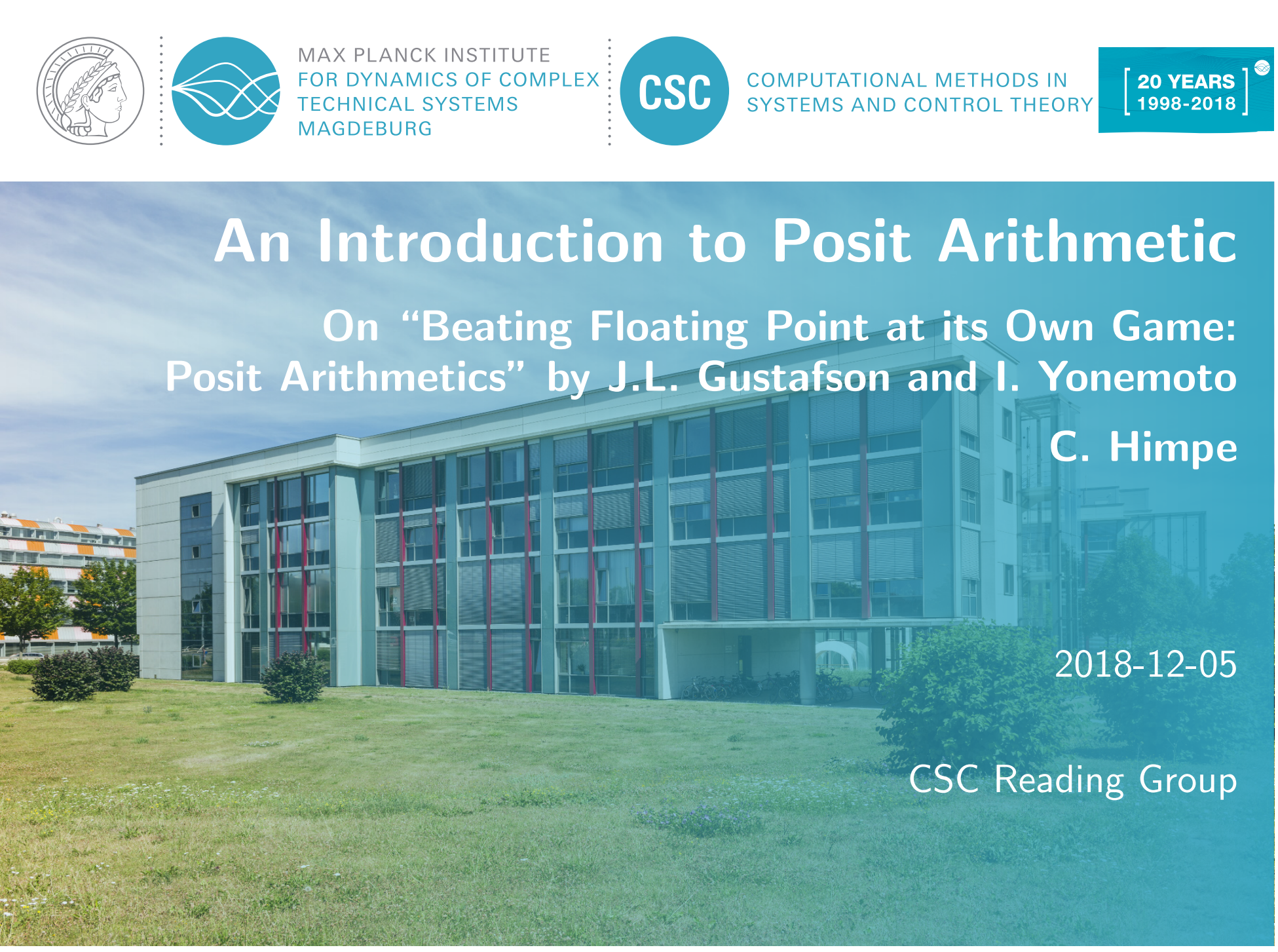


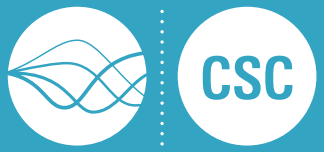
An Introduction to Posit Arithmetic

On "Beating Floating Point at its Own Game: Posit Arithmetics" by J.L. Gustafson and I. Yonemoto

C. Himpe

2018-12-05

CSC Reading Group



SNAFU – Simulation Normal ...

