Computers don't go to high school

Safety and Security Risks Induced by Machine Arithmetic

Thomas Wahl

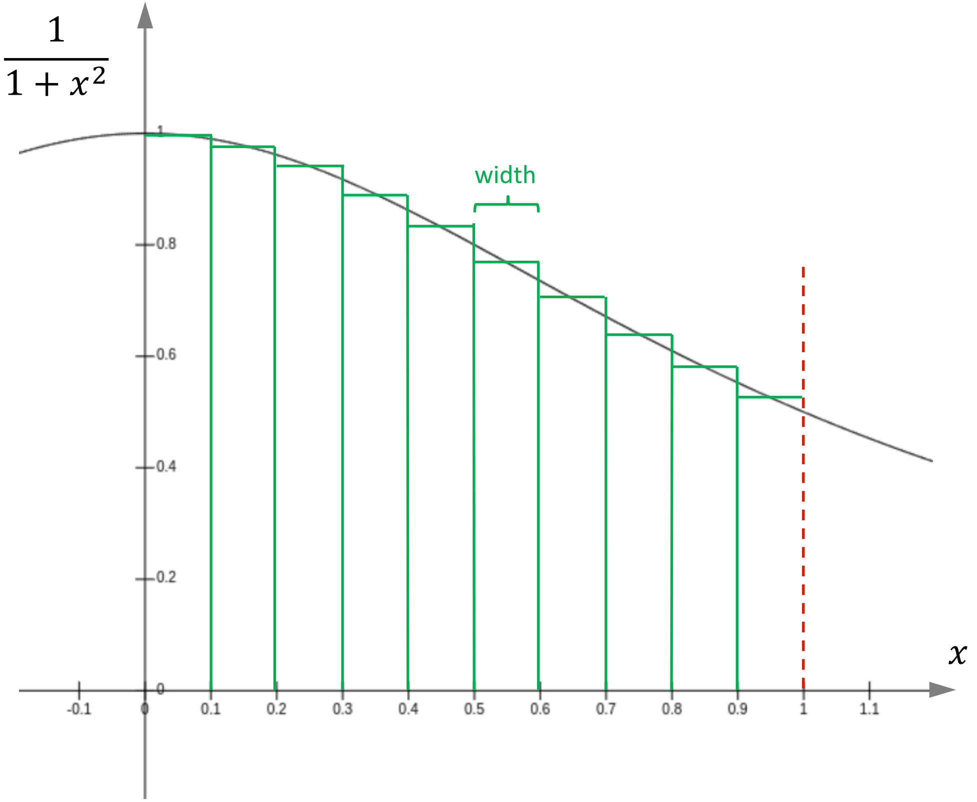
May 9, 2024

The Story

$$\pi = 4 \times \int_0^1 \frac{1}{1+x^2} \partial x$$

How do we approximate the integral?

$\int_0^1 \frac{1}{1+x^2} \partial x$:



Let’s play.

Let's study: Floating-Point Basics

Floating-Point Arithmetic: Basic Motivations

There are just too many real numbers out there.

- Fix the *word size*, i.e., the number of represented digits: 3.141592653 (10)
- But what about the decimal point?

Fixed-point arithmetic:

Point position is fixed:

$$00003.14159$$

✓ Can process them as integers!

– Inflexible. What about probabilities: $p \in [0,1]$?

Floating-point arithmetic (FPA):

Point position is arbitrary (it "floats"):

$$3.141592653 \;\ldots\; 3141592653$$

✓ Flexible: larger range, varying precision

– Hardware more implementation complex

Floating-Point Arithmetic: Basic Motivations

$3.141592653 \ldots 3141592653$ is not how FP numbers are stored in machines:

- Common is *binary* FPA; this talk uses (a simplified version of) *decimal* FPA
- FP number is not one monolithic sequence of digits, but:

$$+0.314159265 = \underbrace{(-1)^0}_{\text{sign}} \times \underbrace{3.14159265}_{\text{mantissa}} \times \underbrace{10^{-1}}_{\text{exponent}}$$

$$-3141592653 = (-1)^1 \times 3.141592653 \times 10^9$$

sign — mantissa — exponent

Formats like `float` and `double` differ in mantissa and exponent bit width.

