



# Introduction to Floating-Point Analysis and Reproducibility



Lawrence Livermore  
National Laboratory



THE  
UNIVERSITY  
OF UTAH®

**UC DAVIS**  
UNIVERSITY OF CALIFORNIA



**JMU**  
JAMES MADISON  
UNIVERSITY.

Ignacio Laguna, Harshitha Menon  
Lawrence Livermore National Laboratory

Michael Bentley, Ian Briggs, Pavel Panchekha, Ganesh Gopalakrishnan  
University of Utah

Hui Guo, Cindy Rubio González  
University of California at Davis

Michael O. Lam  
James Madison University



# Topics in this full-day tutorial

- **Floating-point analysis**
  - For improving the performance of the code
- **Non-Reproducibility**
  - Caused by program execution
  - Caused by floating-point result nondeterminism
- **Why combine FP analysis and Non-Reproducibility in one tutorial?**
  - **Because, separating them through external symptoms alone is difficult in practice!**

# Reproducibility

The hope/expectation that a subsequent run of the program under “**identical conditions**” will produce the same answer

Life works around reproducibility

Even babies expect it



[http://www.clipartpanda.com/clipart\\_images/elevator-clipart-69356901](http://www.clipartpanda.com/clipart_images/elevator-clipart-69356901)

#### Clipart Info

Use this image on your  
Powerpoints, School  
Projects, Reports and More!



Lecun's Turing lecture



# Reasons for Lack of Reproducibility in Numeric Programs, and Tools We Present

- Nondeterministic MPI message matching
  - Send messages that race to match a wildcard receive
    - **TOOL: ReMPI**
- Data races that change the space of executions
  - “Pink-elephant values” [Sutter]
    - **TOOL: Archer**
- Compiler optimizations that change the binaries that get linked
  - Makes code non-portable across platforms
    - **TOOL: FLiT**
- FP Exceptions that are thrown
  - Makes the execution unexpectedly terminate
    - **TOOL: FPChecker**



# Floating-point Analysis Topics and Tools

- Understanding and Benchmarking Floating-Point Codes
  - So that we have an objective truth to tool behavior, comparisons
    - **FRAMEWORK: FPBench**
- Precision Tuning
  - Determine precision changes that improve performance
    - **TOOLS: Precimonious, HiFPTuner**
- Framework for source-to-source precision tuning & analysis
  - Enables community to conduct precision tuning research
    - **TOOLS: ADAPT, FloatSmith**



A quick introduction to today's major topic

## Some basics of Floating-Point Arithmetic

# The Floating-Point number system is not new

Then



Zuse Z1 (~1938)

Now

**IEEE Standard for Floating-Point Arithmetic**

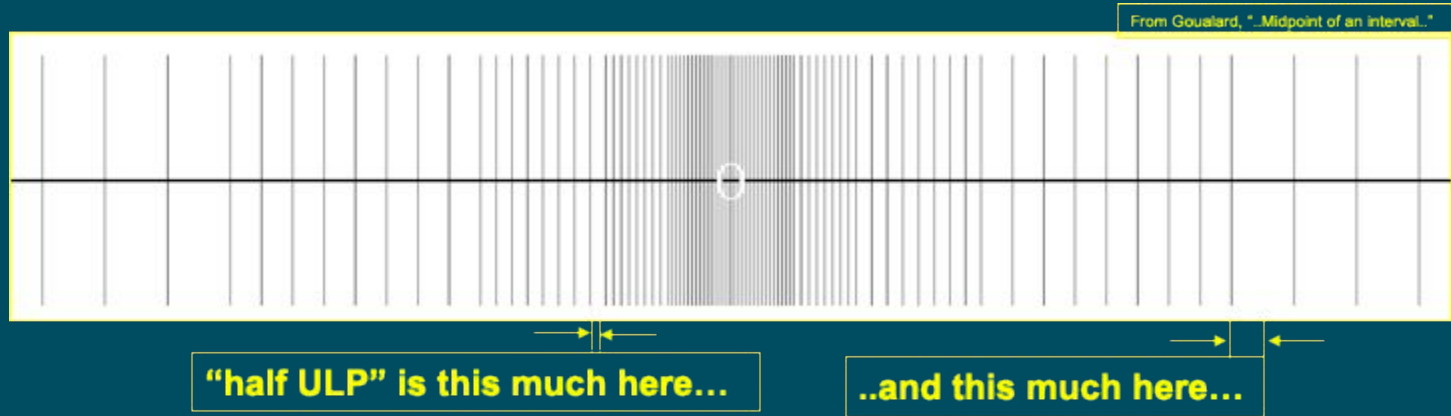


# Floating-Point approximates Reals

- Because of rounding,  $(x+y)+z \neq x+(y+z)$ 
  - **And many more such identities are violated**
- Compilers can change your math
  - $x/y \rightarrow x * (1/y)$
- Rounding errors are non-intuitive
  - **Because of the uneven FP number scale**