- Exascale Challenges: Memory & Power.
- Let's look at the fundamental layer: Numbers.
- Floats are completely bonkers!
- This is **not** a computer science, ...
- ... but a **scientific computing** ($\rightarrow$ math) issue!

Floating Point Craziness

Coarse Dynamic range of 10^{-324} to 10^{308} (DP).

- Number of neurons in the brain: $\sim 10^{14}$
- Age of the universe [ns]: $\sim 10^{25}$
- Planck Scale: $\sim 10^{-35}$
- Atoms in the universe: $\sim 10^{82}$

Wasteful Many different bit patterns for NaN (Not-a-Number).

- Half Precision (16-bit): $\sim 10^{3}$
- Single Precision (32-bit): $\sim 10^{7}$
- Double Precision (64-bit): $\sim 10^{15}$

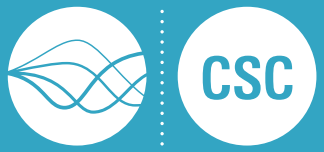
Ambivalent Why is there a -0 ? Should $\sqrt{-0} = -0$?

Weird $\frac{1}{0} = \infty$ and $\frac{1}{-0} = -\infty$, but $0 = -0$, hence $\infty = -\infty$?

Inconsistent `(NaN==NaN)=0`, but `(0==-0)=1` ? `bool(NaN)=1` !

Arbitrary IEEE754 is more a guideline than a standard.

LOL "Extended Precision" (80-bit)



An Alternative Number Format

Unum[1] **History**:

2015 Unum I (Universal NUMber): Superset of floats

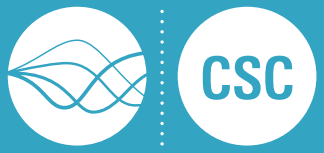
2016 Unum II: Float incompatible interval arithmetic, variable size

2017 Unum III (aka **Posit**): Float related and fixed size

Why Posits:

- Unum I & Unum II have varying (bit) length.
- Fix the worst flaws of floats and improve accuracy.
- A drop-in replacement for floats.
- A "modern" number format.
- No marketing or engineering hacks.

[1] J.L. Gustafson. **The End of Error: Unum Computing**. Chapman and Hall/CRC, 2015



Posits

posit (verb) *Something that is posited.* (Wiktionary)

posit (noun) *To suggest something as a basic fact or principle from which a further idea is formed or developed.* (Camebridge Dictionary)

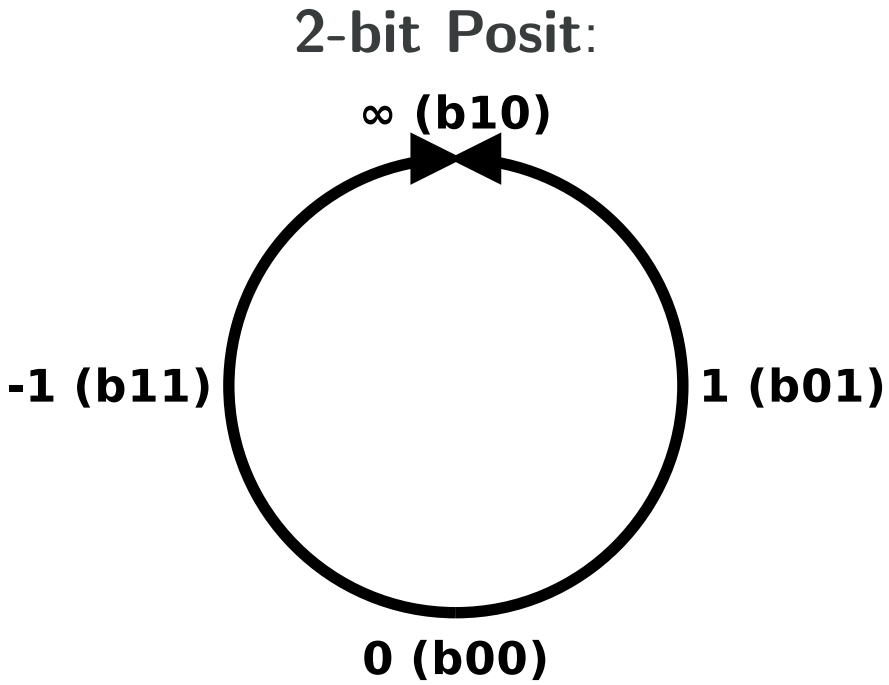
About Non-interval number format with encoding similar to floats.