Floating-Point Arithmetic Approximates

1. Not all numbers are representable:

$$3.14159265358979323846264338327\ldots \rightarrow 3.141592653$$

2. Set of representable FPA numbers *not closed under FP operations*: results may exceed the representable range or the precision:

$$3.141592653 \otimes 3.141592653 = 9.869604397383578409 \rightarrow 9.869604397$$

$$3141592653 \otimes 3141592653 = 9869604397383578409 \rightarrow \infty$$

[Example: Patriot MDS failed to intercept Scud, 28 casualties. February 1991.
Ultimate cause: 0.1 not representable in binary FP!]

FPA and High-School Arithmetic

An unsatisfiable equation:

$$x \oplus y = x \quad \text{for} \quad y > 0 \quad ??$$

Let’s play.

FPA and High-School Arithmetic

An unsatisfiable equation:

$$x \oplus y = x \quad \text{for} \quad y > 0 \quad ??$$

What is happening?

$$3141592653 \oplus 0.1 =$$

$$3.141592653 \quad \times 10^9 \oplus 1.0 \times 10^{-1}$$ → standard FP number repr.

$$3.141592653 \quad \times 10^9 \oplus 0.0000000001 \times 10^9$$ → alignment

$$3.1415926531 \times 10^9$$ → mantissa addition

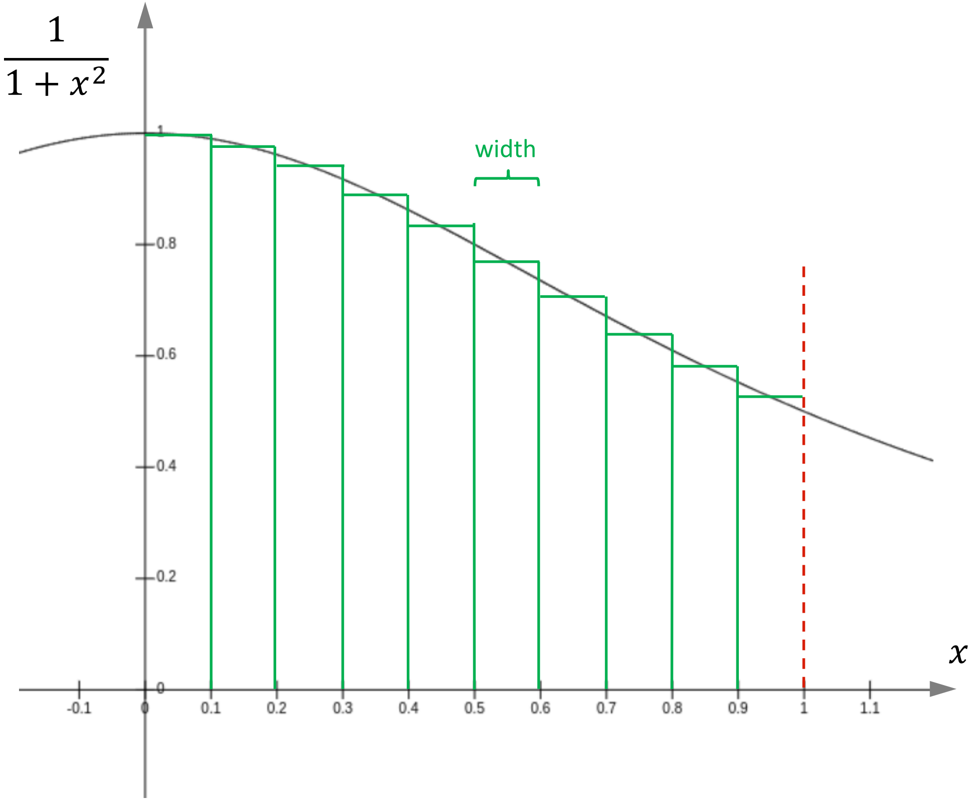
$$3.141592653 \quad \times 10^9$$ → back to standard FP repr.

$$= 3141592653$$

"Absorption"

How do we approximate the integral?

