Tutorial Lecture: Advanced Posit Arithmetic

Prof. John L. Gustafson

A*STAR and National University of Singapore

26 March 2019

Decimal Error and Decimal Accuracy

Decimal Error:

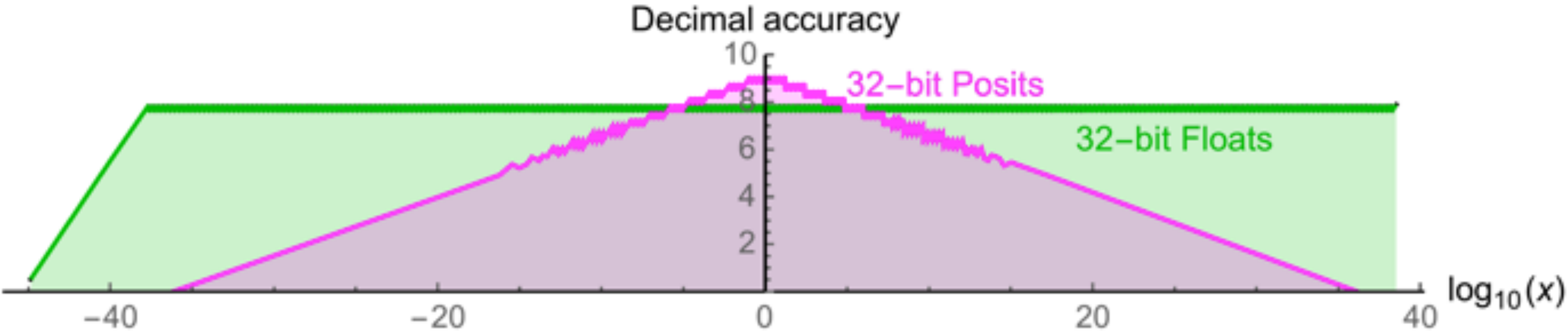
$$|\log_{10}(x_{\text{computed}}) - \log_{10}(x_{\text{exact}})| = |\log_{10}(x_{\text{computed}}/x_{\text{exact}})|$$

Decimal Accuracy:

$$\log_{10}(1/\text{Decimal Error}) = -\log_{10}(|\log_{10}(x_{\text{computed}} / x_{\text{exact}})|)$$

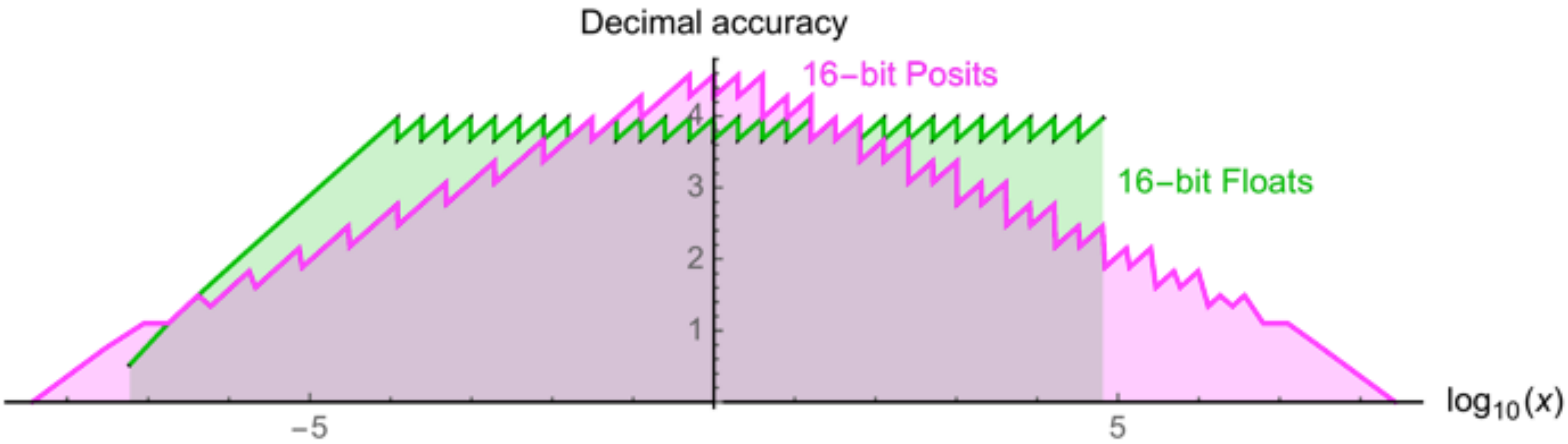
Note: Underflow to zero and overflow to infinity commit ***infinitely large*** decimal errors.

Tapered Accuracy Plots: 32-bit



32-bit posits appear ideal for most HPC tasks that presently use 64-bit floats. Properly used, 32-bit posits maintain 8 correct decimals, more than enough accuracy for the *answer.*

Tapered Accuracy Plots: 16-bit



16-bit posits seems well-suited to signal processing, with more dynamic range than floats. Signals can often be normalized to stay in the "sweet spot" where accuracy is highest.

