Potential Directions for Moving IEEE 754 Forward

Jason Riedy

Center for Research into Novel Computing Hierarchies at Georgia Tech

7 May 2020

The IEEE standards process…

For background:

- Starts with a PAR: Project Authorization Request. Defines scope of work.
  - 2018, um, 2019: No new requirements. “Bug fix.”
- Standards are reviewed / renewed every ten years.
  - 1985, 2008, 2019… Oops.
- **Recruiting for 2028/9.**
  - Please? No, really. Time to start thinking.
- And these are **my** views, not the committee’s.

Motion to the language level

From IEEE 754-1985:

> It is intended that an implementation of a floating-point system conforming to this standard can be realized entirely in software, entirely in hardware, or in any combination of software and hardware. It is the environment the programmer or user of the system sees that conforms or fails to conform to this standard. Hardware components that require software support to conform shall not be said to conform apart from such software.

Motion to the language level

- Interchange formats
  - Binary and decimal
  - Really the only hardware-ish portion
- Operations, attributes (rounding), *etc.*

General goal: Provide a predictable, well-defined way to map programming languages to arithmetic hardware.

Obstacles: Languages, religion.

- Operation context is non-local.
  - Rounding modes and exception flags are not well mapped.
- (FYI, there are no traps. No. Those are gone. Now "alternate exception handling.")

“Little” aspects: graduating recommendations?

Which recommended operations graduate to being required?

- Fixed min/max?
  - Signaling NaNs shouldn’t signal in a quiet op.
  - Oops. An unexpected interaction in seperate sections.
- Correctly rounded special functions?
- Augmented arithmetic operations?
- Reductions?
- NaN payload operations?

And there are security aspects to consider now.

“Little” aspects: retiring unused pieces?

- Extended and extensible precisions?
- Nail down underflow?
- (Sure others will have more opinions…)

“Little” aspects: debatable decisions

- Special function special cases
  - Power, x^y. All the joy for integral values of y.
  - Preference for conformal mappings
  - “Much ado about nothing’s sign bit…”
- `abs`, `negate` as numeric rather than “bit”?
  - So raise invalid on signaling NaNs.
  - What about `copy`? Traditionally left to implementations.

Relations between precisions

Format	Significand	Exponent
binary16	11	5
binary32	24	8
binary64	53	11
binary128	113	19

Roughly doubling the number of bits / digits, roughly bumping the exponent by 1.5×.

And bfloat16 doesn't fit that model… It's not the only one.

bfloat16 8 8

The landscape is changing rapidly.

Mixing precisions

DSDOT: BLAS 1! 1979!

- C. L. Lawson, R. J. Hanson, D. R. Kincaid and F. T. Krogh, Basic linear algebra subprograms for Fortran usage, Algorithm No. 539, Transactions on Mathematical Software 5, 3 (September 1979), pp. 308-323.
- DOUBLE accumulator for REAL data.
- Subject to Fortran compiler flags.

XBLAS: 2008.

- X stands for explosion.
- All combinations of internal and external precisions.
- Not maintainable, not widely adopted.

But people *want* to use the smaller, faster precisions.

- Occasionally larger precisions as well.

Moving precisions and contexts

More than just ubiquitous parallelism.

- We've had vector machines for a long time.
  - And hated debugging them for at least as long.
- Kogge: Make sparse support vector machines scalable with remote atomic FP. (CRNCH Summit 2020)
- Eckert, Fujiki, *et al.* (U Mich): Abuse cache SRAM into being a vector unit! (ARM Research Summit 2019)

How do these interact with the contexts (rounding, exceptions)?

- My preference: All are per-operation. Hardware folks laugh.

Telescoping precisions and more

- IEEE binary32: 24×24 bit multiplier
- bfloat16: 8×8 multiplier
  - Around nine fit in the same space / energy.
  - Order of magnitude!
- So build a DGEMM from multiple tiny floats...
  - If the data is “nice.”
  - (Ongoing work by Greg Henry)

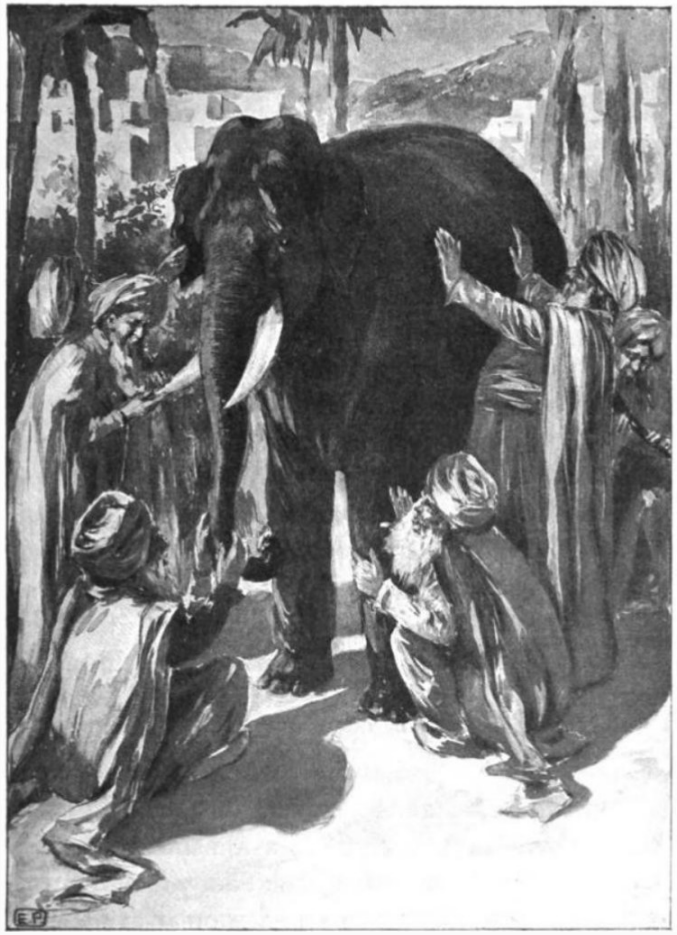
But wait, there’s more...