$\int_0^1 \frac{1}{1+x^2} \partial x$:



FPA and High-School Arithmetic

<u>Corollary:</u> FP addition (multiplication, etc.) is infamously *not associative*:

$$(-x \oplus x) \oplus y = \quad 0 \oplus y = y,$$
$$-x \oplus (x \oplus y) = -x \oplus x = 0.$$

if $0 < y \ll x$.

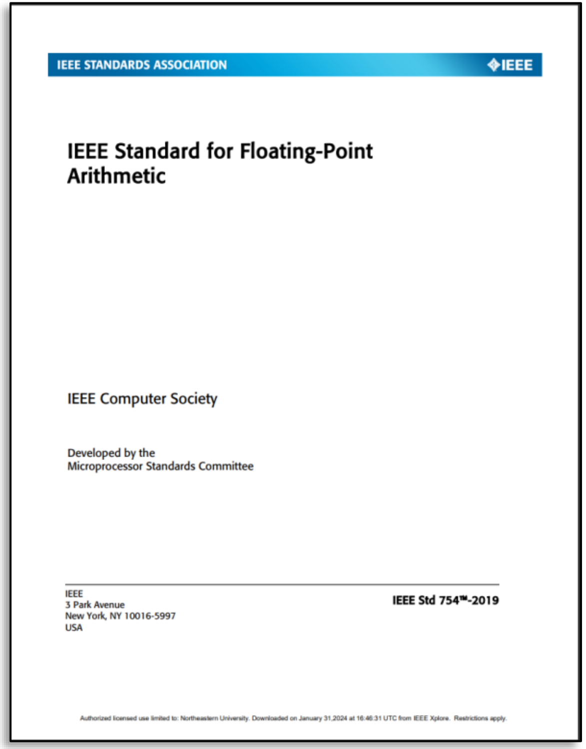
"… can result in incorrect output for programs that depend on an exact implementation [of FPA]."

A nightmare for rewriting tools like optimizers!

```
gcc -ffast-math
gcc -funsafe-math-optimizations
```

Platform-Dependence of FPA

Lack of (Full) FPA Standardization


IEEE STANDARDS ASSOCIATION ◆IEEE

IEEE Standard for Floating-Point Arithmetic

IEEE Computer Society

Developed by the
Microprocessor Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 754™-2019

"Wait, what?"

- Aren't operations fixed? $x \oplus y = \mathrm{rd}(x + y)$.
- Yes, but what is not fixed is *expression* evaluation:

$$x \oplus y \oplus z$$

  – Expressions are not computed by hardware (IEEE 754 is about standardizing FPU implementation on microprocessors)

  – *"A programming language standard specifies one or more rules for expression evaluation"*, includ-ing *"the order of evaluation of operations."* [p. 72]

Impact of Evaluation Order on FPA

Why is this a problem?

- Absorption and non-associativity can cause reordering to change results

Why would compilers reorder?

- Peephole optimizations: $x \oplus y \oplus (-x)$
- Massive optimizations: *parallelization* on multicore and multiprocessors

Impact of Evaluation Order on FPA: Theory

$$a + b + c + d + e + f + g + h$$

$$\Sigma\, a, b \qquad \Sigma\, c, d \qquad \Sigma\, e, f \qquad \Sigma\, g, h$$

Four-node cluster:

$$\Sigma\, a, b, c, d \qquad \Sigma\, e, f, g, h$$

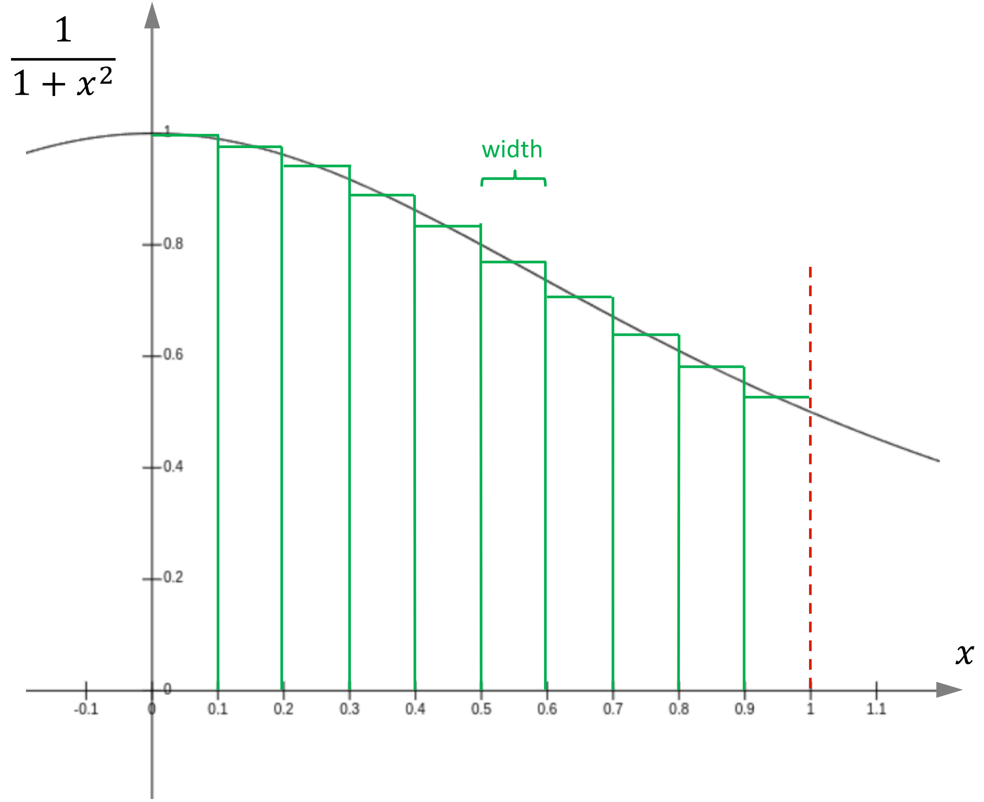
$$\Sigma\, a, b, c, d, e, f, g, h$$

$$((a + b) + (c + d)) + ((e + f) + (g + h))$$

Let’s play.

How do we approximate the integral?

$$\int_0^1 \frac{1}{1+x^2}\,\partial x:$$



Architecture-Dependence of Floating-Point

High-performance computing: matrices, matrices, matrices!

- Dot product: $a \times b + c \times d + e \times f$
- Very common form of expression: $x \times y + z$
- Speed *and precision* optimization: Expression becomes *operation*:

$$\mathrm{FMA}(x, y, z) = \mathrm{rd}(x \times y + z)$$

(single FP instruction) instead of $\mathrm{rd}(\mathrm{rd}((x \times y) + z)$ (two instructions).

→ Fused Multiply-Add

Architecture-Dependence of Floating-Point

<u>FMA example:</u> Ray Tracing

For some input with very small `radiusSq`, we obtained:

```
1 int raySphere(float *r, float *s, float radiusSq) {
2   float A = dot3(r,r);
3   float B = -2.0 * dot3(s,r);
4   float C = dot3(s,s) - radiusSq;
5   float D = B*B - 4*A*C;
6   if (D > 0)
7     ...
8 }
```

Architecture	Value of `D` (line 6)
Intel 64-bit CPU	+4.55
NVIDIA Quadro 600 GPU	−3.56

Platform-dependent **control-flow**!

GRAMMATECH

Security Risks Induced By FPA

Special Values in Floating-Point Arithmetic

FP values are not a subset of the real numbers:

- $\pm\infty$: overflow, e.g. "big" $\otimes$ "big", $1.0/0.0$
- NaN : e.g. $0.0/0.0$, $\infty - \infty$
- "subnormals" : underflow, i.e., $< \min_{norm} = 0.1 \times 10^{e_{\min}}$

Operation	CPU cycles
$normal \cdot normal = normal$	10
$normal \cdot normal = subnormal$	124
$subnormal \cdot normal = normal$	124
$subnormal \cdot normal = subnormal$	124
$subnormal \cdot subnormal = 0$	10
$subnormal \cdot 0 = 0$	10