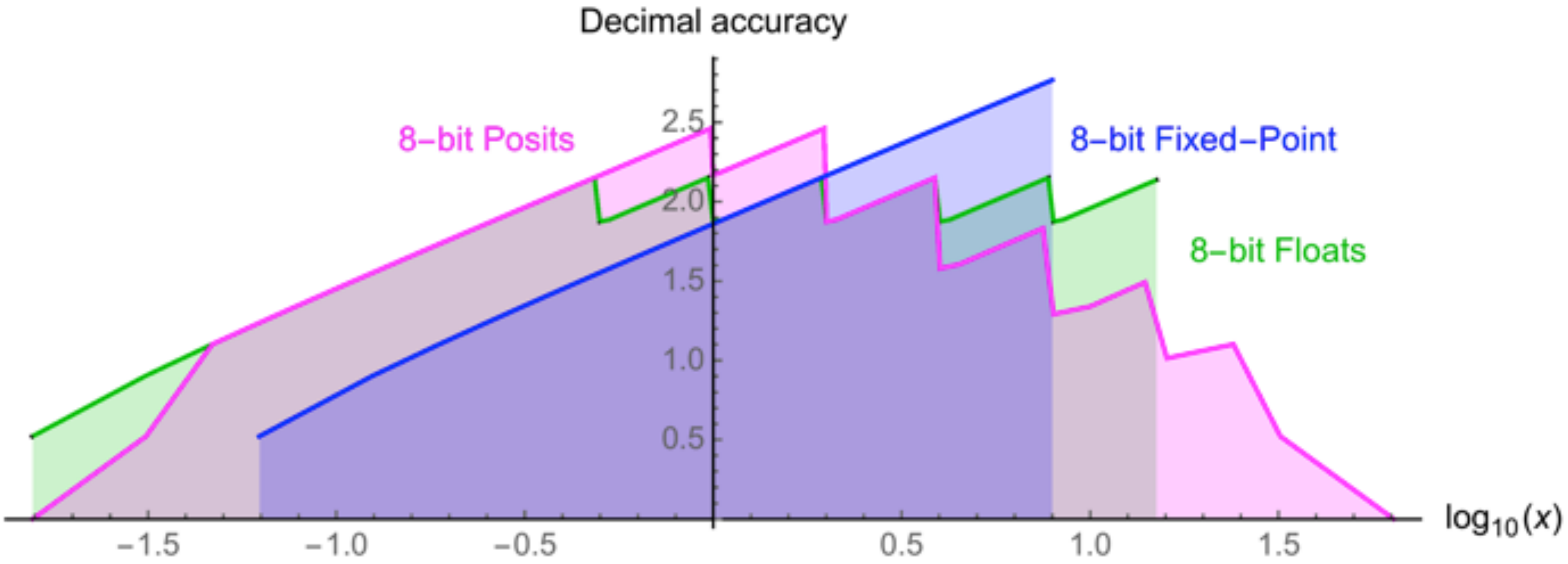
Decimal Accuracy: Floats versus Posits

Size, bits	Float maximum accuracy, bits	Posit maximum accuracy, bits	Posit accuracy advantage	Where posit accuracy is ≥ float accuracy
8	N.A.	6	N.A.	N.A.
16	11	13	0.6 decimals	$^{1}/_{64}$ to 64
32	24	28	1.2 decimals	10^{-6} to 10^{6}
64	53	59	1.8 decimals	1.4×10^{-17} to 7.2×10^{16}

Dynamic Range: Floats versus Posits

Size, bits	IEEE Standard float exponent size, bits	Float dynamic range	Draft Standard posit exponent size, bits	Posit dynamic range
8	(3)	($1/64$ to 16)	0	$1/64$ to 64
16	5	6×10^{-8} to 7×10^{4}	1	3.7×10^{-9} to 2.7×10^{8}
32	8	1.4×10^{-45} to 3×10^{38}	2	7×10^{-37} to 1.3×10^{36}
64	11	5×10^{-324} to 2×10^{308}	3	5×10^{-150} to 2×10^{149}

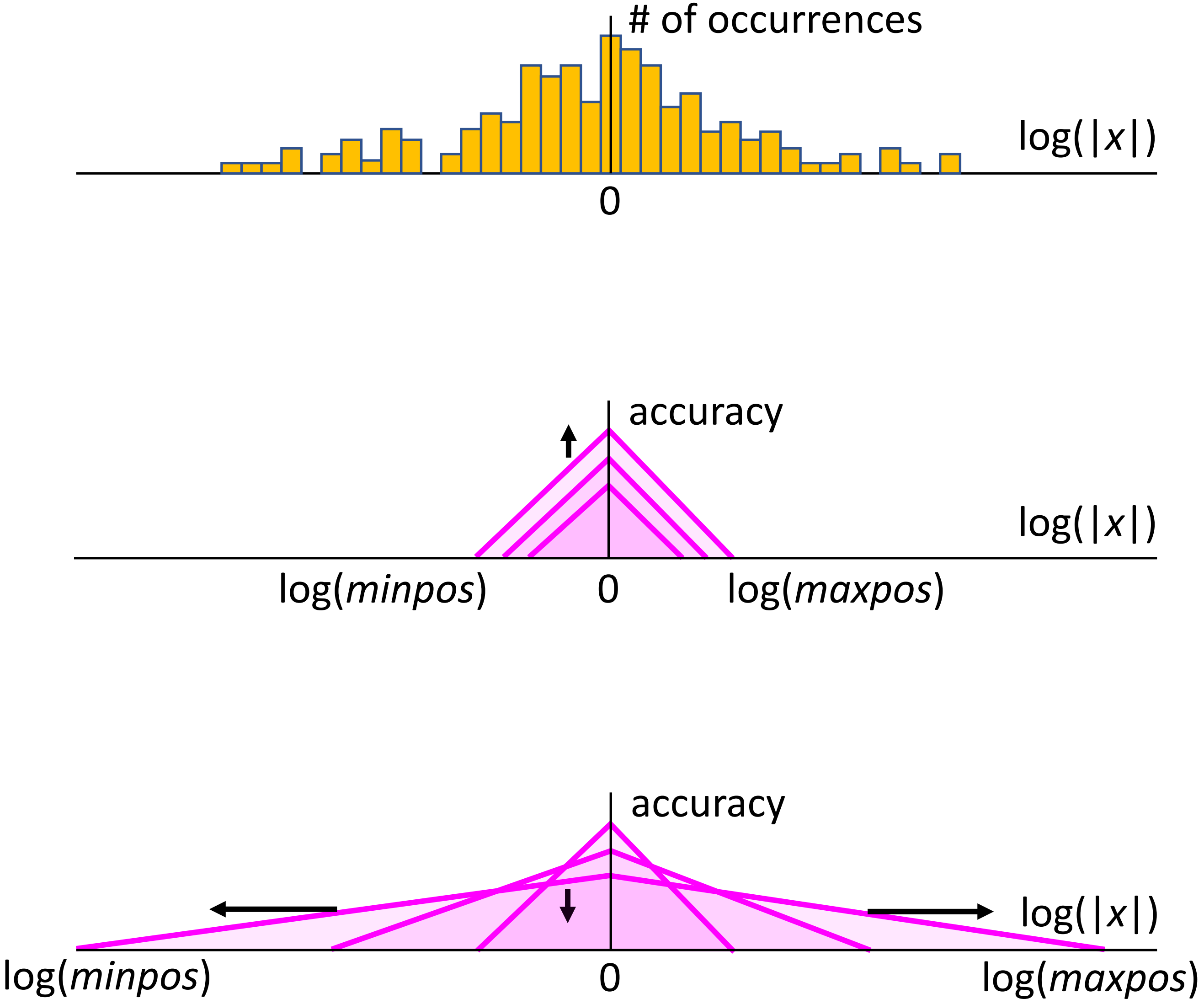
What about fixed-point decimal accuracy?



Fixed point makes sense when you need uniform *absolute* error. The ramp shows how *relative* accuracy is not maximized by using fixed-point.