- Residue number systems for reliability at low power: CREEPY
- Reproducibility similar to double-double: ReproBLAS

The elephant in the room

Machine learning.

- What is being computed?
- What accuracy is needed?
- Do we know what we're doing?
  - (I have started seeing numerical analysis.)
- Will five-eight years be enough for convergence?
  - (Vanishing gradients are an issue.)

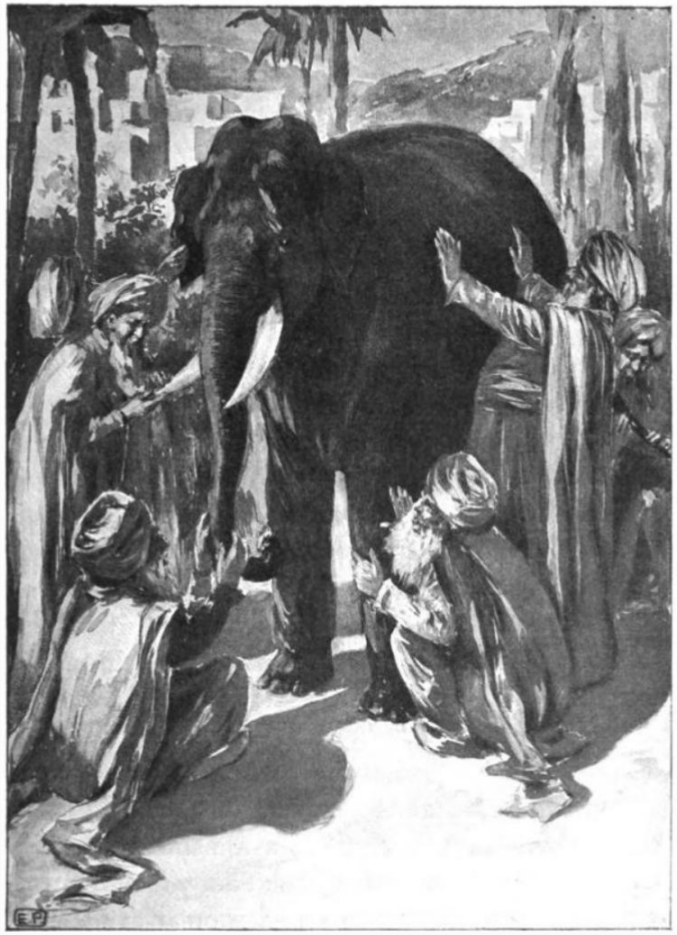


(Public domain from Wikipedia)

The other elephant in the room

Missing data.

- Many mechanisms exist for coping with missing data.
- Should we standardize some NaN to be missing?
  - IIRC, R uses a NaN.
- How does this impact the rest of the standard?
  - NaN propagation always is a sticking point.



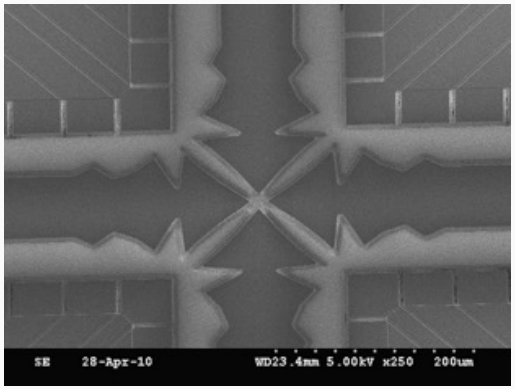
(Public domain from Wikipedia)

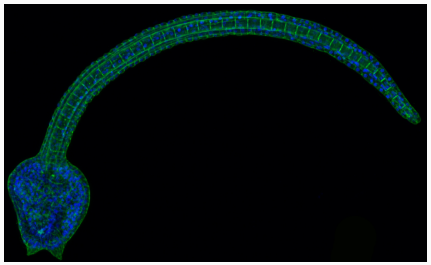
And hardware is (de?)evolving

Quantum, neuromorphic, analog, and more...