# The Floating-Point Rounding is Non-Intuitive



The FP number system tries to span a large range using an “insufficient number of bits”



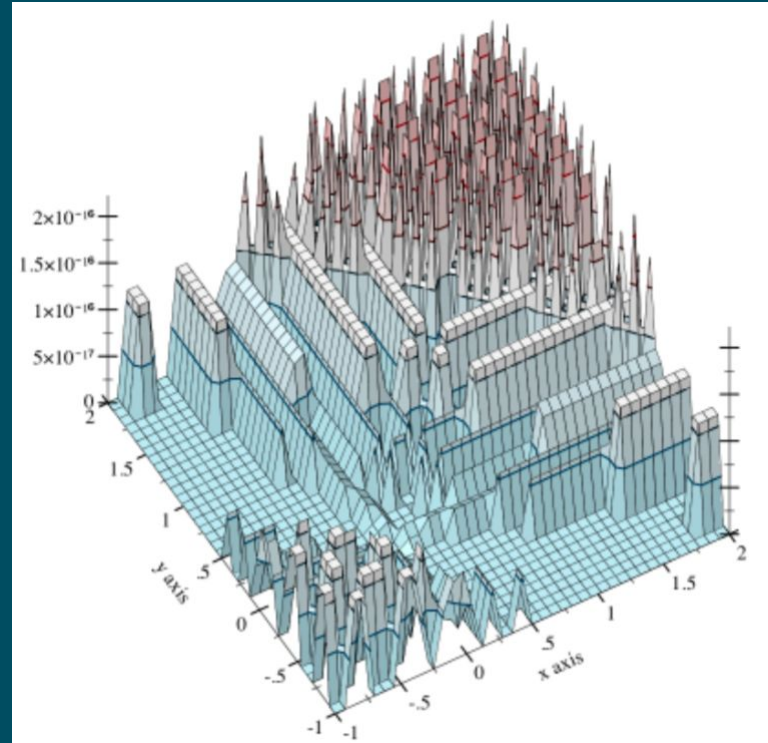
# The FP error function is highly non-intuitive

E.g.

Rounding error of  $(x+y)$  as  
a function of  $x$  and  $y$

IEEE Model of error:

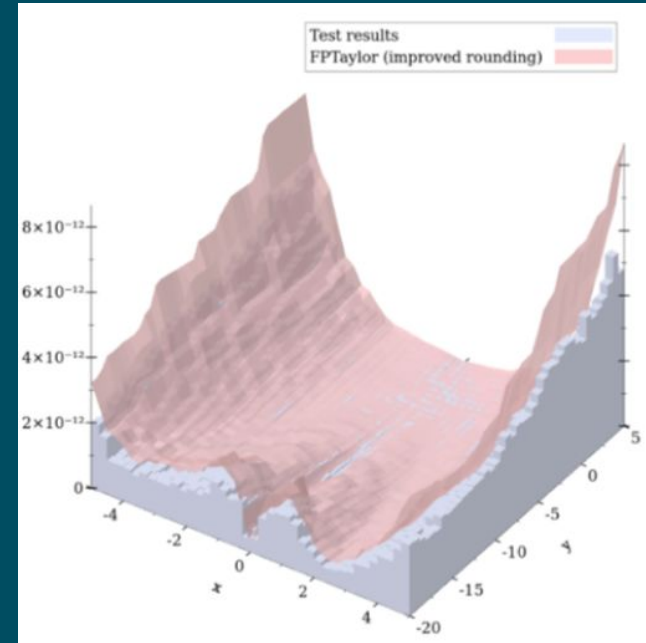
$$(x+y) \cdot \text{half-ULP}$$



# The FP error function is highly non-intuitive even at a more macroscopic level... Heuristic solutions recommended in practice!

E.g. Rounding error of the JetEngine benchmark of Darulova and Kuncak as plotted by Solovyev using his FPTaylor tool, with ground-truth obtained thru shadow-value simulation.

```
double jetEngine(double x1, double x2) {  
    double t = 3 * x1 * x1 + 2 * x2 - x1  
    double q = x1 * x1 + 1  
    double p = t / q  
    double s1 = 2 * x1 * p * (p - 3)  
    double s2 = x1 * x1 * (4 * p - 6)  
    double s3 = 3 * x1 * x1 * p  
    double s4 = x1 * x1 * x1  
    double s5 = 3 * p  
    return x1 + ((s1 + s2) * q + s3 + s4 + x1 + s5)  
}
```





## Kahan's observation

Numerical errors are rare, rare enough  
not to care about them all the time,  
but yet not rare enough to ignore them.  
— William M. Kahan



# Floating-Point Analysis is Suddenly “Front and Center” in HPC + many other areas

- **Allocating needlessly high precision increases data movement**
  - Multiple precision types are on the rise
    - Often driven by ML
- **The variety of hardware is increasing**
  - GPUs and other accelerators
    - Their normal behaviors as well as EXCEPTIONS are on the rise
- **Compilers exploit floating-point in an increasing number of ways**
  - Compiler flags mean different things
    - Compilers may heed your flags selectively