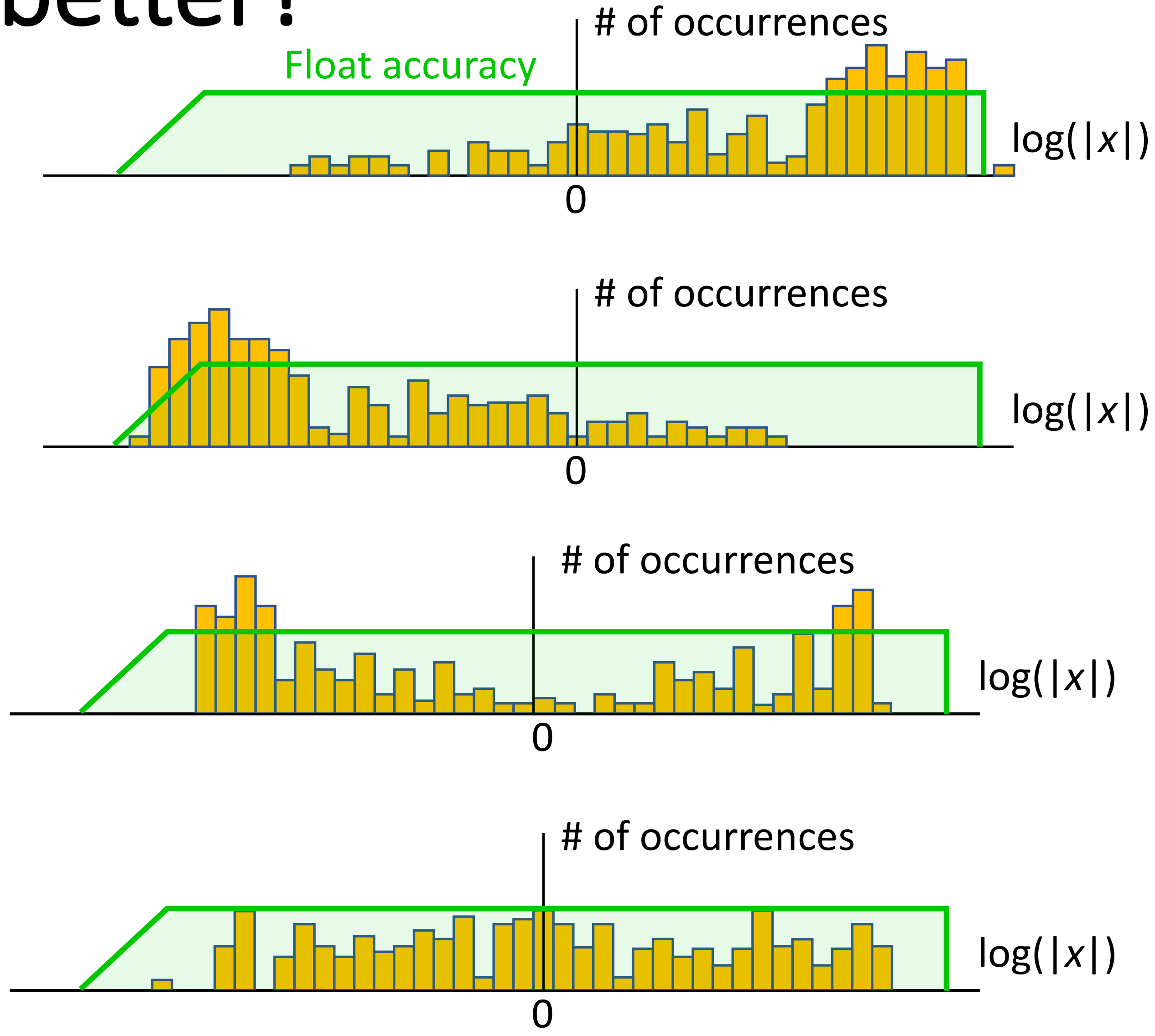
Flexible size posits

- For any application, examine the histogram of magnitudes.
- Raising *ps* raises the “tent” accuracy plot (and widens it).
- Raising *es* doubles tent width (and lowers accuracy).
- Software and FPGAs need not follow Draft Posit Standard; you can customize to match the application requirements.



When might floats do better?

- When histogram is skewed to the large or small magnitudes. This can be corrected by rescaling.
- When histogram needs both large **and** small values at high accuracy, or the histogram looks more like a box than a tent. Hard to correct, but unusual for real applications.



Disadvantages of the posit format

- Fast hardware implementations not yet in conventional CPUs
- Incomplete math libraries for all *ps*, *es* sizes (cos, log, etc.)
- Count Leading Zeros (CLZ) operation needed for every input. (Floats need CLZ only for subnormal inputs. Disadvantage applies to software, not hardware.)
- If a computation wanders into very large or very small magnitude numbers, accuracy can be **less** than for floats.
- A float algorithm that “converges” when a value **underflows to zero** needs to be altered to work for posits.

The Quire

quire | kwī(ə)r |

noun

four sheets of paper or parchment folded to form eight leaves, as in medieval manuscripts.

- any collection of leaves one within another in a manuscript or book.

The quire is based on the Exact Dot Product (EDP) of Ulrich Kulisch and provides a **lifeline to exact mathematics**. It is a fixed-point 2's complement format, with one exception value (NaR = 1000⋯000).

Quire sizes for standard posits is ½ ps^2

Posit size (*ps*)	*es* value	Quire size in bits
8	0	32
16	1	128
32	2	512
64	3	2048

Note that for 32-bit posits, the quire is the same size as a typical cache line, or the SSE data width.

64-bit posits should *very* rarely be needed. (32-bit posits can replace them in most cases.)

Exact dot products built for 64-bit IEEE floats are less practical; their size is not a power of 2, and they are over 4000 bits long.

Fused operations should never be *covert*

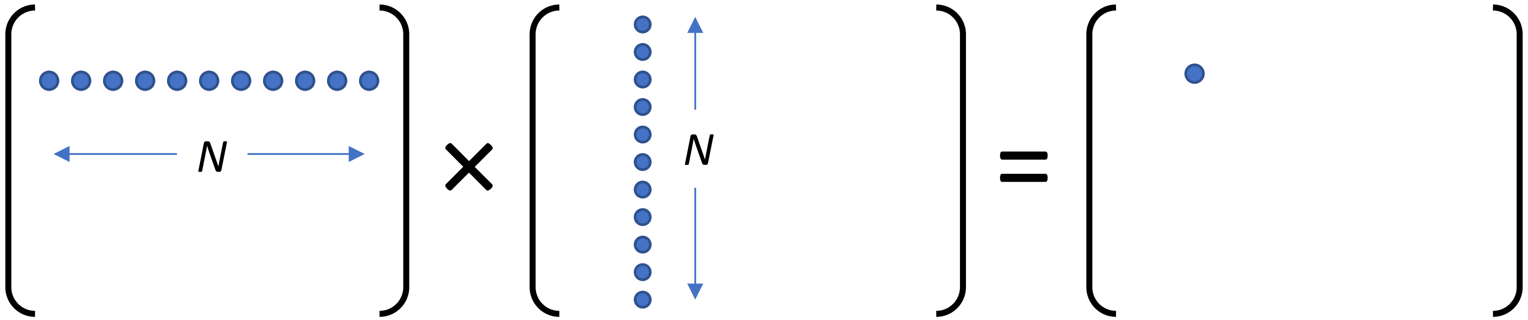
```
F(x,y):= sqrt(x*x - y*y)
```

Compiler tries to improve accuracy by using fused multiply-add:

```
Reg1=x*x; // this rounds down half the time.
Reg2=Reg1-y*y; //fused; y*y doesn't round
Return sqrt(Reg2);
```

