and use (with manual effort)	Short and Tricky : Basic Logic of Kernel can be broken!	Too Large to Debug : Don't wait till this time!
BatchNorm1d : Switch from FP32 to FP64 with better accumulation	#160876: silu(x) = x * sigmoid(x) returns NaN; x = -inf (used to be 0)	Many issues are “too large to be posted” – very little can be done
Evident that non-experts suffer. Still gotta help them (non-uppity)	Naive implementation: -inf * 0 L’Hopital’s rule: 0	Solution: Avoid inserting bugs into simpler APIs by having unit-testing

For effective root-causing, we need to bridge across levels of calls

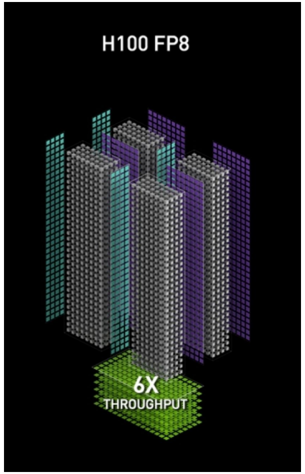
GPTs may help bridge **PyTorch** calls, **GPU** calls, and **Exception-annotated SASS**

Folder	NameOfKernel	FunctionsRun	
medgemma	void at::native::vectorized_elementwise_kernel	{}	[]
medgemma	void at::native::elementwise_kernel	{}	[]
medgemma	void at::native::reduce_kernel	{}	[]
medgemma	void at_cuda_detail::cub::DeviceScanInitKernel	{}	[]
medgemma	void at_cuda_detail::cub::DeviceScanKernel	{}	[]
medgemma	void at::native::unrolled_elementwise_kernel	{}	[]
medgemma	void at::native::	{}	[]
medgemma	void gemmk1_kernel	{at::native::vectorized_elementwise_kernel}	Details 1
medgemma	void cutlass::Kernel2	{at::native::vectorized_elementwise_kernel, cutlass::Kernel2}	Details 2
medgemma	void pytorch_flash::flash_fwd_kernel	{cutlass::Kernel2, pytorch_flash::flash_fwd_kernel}	Details 3
medgemma	void cublasLt::splitKreduce_kernel	{}	[]
medgemma	ampere_s16816gemm_bf16_256x128_ldg8_stages_32x3_tn	{at::native::vectorized_elementwise_kernel, cutlass::Kernel2}	Details 4
medgemma	ampere_bf16_s16816gemm_bf16_64x64_ldg8_f2f_stages_64x6_tn	{at::native::vectorized_elementwise_kernel, cutlass::Kernel2, pytorch_flash::flash_fwd_kernel}	Details 5

Note that all line-numbers are 0 (feature "vanished" in one NVBit release…)

```
#nixnan: error [NaN] detected in operand 1 of instruction @P0 FADD.FTZ R6, R0, 1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction @P0 FADD.FTZ R6, R0, 1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction MUFU.RCP R7, R9 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction MUFU.RCP R7, R9 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction FADD R0, R0, -1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction FADD R0, R0, -1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction FMUL R10, R6, R7 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 2 of instruction FMUL R10, R6, R7 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction FMUL R10, R6, R7 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction FFMA R7, R0, R0, 1 ; in function archimedes_kernel_fp16 at line 0 of type f32
```

Deep-Dive : Tensor-Core Support of NixNan



> LD_PRELOAD = nixnan.so
Pre-Compiled Application Binary

Intercept all "Kernel Launch" CUDA Driver API Calls

For Each FP instruction :

Inject_instrumentation();

CUDA Driver APIs

```
----- Test NaN: A[0,0]=inf, B[0,0]=1, C[0,0]=-inf -----
A[0,0] = inf (0x7c00), B[0,0] = 1.000000 (0x3c00), C[0,0] = -inf (0xfc00)
Computing D = A * B + C with Tensor Cores...
#nixnan: error [NaN,infinity] detected in instruction HMMA.16816.F16 R20, R4.reuse, R16, RZ ;
#nixnan: error [NaN] detected in instruction HMMA.16816.F16 R22, R4, R18, RZ ; in function WMM
D[0,0]=nan (0x7fff)
```