Note POsit OPerations (POPs) instead of FLOPs.

Features

- More accurate.
- Less wasteful.
- More operations per Watt ($\rightarrow$ Power Aware COmputing).
- Simple hardware implementation.
- Every bit pattern has a unique meaning.
- Only two special "numbers", no NaNs.

Projective Mapping

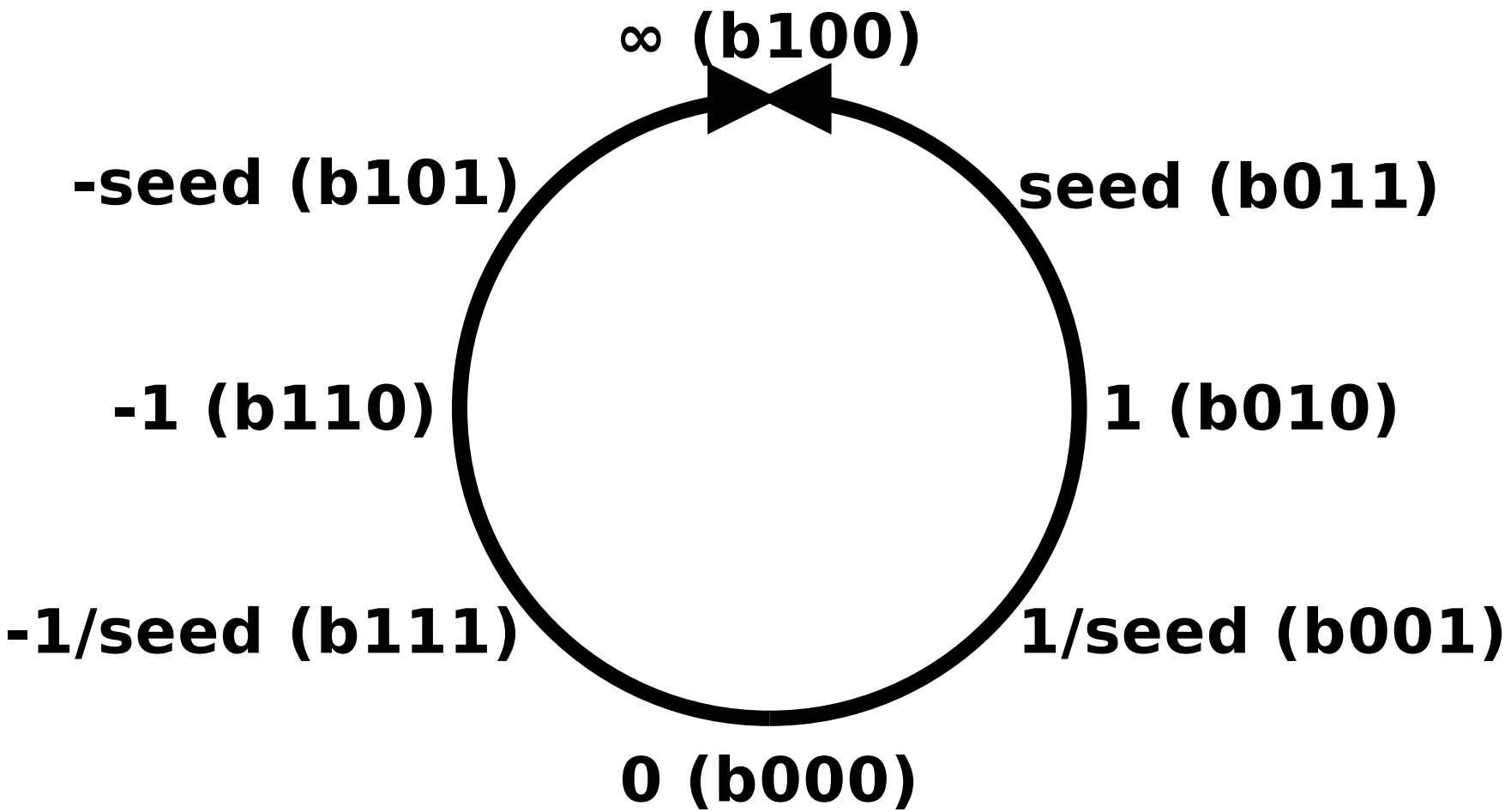


Remember Complex Analysis:

- Stereographic projection of the complex plane.
- Projective completion: Point at infinity[2].
- Posits are a discrete representation of the real projective line.