If **`Reg1`** holds a value that was rounded down, **`Reg1-y*y`** will be negative since **`y*y`** doesn't round! So **`Reg2`** is a negative number with no real square root. **FAIL**

Quire benefits for matrix multiplication



- *N* rounding errors per dot product become 1 rounding error.
  - If errors are statistically independent, quire is $\sqrt{N}$ times more accurate.
  - For HPC-size problems, this can easily add two decimal places more accuracy.
- BLAS routine SDOT is easily replaced with a quire fused dot product
- For parallel algorithms, use quire for **partial** summations

Examples of what posits can do

Fluid dynamics, linear algebra, FFTs, neural networks

An independent test by LLNL

- Shock wave passing through initially quiescent L-shaped chamber
- Ideal gas Euler equations

$$\partial_t u + \nabla \cdot F(u) = 0$$

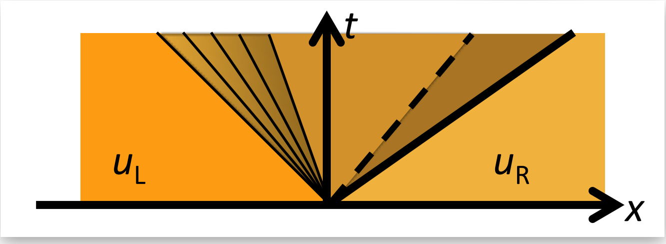
$$u = \begin{pmatrix} \rho \\ \rho v \\ \rho E \end{pmatrix} \qquad F(u) = \begin{pmatrix} \rho v \\ \rho v \otimes v + p \\ \rho v H \end{pmatrix}$$

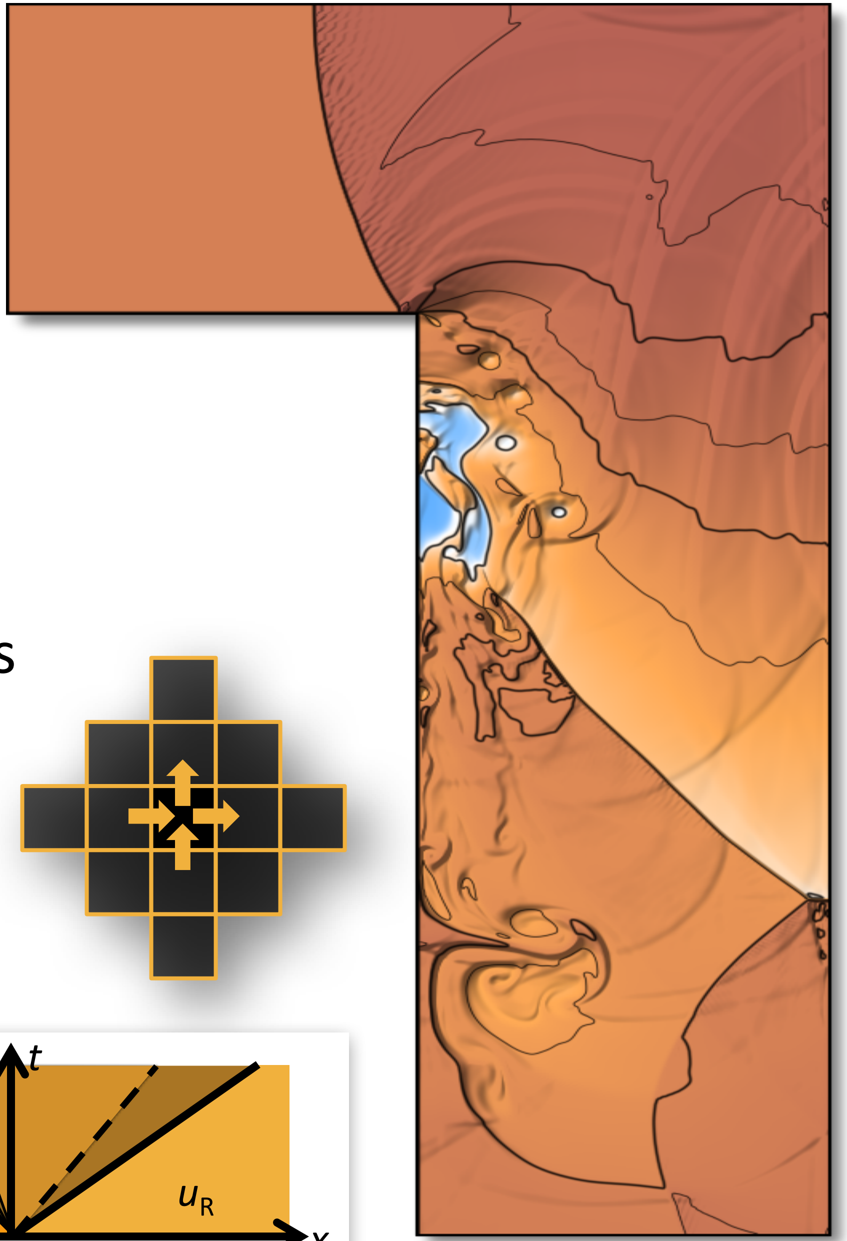
$$\rho E = \frac{p}{\gamma - 1} + \frac{1}{2}|v|^2 \qquad \rho H = \rho E + p$$

$$u_{\mathbf{i}}^{n} = u_{\mathbf{i}}^{n} - \frac{\Delta t}{\Delta x} \sum_{d=1}^{2} \left[F^d_{\mathbf{i}+\frac{1}{2}\mathbf{e}^d} - F^d_{\mathbf{i}-\frac{1}{2}\mathbf{e}^d} \right]$$