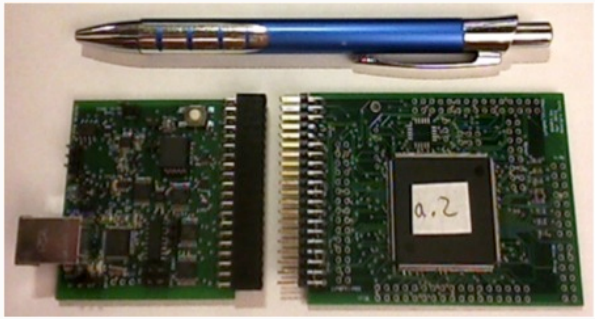
Stiff ODEs: Just implement them!

Quantum.





FPAA



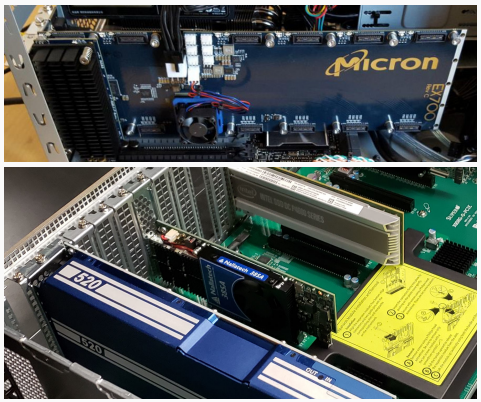
Introducing the CRNCH Rogues Gallery

Georgia Tech Center for Research into Novel Computing Hierarchies

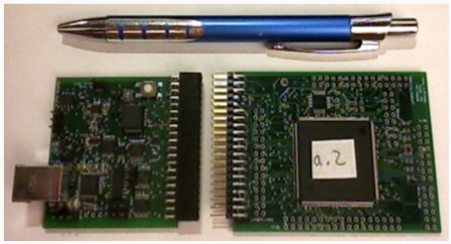
A physical & virtual space for hosting novel computing architectures, systems, and accelerators.

Emu Chick

FPGAs & HMC/HBM



FPAA



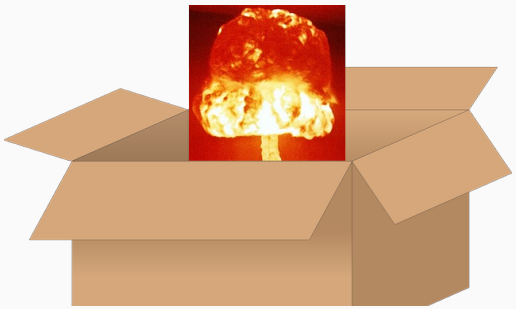
Amortize effort and cost of trying novel architectures.

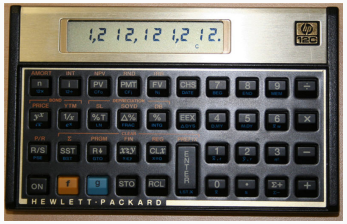
Break the “but it’s too much work” barrier.

`http://crnch.gatech.edu/rogues-gallery`

(Some) kinds of reproducibility

- Debugging
  - Just developer's platform.
  - Then users occur.
- Investigate rare instance
  - Small job, similar to debugging.
  - Larger? # proc changes.
- Schrödinger's nuke, climate negotiations, ...
  - Likely little control over the runtime environment.
- Accounting, some finance
  - Legal: identical across history.





Assuming “agreement” on exceptions…

(Some) current approaches:

- Specific platform reproducibility for debugging.
  - Intel CNR, NVIDIA
- Arbitrary precision / exact comp.
  - Not saying more on this.
- Correctly *rounded* results
  - ExBLAS
  - *Not* “faithful” rounding. One of *two* choices, but another implementation may choose the other.
- Reproducible accumulators
  - Very wide accumulators (Kulisch, ARM HPA)
  - Binned accumulators (ReproBLAS)

What should IEEE 754 do to support reproducibility?

We adopted “twoSum” as a recommended operation.

Emulating augmentedAddition as two instructions improves double-double:

Operation		Skylake	Haswell
Addition	latency	−55%	−45%
	throughput	+36%	+18%

DDGEMM MFLOP/s from reduced insn dependencies:

Operation	Intel Skylake	Intel Haswell
“Typical” implementation	1732 ($\approx 1/37$ DP)	1199 ($\approx 1/45$ DP)
Two-insn augmentedAddition	3344 ($\approx 1/19$ DP)	2283 ($\approx 1/24$ DP)

Dukhan, Riedy, Vuduc. “Wanted: Floating-point add round-off error instruction,” PMAA 2016, ArXiv 1603.00491.

ReproBLAS dot product: 33% rate improvement,
only 2× slower than non-reproducible.

Formal modeling

- If IEEE 754 is moving to the language level…
  - Should we standardize operational semantics?
  - Provide a little language for mapping?
  - Procedural, declarative, … options.
- Notably: The Coq proof environment has been used to model IEEE 754 formally in Flocq.

Education: Have we written ourselves out?

- That recruiting bit is a “ha ha only serious” aspect.
- No students participated in 754-2019.
- Is the standard too good? Nothing needed?
- No one needs to think about these issues now?

Or are we (um, me) just asking the wrong questions?

How do we grow the FP community?