[2] This also "fixes" division by zero (`https://www.1dividedby0.com`).

Generic 3-bit Posit



Float Bit Pattern

Float:

0000000000000000000000000000000

Formula:

$$x = (-1)^{\text{sign}} \cdot 2^{\text{exponent}} \cdot (1 + \text{fraction})$$

Sign positive if zero, negative if one (one bit)

Exponent integer expressing power of two (fixed length)

Mantissa fractional remainder (fixed length)

Posit Bit Pattern

(e_s Exponent Size) Posit:

$$00\ldots000\ldots$$

Formula:

$$x = (-1)^{\text{sign}} \cdot \left(2^{2^{e_s}}\right)^{\text{regime}} \cdot 2^{\text{exponent}} \cdot (1 + \text{fraction})$$

$\mathbf{e_s}$	maximum number of exponent bits,
Seed	$2^{2^{e_s}}$
Sign	positive if zero, negative if one (one bit),
Regime	integer expressing power of power of two (varying length),
Exponent	integer expressing power of two (varying length),
Mantissa	(linear) fractional remainder (varying length),
Size	sizeof(Regime) + sizeof(Exponent) + size(Fraction) = constant

Regime Bits

Rationale:

- Determines "larger order of magnitude".
- Golomb-Rice type encoding.
- Algorithm: Count leading zeros. Available in:
  - Assembly (x86-32 `bsf`, x86-64 `lzcnt`, arm `clz`)
  - GNU (`__builtin_clz`)
  - Javascript (`Math.clz`)

Example (Four Regime Bits[3]):

$$\texttt{0000}: -4 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{-4} \qquad \texttt{10ZZ}: 0 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{0}$$

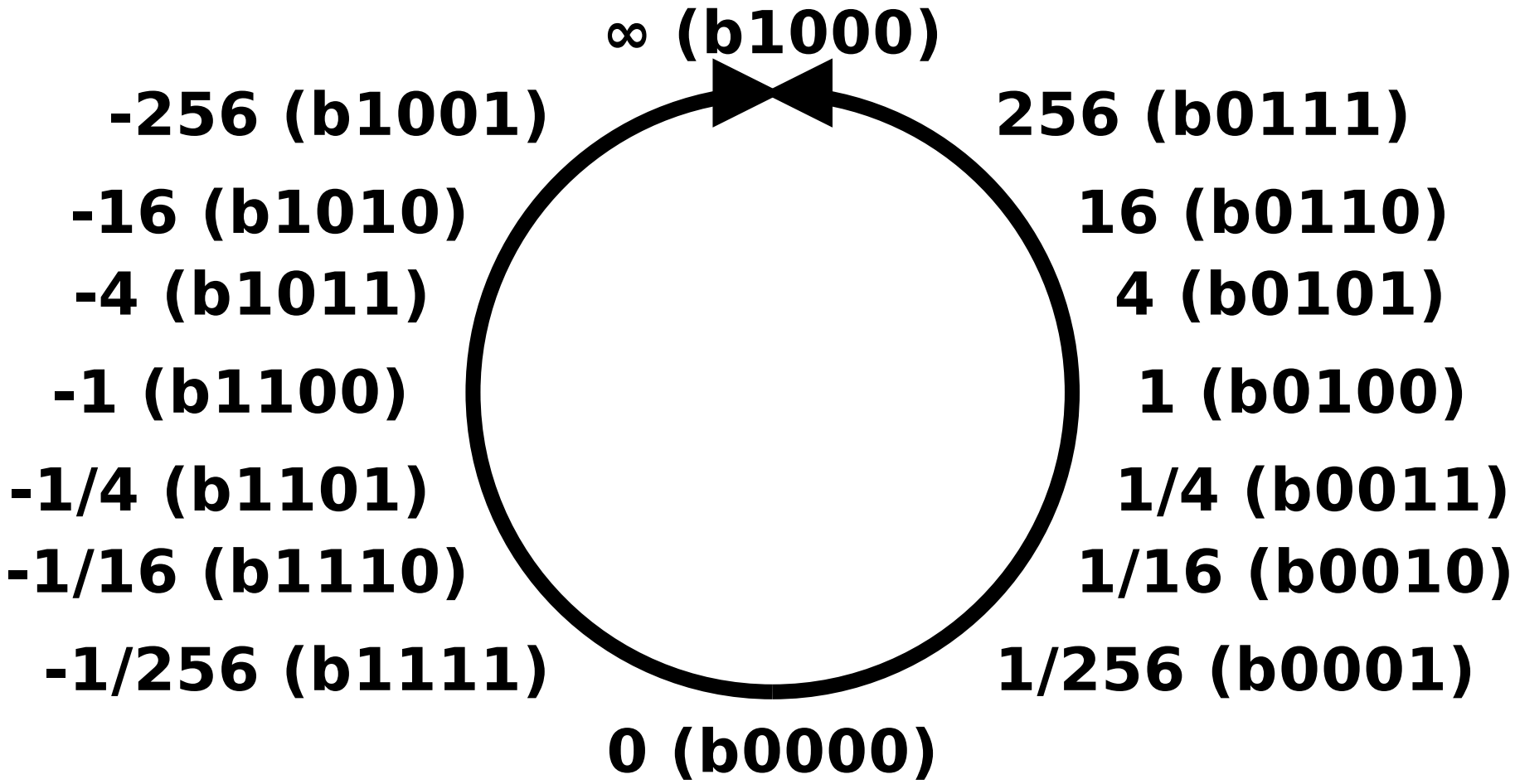
$$\texttt{0001}: -3 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{-3} \qquad \texttt{110Z}: 1 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{1}$$