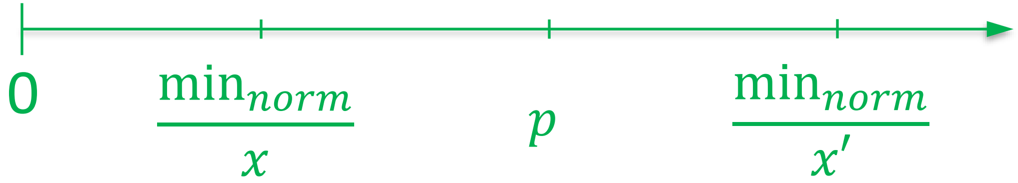
(Intel i7-7700 quad-core)

FPA-Induced Timing Channels

Suppose a device computes $x \otimes p$.

x is an input; goal is to determine design parameter p.

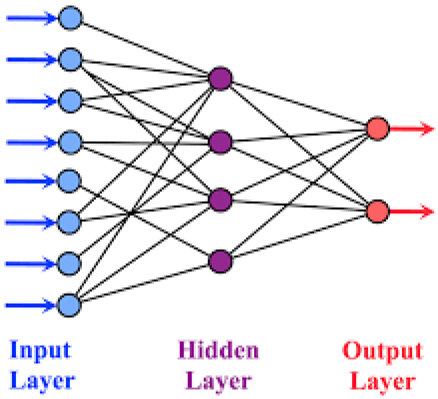
1. Find small inputs x, x' such that $T(x \otimes p) \ll T(x' \otimes p)$
2. Hence $x \otimes p$ is normal, $x' \otimes p$ is subnormal
3. Hence $x' \times p < \min_{norm} \leq x \times p$, i.e.



Reverse-Engineering NN Parameters

<u>Goal:</u> recover weights and biases in a **neural network**.

<u>Assumption:</u> attacker can measure time *per layer*



$$\begin{pmatrix} w_{11} & \dots & w_{1m} \\ w_{21} & \dots & w_{2m} \\ & \vdots & \\ w_{n1} & \dots & w_{nm} \end{pmatrix} \times \begin{pmatrix} i_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} w_{11} \times i_1 + b_1 \\ w_{21} \times i_1 + b_2 \\ \vdots \\ w_{n1} \times i_1 + b_n \end{pmatrix}$$

Exploiting FPA-Induced Timing Channels

Victims:

← expensive IP

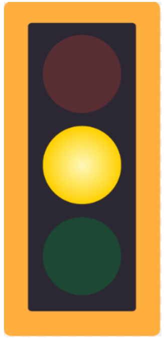
sensitive personal data →
model inversion attacks map
DNN model back to training data

<u>Mitigation:</u> **disable** subnormal numbers: `-ftz=true` (NVIDIA C compiler)

Summary

Floating-Point Arithmetic: Cautions

Enables math with a wide range of real-ish numbers. But:



- Approximates “too large” and “too precise” numbers.
  This sabotages algebra rules → not reliably optimizable
- Results depend on language/compiler/computational platform.
  → not portable
- Compute time (and power!) of operations result dependent.
  Clever reverse-engineering breaks confidentiality → exploitable

References

- T. Mattson, R. Eigenmann: OpenMP Tutorial. International Conference on High-Performance Computing (SC), 1999. [π example]
- C. Gongye, Y. Fei, T. Wahl: Reverse engineering deep neural networks using floating-point timing side-channel. Design-Automation Conference (DAC), 2020.
- Y. Gu, T. Wahl, M. Bayati, M. Leeser: Behavioral non-portability in scientific numeric computing. European Conference on Parallel and Distributed Computing (EURO-PAR), 2015. [Impact of reordering and FMA]
- Institute of Electrical and Electronics Engineers, Inc.: Standard for Floating-Point Arithmetic. IEEE Std 754™-2019, 22 July 2019.
- M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, H. Shacham: On Subnormal Floating Point and Abnormal Timing. Security and Privacy, 2015. [Timing channel extracts webpage content in `<iframe>`]