## Frenetic pace of FP research now

- Multiple conferences
- Many sessions per conference
- Many different issues

Very little that is tangible for a practitioner to try some of these out

**PURPOSE OF THIS TUTORIAL: Change this!**



# Goals of this Tutorial

- **Introduce FOUR mileposts in your repertoire of knowledge**
  - Four tools you can practice during the tutorials
  - You can apply them in your own projects!
- **We are a resource you can count on during your future work**
  - We are invested in multiple research projects in this area
  - We know many more researchers and practitioners whose work we can refer

We hope to build a community of researchers and practitioners



# Access to AWS Instances (changeme)

- You will be given access to AWS instances
  - User, password, and IP address will be provided

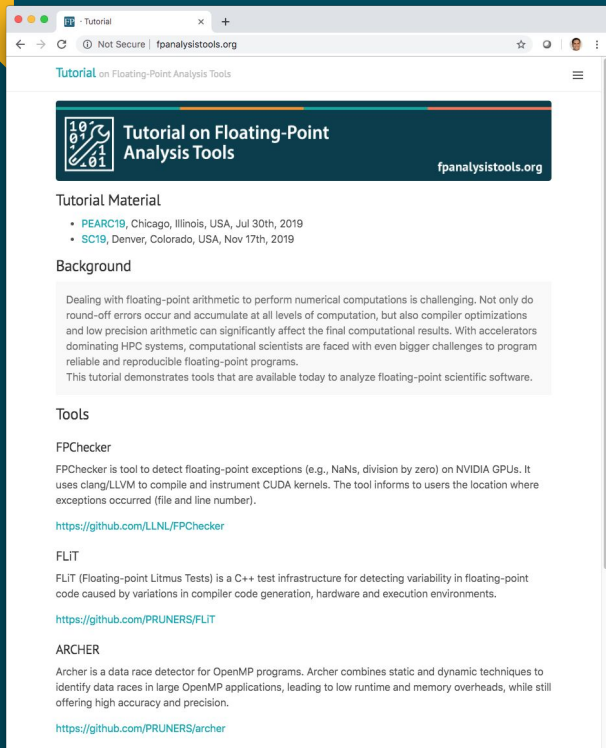
- How to access your instance:

```
ssh user@1.2.3.4
```

- Exercises for each module located in user's /home directory

```
/home/user1/  
|---Module-T00L1  
|---exercise-1  
|---exercise-2  
|---exercise-3  
|---Module-T00L2  
|---exercise-1  
|---exercise-2  
|---exercise-3  
...
```

# Website & Schedule: [fpanalysistools.org](http://fpanalysistools.org)



Tutorial on Floating-Point Analysis Tools

**Tutorial Material**

- PEARC19, Chicago, Illinois, USA, Jul 30th, 2019
- SC19, Denver, Colorado, USA, Nov 17th, 2019

**Background**

Dealing with floating-point arithmetic to perform numerical computations is challenging. Not only do round-off errors occur and accumulate at all levels of computation, but also compiler optimizations and low precision arithmetic can significantly affect the final computational results. With accelerators dominating HPC systems, computational scientists are faced with even bigger challenges to program reliable and reproducible floating-point programs. This tutorial demonstrates tools that are available today to analyze floating-point scientific software.

**Tools**

**FPChecker**

FPChecker is tool to detect floating-point exceptions (e.g., NaNs, division by zero) on NVIDIA GPUs. It uses clang/LLVM to compile and instrument CUDA kernels. The tool informs to users the location where exceptions occurred (file and line number).

<https://github.com/LLNL/FPChecker>

**FLIT**


FLIT (Floating-point Litmus Tests) is a C++ test infrastructure for detecting variability in floating-point code caused by variations in compiler code generation, hardware and execution environments.

<https://github.com/PRINERS/FLIT>

**ARCHER**

Archer is a data race detector for OpenMP programs. Archer combines static and dynamic techniques to identify data races in large OpenMP applications, leading to low runtime and memory overheads, while still offering high accuracy and precision.


<https://github.com/PRINERS/archer>



Tutorial on Floating-Point Analysis Tools

## Tutorial on Floating-Point Analysis and Reproducibility Tools for Scientific Software

SC19, Denver, Colorado, USA  
Nov 17th, 2019  
Time: 8:30am - 5pm (full day tutorial)



**Presenters**

- **Ignacio Laguna** (organizer), Lawrence Livermore National Laboratory
- Michael Bentley, University of Utah
- Ian Briggs, University of Utah
- **Ganesh Gopalakrishnan**, University of Utah
- **Hui Guo**, University of California, Davis
- **Michael O. Lam**, James Madison University
- **Harshitha Menon**, Lawrence Livermore National Laboratory
- **Pavel Panckeha**, University of Utah
- **Cindy Rubio González**, University of California, Davis