$$\texttt{001Z}: -2 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{-2} \qquad \texttt{1110}: 2 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{2}$$

$$\texttt{01ZZ}: -1 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{-1} \qquad \texttt{1111}: 3 \rightarrow \text{regime} = \left(2^{2^{es}}\right)^{3}$$

[3] Remember: This is just an example, the regime field has variable length!

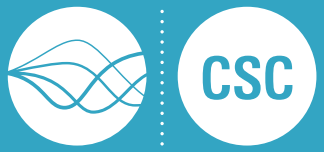
4-bit Posit (2-bit Exponent Size)



Examples

Posit (8-bit, 2-bit Exponent)

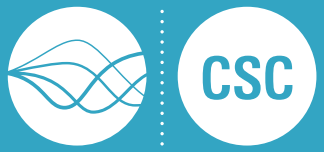
- $11000000 : -1^1 \cdot \left(2^{2^2}\right)^0 \cdot 2^0 \cdot (1+0) = -1$
- $01001101 : -1^0 \cdot \left(2^{2^2}\right)^0 \cdot 2^1 \cdot (1+\frac{5}{8}) = 3.25 \approx \pi$
- $01001011 : -1^0 \cdot \left(2^{2^2}\right)^0 \cdot 2^1 \cdot (1+\frac{3}{8}) = 2.75 \approx e^1$



Dynamic Range

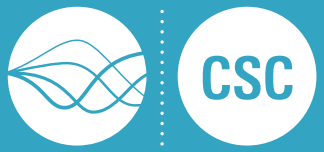
- Posits can be adapted in range.
- The (max) exponent size determines range (seed).
- Can be selected to be similar to floats.

Size (Bits)	Range (Float)	Range (Posit)	e_s (Posit)
16	$6 \cdot 10^{-8}$ ... $7 \cdot 10^{4}$	$4 \cdot 10^{-9}$ ... $3 \cdot 10^{8}$	1
32	$1 \cdot 10^{-45}$ ... $3 \cdot 10^{38}$	$6 \cdot 10^{-73}$ ... $2 \cdot 10^{72}$	3
64	$5 \cdot 10^{-324}$ ... $2 \cdot 10^{308}$	$2 \cdot 10^{-299}$ ... $4 \cdot 10^{298}$	4



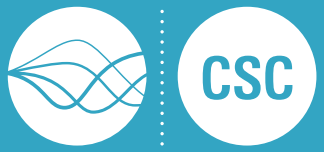
Quantitative Comparison

See Figures 8, 9, 10, 11, 12, 14, 16 in:
J.L. Gustafson, I. Yonemoto. **Beating Floating Point at its Own Game: Posit Arithmetic**. Supercomputing Frontiers and Innovations: An International Journal Archive, 4(2): 71–86, 2017.



Addition Closure

See Figure 13 in:
J.L. Gustafson, I. Yonemoto. **Beating Floating Point at its Own Game: Posit Arithmetic**. Supercomputing Frontiers and Innovations: An International Journal Archive, 4(2): 71–86, 2017.