- Explicit finite volume discretization
- High-resolution Godunov solver

Uniform grid:
512×256 +
256×768 cells

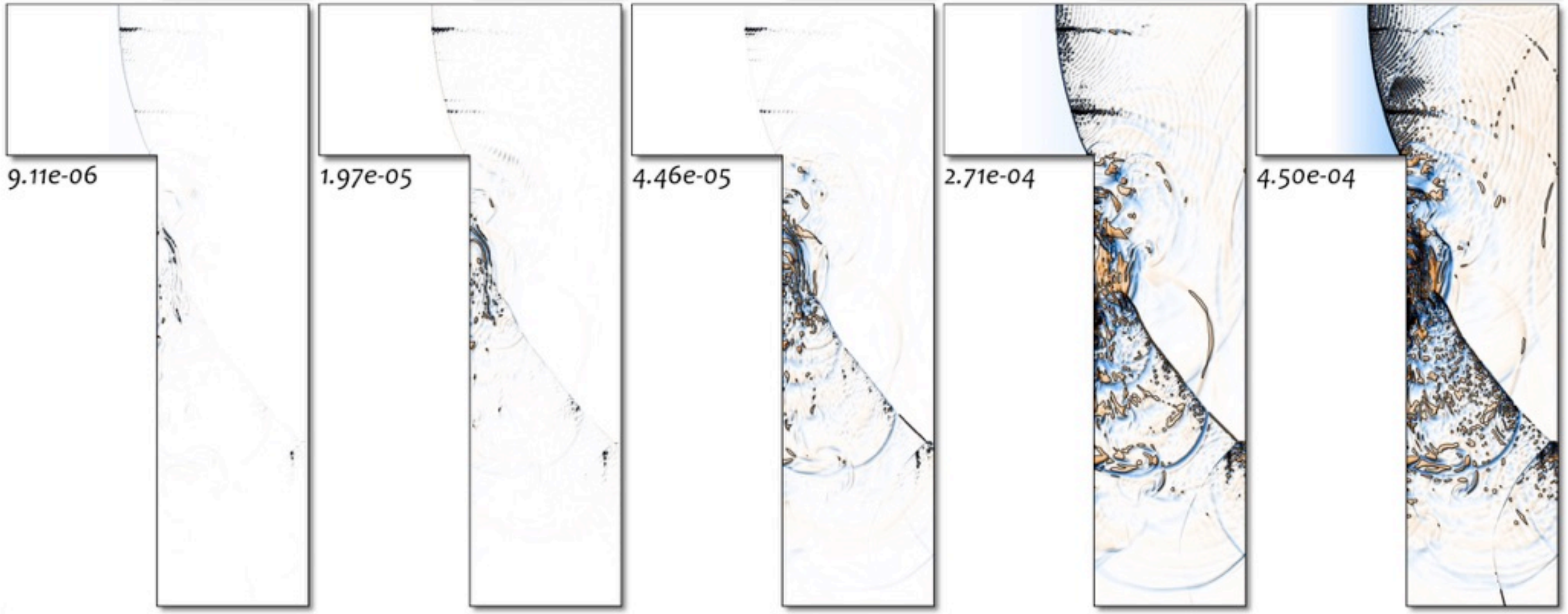




3rd Party Validation of Posits

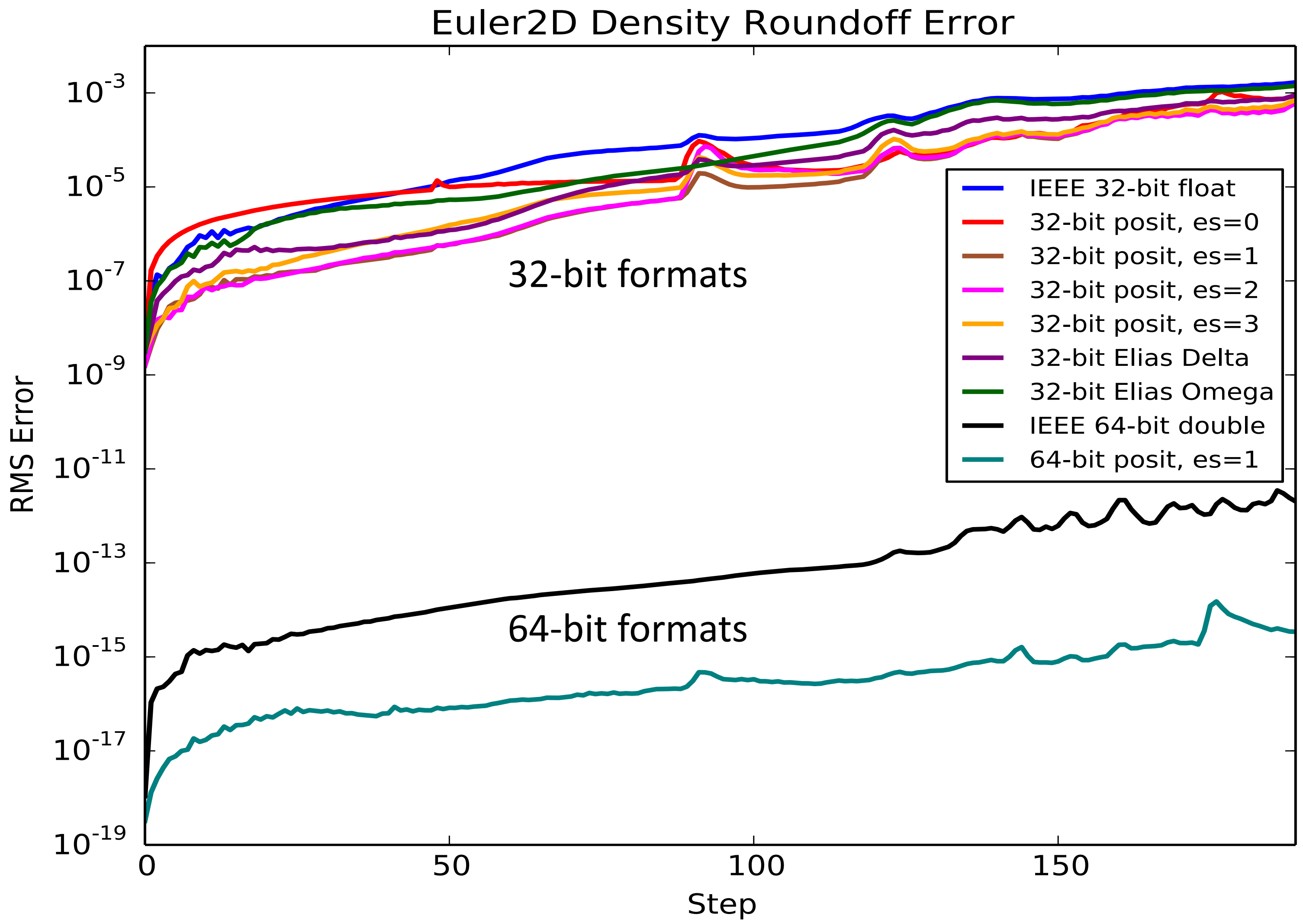
From Lawrence Livermore National Laboratory, with permission. Simulation of a fluid shock wave in an L-shaped region:

Errors for six different 32-bit formats (white = no error):



Posits are ***50x more accurate*** than floats, the best of any format tested.