Tensor core register layout

Each thread holds part of the matrices in its registers

T1

R0	R1	R2	R3	R4	R5	R6	R7
A0	*A1*	*A2*	*A3*	*B0*	*B1*	*C0*	*C1*

T2

R0	R1	R2	R3	R4	R5	R6	R7
A4	*A5*	*A6*	*A7*	*B2*	*B3*	*C2*	*C3*

T3

R0	R1	R2	R3	R4	R5	R6	R7
A8	*A9*	*A10*	*A11*	*B4*	*B5*	*C4*	*C5*

T4

R0	R1	R2	R3	R4	R5	R6	R7
A12	*A13*	*A14*	*A15*	*B6*	*B7*	*C6*	*C7*

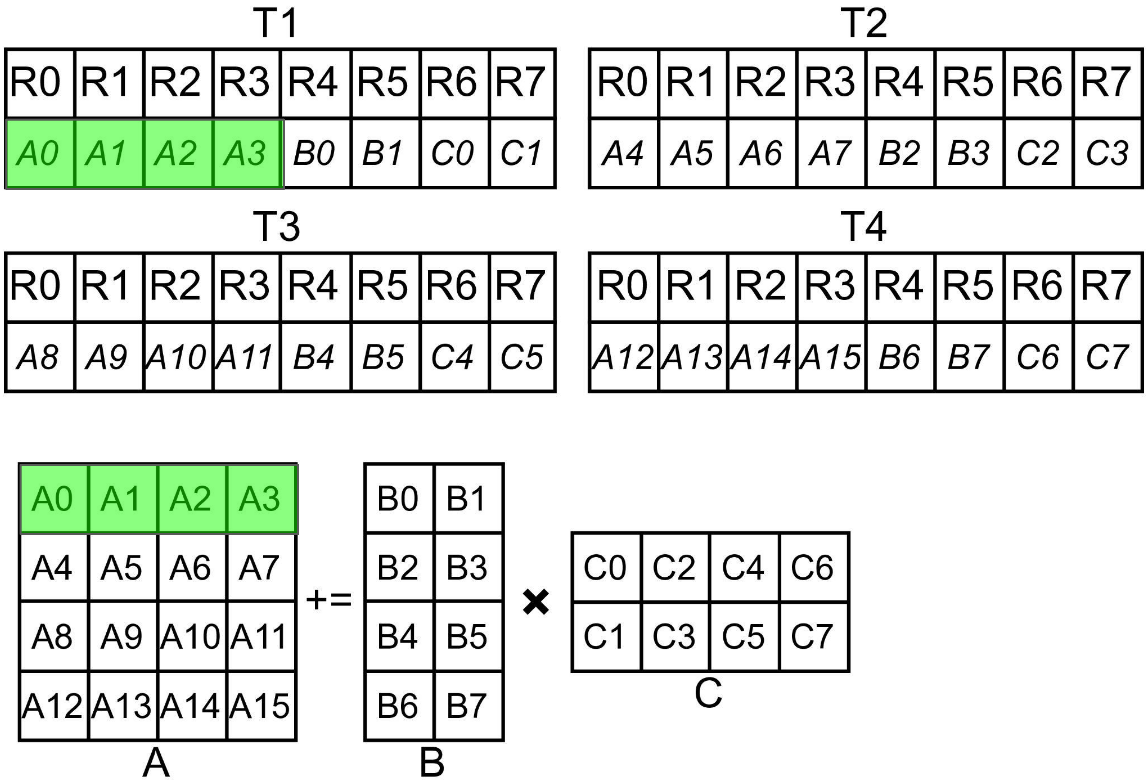
$$\underbrace{\begin{bmatrix} A0 & A1 & A2 & A3 \\ A4 & A5 & A6 & A7 \\ A8 & A9 & A10 & A11 \\ A12 & A13 & A14 & A15 \end{bmatrix}}_{A} \mathrel{+}= \underbrace{\begin{bmatrix} B0 & B1 \\ B2 & B3 \\ B4 & B5 \\ B6 & B7 \end{bmatrix}}_{B} \times \underbrace{\begin{bmatrix} C0 & C2 & C4 & C6 \\ C1 & C3 & C5 & C7 \end{bmatrix}}_{C}$$