They also compared 64-bit floats and posits



"Posits are two orders of magnitude more accurate than floats" — *Stephen Lindstrom, LLNL*

Challenge: Invert a Hilbert Matrix Numerically

In least-squares curve fitting, *n*-by-*n* Hilbert matrices $\mathbf{H}_n$ arise

They're *brutally* ill-conditioned (nearly singular). This one has a condition number of **~5×10⁵**. The exact inverse is all integers.

$$\mathbf{H}_5 = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

$$h_{i,j} = \frac{1}{i+j-1}$$

Scale both sides by 5×7×9 = 315 so the matrix can be **expressed exactly** by both floats and posits.

$$315 \cdot \mathbf{H}_5 = \begin{bmatrix} 315. & 157.5 & 105. & 78.75 & 63. \\ 157.5 & 105. & 78.75 & 63. & 52.5 \\ 105. & 78.75 & 63. & 52.5 & 45. \\ 78.75 & 63. & 52.5 & 45. & 39.375 \\ 63. & 52.5 & 45. & 39.375 & 35. \end{bmatrix}$$

Let's try 32-bit IEEE floats, and then 32-bit posits.

Use LDL^T decomposition

Float result has worst-case accuracy of only 3.7 decimals correct:

$$\text{Float } \mathbf{H}_5^{-1} = \begin{bmatrix} 25.0056\cdots & -300.100\cdots & 1050.42\cdots & -1400.62\cdots & 630.30\cdots \\ -300.100\cdots & 4801.79\cdots & -18907.6\cdots & 26891.3\cdots & -12605.4\cdots \\ 1050.42\cdots & -18907.6\cdots & 79411.9\cdots & -117647.\cdots & 56722.9\cdots \\ -1400.62\cdots & 26891.3\cdots & -117647.\cdots & 179270.\cdots & -88234.1\cdots \\ 630.30\cdots & -12605.4\cdots & 56722.9\cdots & -88234.1\cdots & 44116.5\cdots \end{bmatrix}$$

Worst-case posit accuracy is 6.3 decimals, over **400 times** as accurate:

$$\text{Posit } \mathbf{H}_5^{-1} = \begin{bmatrix} 24.999994\cdots & -299.99992\cdots & 1049.9998\cdots & -1400.00003\cdots & 630.0001\cdots \\ -299.99992\cdots & 4799.9993\cdots & -18900.0009\cdots & 26880.007\cdots & -12600.005\cdots \\ 1049.9998\cdots & -18900.0009\cdots & 79380.02\cdots & -117600.06\cdots & 56700.04\cdots \\ -1400.00003\cdots & 26880.007\cdots & -117600.06\cdots & 179200.1\cdots & -88200.08\cdots \\ 630.0001\cdots & -12600.005\cdots & 56700.04\cdots & -88200.08\cdots & 44100.04\cdots \end{bmatrix}$$

The posit advantage typically far exceeds what you'd expect from having a few extra bits in the fraction (27 bits versus 23 bits, here)

But posits have a secret weapon, the *quire*

Compute the posit inverse times $\mathbf{H}_5$ *exactly*, using the quire. Compare with the identity matrix (i.e., compute the *residual*), and use that as a correction to the original calculation:

$$\text{Quire-corrected Posit } \mathbf{H}_5^{-1} = \begin{bmatrix} \mathbf{25} & \mathbf{-300} & \mathbf{1050} & \mathbf{-1400} & \mathbf{630} \\ \mathbf{-300} & \mathbf{4800} & \mathbf{-18900} & \mathbf{26880} & \mathbf{-12600} \\ \mathbf{1050} & \mathbf{-18900} & \mathbf{79380} & \mathbf{-117600} & \mathbf{56700} \\ \mathbf{-1400} & \mathbf{26880} & \mathbf{-117600} & \mathbf{179200} & \mathbf{-88200} \\ \mathbf{630} & \mathbf{-12600} & \mathbf{56700} & \mathbf{-88200} & \mathbf{44100} \end{bmatrix}$$

Worst-case error is… **zero**. There isn't any error. That's the **exact** inverse.

Floats cannot do this, even using double-precision.

An astonishing technique for maximum accuracy

Any expression using + – × / can be written as L*x* = *b* where L is lower triangular and the last x_n is the desired calculation. Example:

$$\boxed{f = (a + b) \times c - d / e}$$

$$
\begin{aligned}
x_1 &= a \\
x_2 &= x_1 + b \\
x_3 &= c \times x_2 \\
x_4 &= d \\
e \times x_5 &= x_4 \\
x_6 &= x_3 - x_5
\end{aligned}
\quad \Rightarrow \quad
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 \\
0 & c & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 & e & 0 \\
0 & 0 & -1 & 0 & 1 & 1
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix}
=
\begin{pmatrix} a \\ b \\ 0 \\ d \\ 0 \\ 0 \end{pmatrix}
$$

Solving for x using the quire lets us evaluate $f = x_6$ correctly rounded!

The quire allows "Karlsruhe Accurate Arithmetic." This removes most of the need for 64-bit format.

Ulrich Kulisch

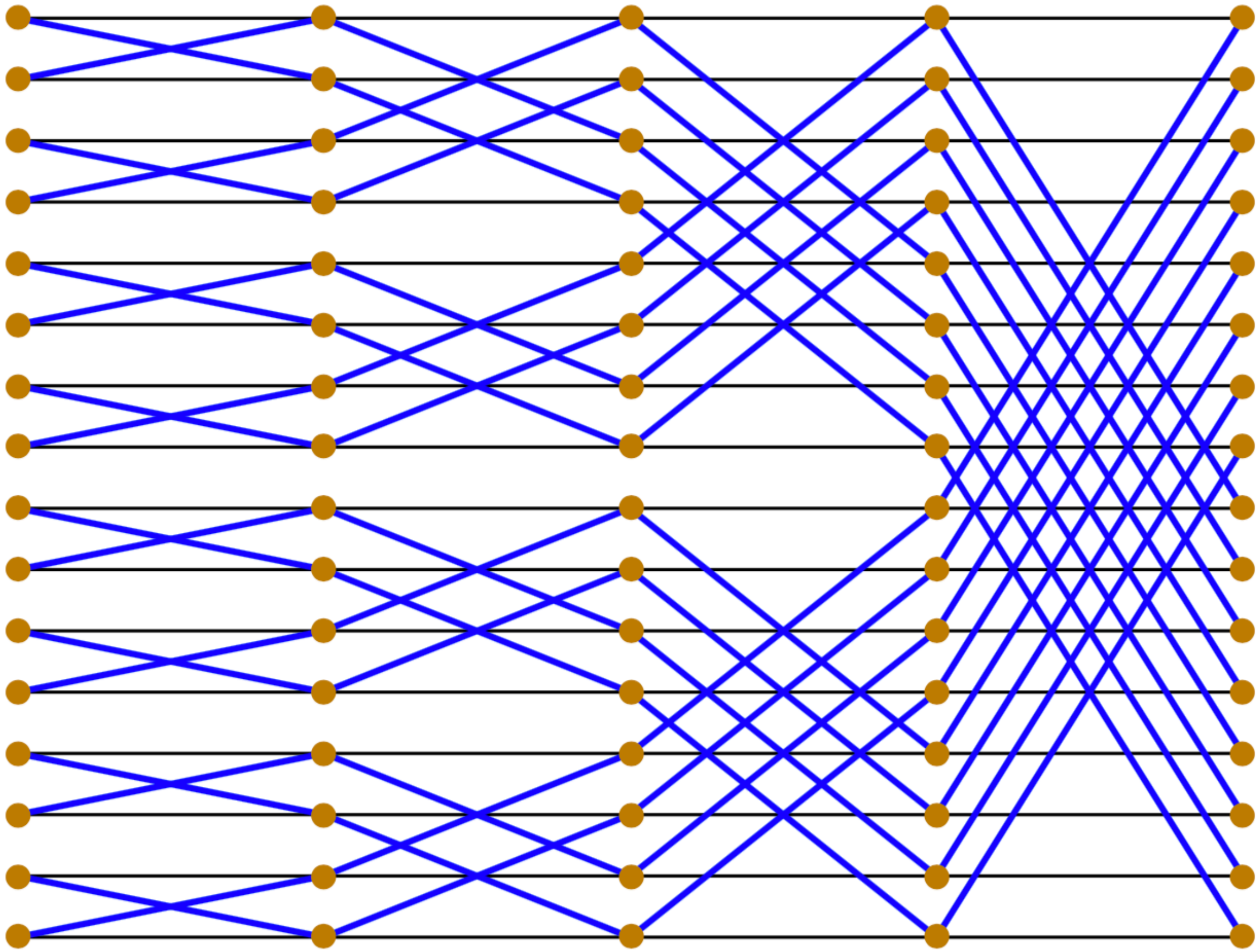
$$
\begin{aligned}
x_1 &= a \\
x_2 &= x_1 + b \\
x_3 &= c \times x_2 \\
x_4 &= d \\
e \times x_5 &= x_4 \\
x_6 &= x_3 - x_5
\end{aligned}
$$

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 \\
0 & c & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 & e & 0 \\
0 & 0 & -1 & 0 & 1 & 1
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix}
=
\begin{pmatrix} a \\ b \\ 0 \\ d \\ 0 \\ 0 \end{pmatrix}
$$

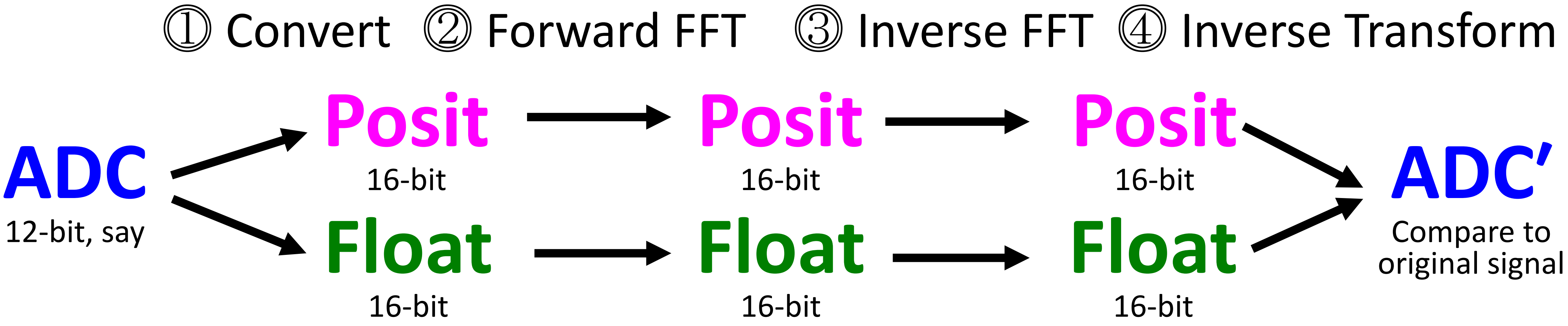
- Great for polynomials, even badly-conditioned ones
- Can be done *automatically by the compiler*.
- "XSC" languages apply this technique, but not to reduce data size.
- Idea: Let user **mark** the variables that need guaranteed accuracy, instead of applying everywhere.
- 32-bit posit answers can be **more accurate than 64-bit float answers**.

Posits for Fast Fourier Transforms (FFTs)

- FFTs are very *communication bound.*
- Most efforts (like FFTW) focus on lowest op count.
- More potent approach: reduce the data size by using posit format



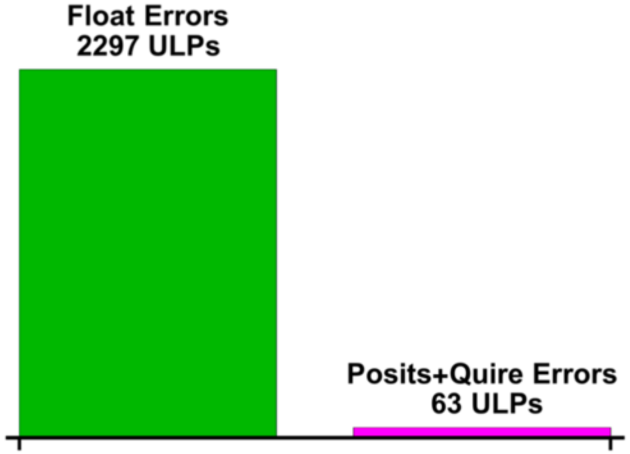
Methodology: Measure "round trip" error for 1024-point and 4096-point complex FFTs.