Tensor core register layout

Each thread holds part of the matrices in its registers

Thread T1 holds:

- The first four elements of A

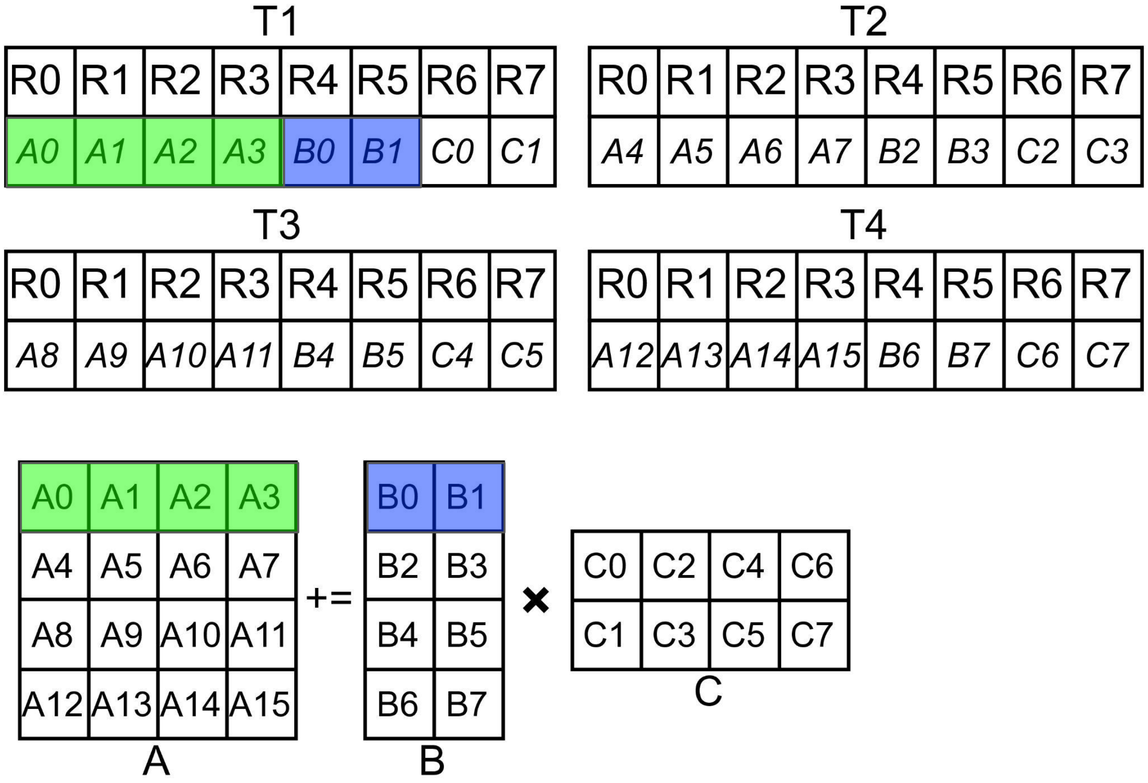


Tensor core register layout

Each thread holds part of the matrices in its registers

Thread T1 holds:

- The first four elements of A
- The first two elements of B

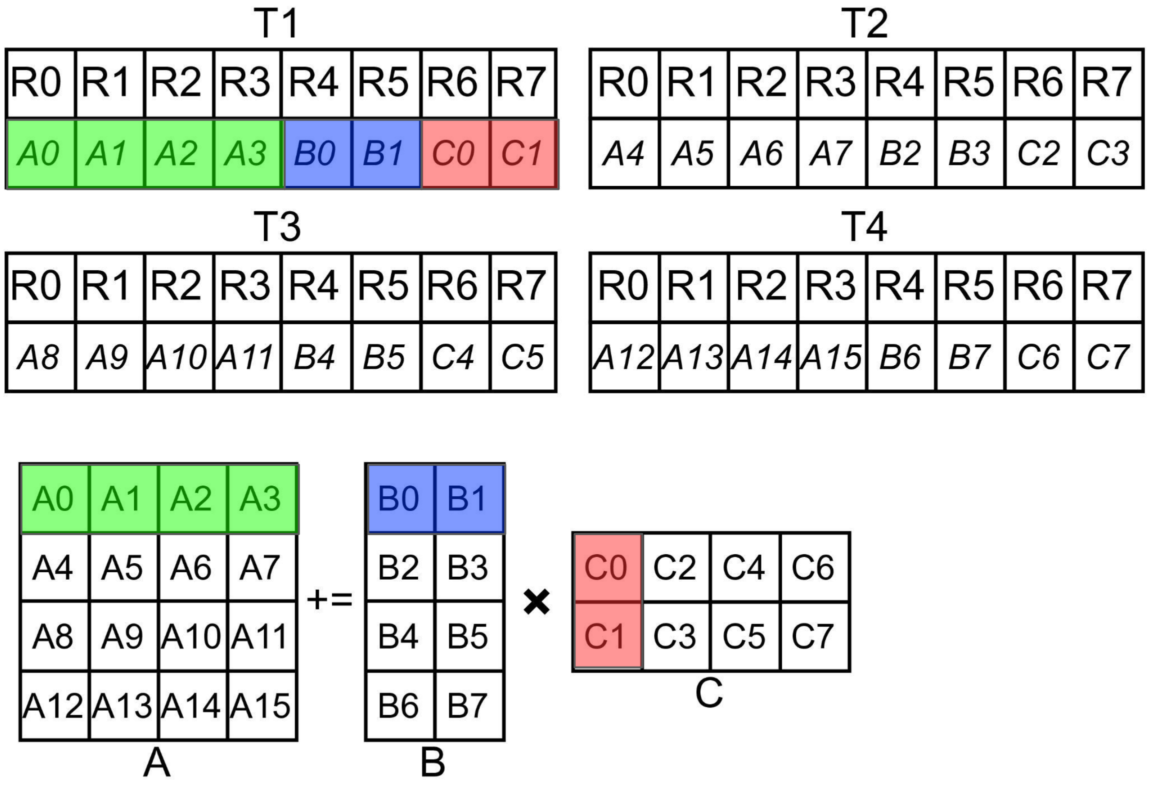


Tensor core register layout

Each thread holds part of the matrices in its registers

Thread T1 holds:

- The first four elements of A
- The first two elements of B
- The first two elements of C



Concluding Remarks : Let's visit all the topics discussed

- **NixNan helps reveal exceptions via Binary Instrumentation**
  - **Helpful for closed-source libraries (many are)**
- **Knowing which exceptions matter is not straightforward**
  - **Easier with HPC codes**
    - **NaN / INF Exceptions almost always are bad**
  - **Not well-understood with ML codes**
    - **NaN / INF Exceptions seem to fly around with abandon**
    - **Can "jiggle" ML models and see exceptions change (as on Slide 32)**
      - **This may give insights for improvement / debugging**
      - **Users are suffering from exceptions (e.g., PyTorch Open issues)**
- **Limitations, Need for Community + Manufacturer Cooperation**
  - **GPUs are not well-documented**
    - **Reverse-engineering GPU behaviors was necessary**
      - **Not easy to scale, considering the rate of arrival of new GPUs**
  - **Help from GPU manufacturers can greatly help advance FP debugging**

Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- **11:05 : FloatGuard**
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

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

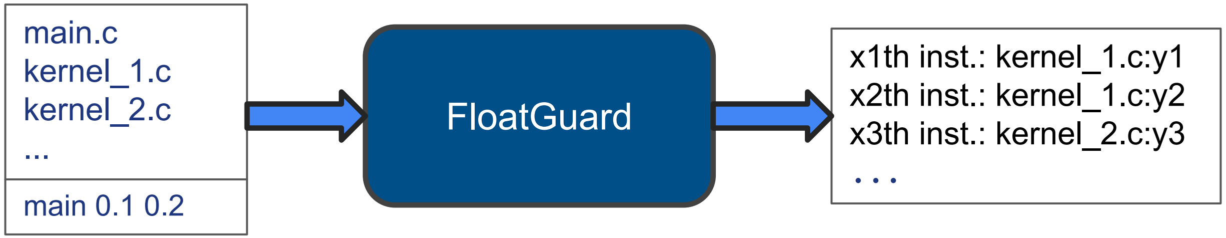
Challenges using FP Exception Registers

What can we conclude from our demo?