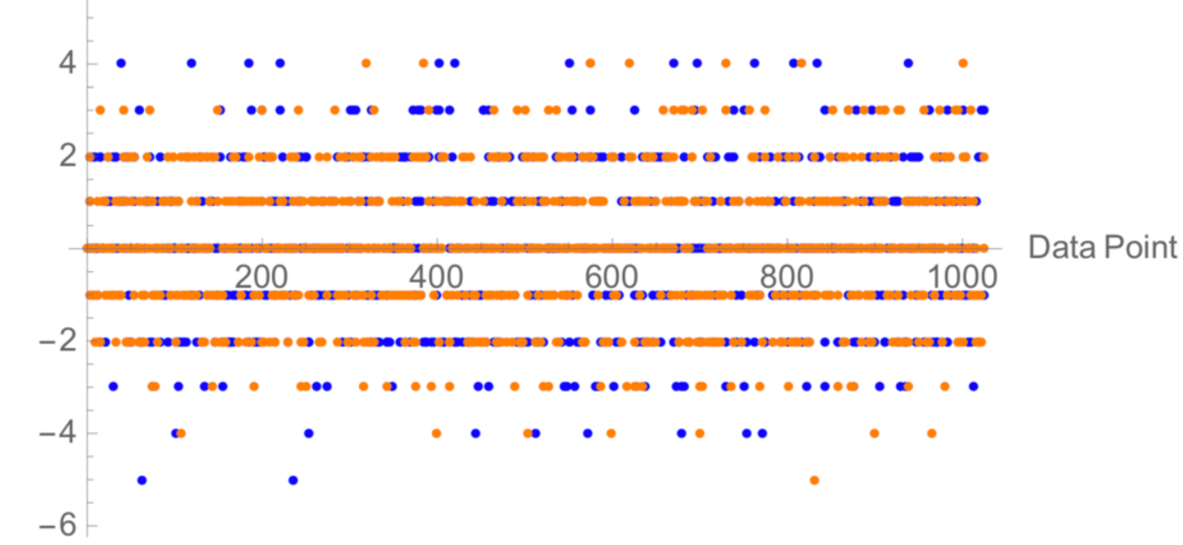
If a format can transform Analog-to-Digital Converter (ADC) signals forward and backward **without loss**, there is no need for higher precision.

16-bit posits can do this; 16-bit floats cannot.

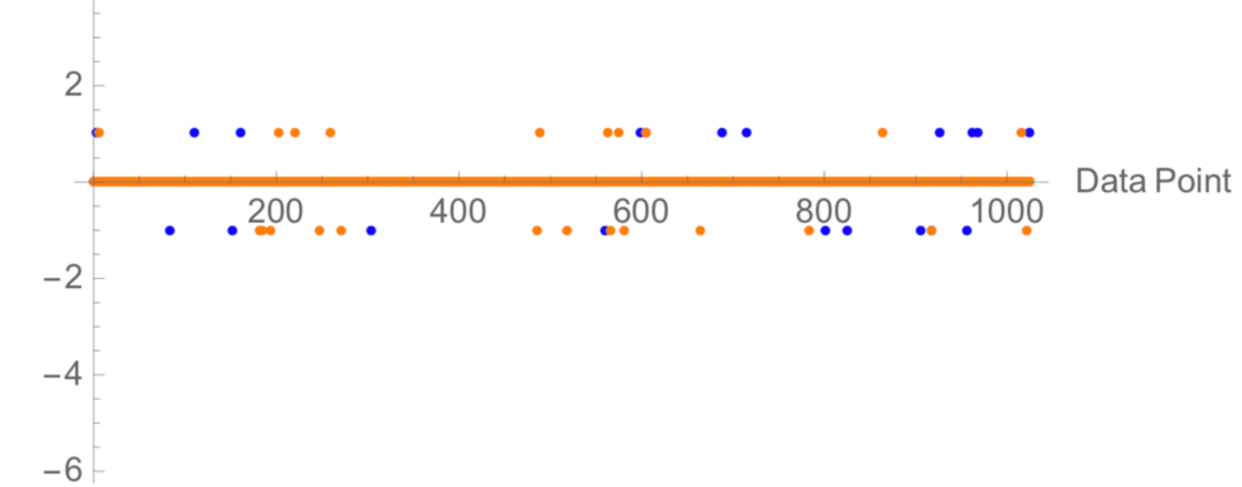
1024-point
Complex FFT
Round Trip



Floats only get 32% of the original values back



Posits get 98% of the original values back, and any errors are at most one ULP



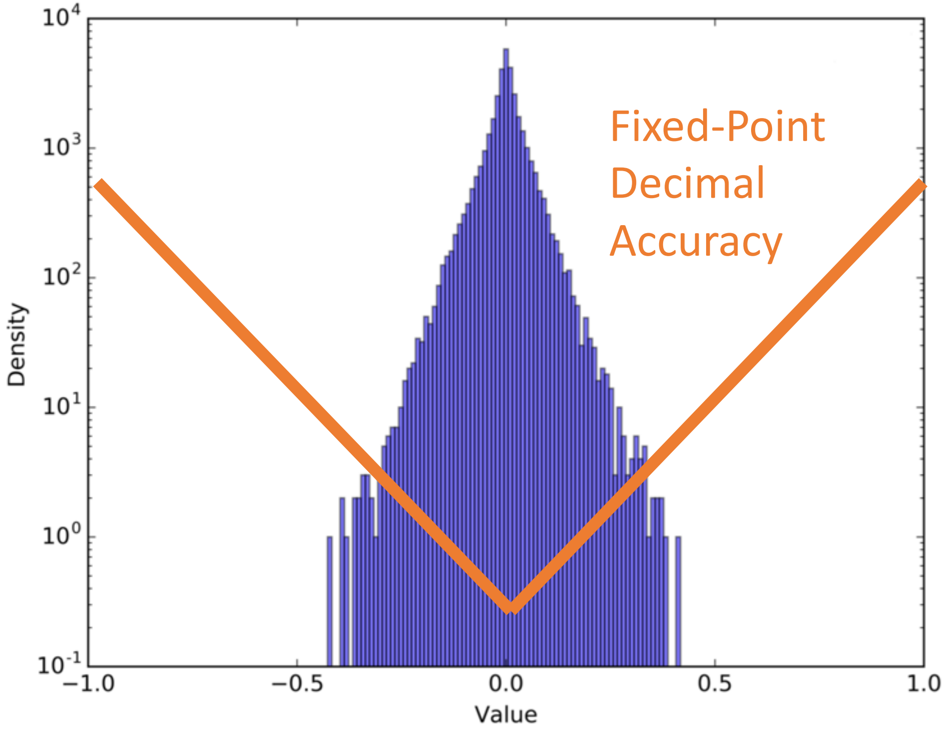
Neural Networks, both Training and Inference

Cifar-10, ConvNet study by RIT (simple 5-layer net, preliminary)

Number of bits	Posit Accuracy	Float Accuracy
8	*67.64%*	67.93%
7	**67.52%**	67.37%
6	**63.77%**	46.23%
5	31.18%	44.85%

Float32 accuracy is 68.45%.
8-bit Float used 3 exponent bits.
Quire was not used for posit results.

Histogram of values used in AlexNet



From "Deep Positron: A Deep Neural Network Using the Posit Number System," D. Kudithipudi, J. Gustafson, H. Langroudi, Z. Carmichael

This shows why fixed-point is **not** the best choice.

Summary

- Posits can fix float shortcomings, for AI and HPC:
  - Taper the accuracy for more info-per-bit.
  - Simplify exceptions, rounding, unused features
  - Match dynamic range to application needs
  - Make the exact dot product practical
  - Restore associative, distributive laws
  - Make calculations bitwise-reproducible
- Faster; reduces communication costs
- More accurate, especially if the quire is used
- Lower energy and power
- Less chip area

End of
Advanced Posit Tutorial
Lecture