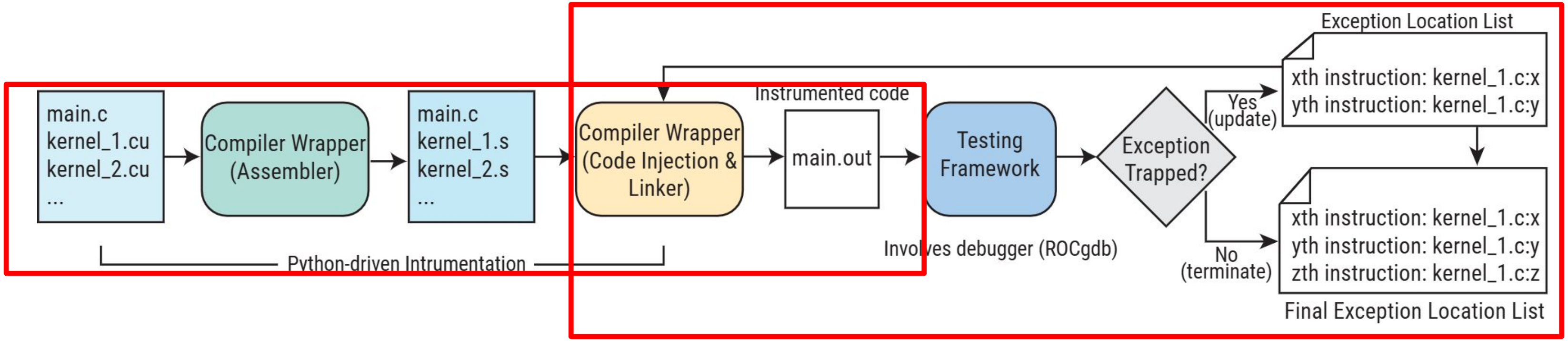
1. Exception trapping is off by default in kernels
   Need to manually enable in each kernel thread
2. Program counter after a trap may be delayed
3. Program state unrecoverable with trapped exception
   Difficult to track exception after the first

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



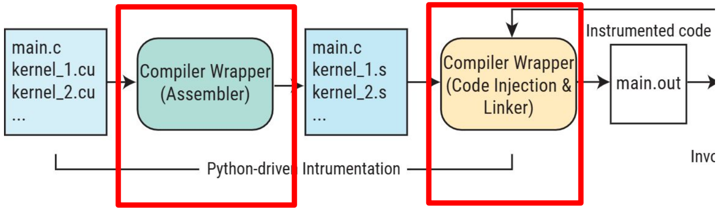
Code Instrumentation

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



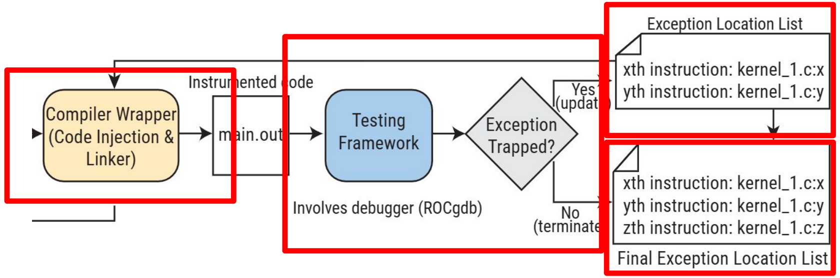
Code Instrumentation - Assembly Injection

- 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!

Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- **11:50 : Closing Remarks**
- **11:55 : Fill Survey**