Multiplication Closure

See Figure 15 in:
J.L. Gustafson, I. Yonemoto. **Beating Floating Point at its Own Game: Posit Arithmetic**. Supercomputing Frontiers and Innovations: An International Journal Archive, 4(2): 71–86, 2017.

LINPACK Benchmark

Linear System Solver Package:

- Performance of dense linear system solve.
- Measures Flop (Pop) throughput.
- Determines the "Top 500" of super computers.

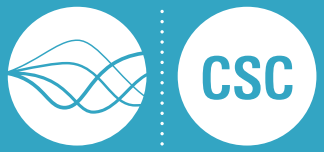
$$A \in U_{[0,1]}^{N \times N}$$

$$A \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = b$$

$$\rightarrow A \quad x \quad = b$$

32-bit Posits (3 regime bits) give a vector of exact ones![4]

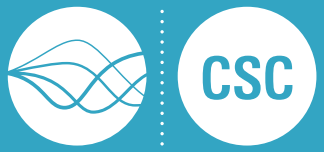
[4] (64-bit Floats don't)



Advantages

- No more over- and underflow.
- Higher accuracy.
- Unique bit patterns.
- Sane dynamic range.
- Negation symmetry.
- Sorted like integers.
- Possibly halves memory demand.
- Posit design suggests fused operations
  for chained multiplication and additions[5]
  in larger scratch registers, the so-called *quire*.
- Related formats are in development (Elias γ, δ, ω)

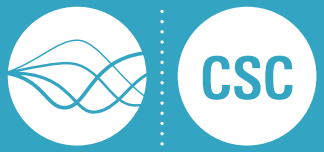
[5] See also FMA and MAD.



“Disadvantages”

- No reciprocal symmetry[6] (Compared to Unum II).
- No associativity.
- No hardware implementations (yet).
- Only signalling NaNs.
- No distinction between $-\infty$ and ∞.

[6] Except for powers of two, or different fraction maps.

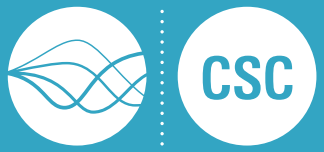


Implementation

Soft Posits:

- **Octave**: `https://github.com/diegofgcoelho/positsoctave`
- **Matlab**: `https://bitbucket.org/kvasnica/munum`
- **Python**: `https://github.com/xman/numpy-posit`
- **Julia**: `https://github.com/Etaphase/FastSigmoids.jl`
- **C++**: `https://github.com/stillwater-sc/universal`
- **C**: `https://github.com/libcg/bfp`
- and more[7] at: `https://posithub.org/khub_community`

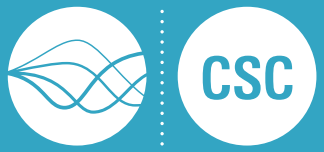
[7] FYI: I am dabbling in soft posits, too.



Summary

- less memory (use 32-bit posit instead of 64-bit float)
- more accurate (can be shown quantitatively and qualitatively)
- very elegant (projective mapping of reals)

More info: `https://posithub.org`



References

- D. Goldberg. **What Every Computer Scientist Should Know About Floating-Point Arithmetic**. Computing Surveys, 32(1): 6–48, 1991.
- J.L. Gustafson, I. Yonemoto. **Beating Floating Point at its Own Game: Posit Arithmetic**. Supercomputing Frontiers and Innovations: An International Journal Archive, 4(2): 71–86, 2017.
- J.L. Gustafson. **Posit Arithmetic**. Posithub, Knowledge Hub, 2017.
- J.L. Gustafson. **The End of Error: Unum Computing**. Chapman and Hall/CRC, 2015.
- Laslo Hunhold. **The Unum Number Format: Mathematical Foundations, Implementation and Comparison to IEEE 754 Floating-Point Numbers**. Bachelor Thesis, University of Cologne, 2016.
- P. Lindstrom, S. Lloyd, J. Hittinger. **Universal Coding of the Reals - Alternatives to IEEE Floating Point**. Proceedings of the Conference for Next Generation Arithmetic: 5, 2018.
- Wikipedia contributors. **Unum (number format)**. Wikipedia, The Free Encyclopedia, 2018.

Bonus Level: Recovering Reciprocal Symmetry via Fraction Maps

Float and Posit Fraction Map: $\phi(f) = 1 + f$ (linear)

Reciprocal Symmetry Fraction Map[8]: $\phi(f) = \begin{cases} 1+f & e > 0 \\ \frac{2}{2+f} & e < 0 \end{cases}$ (nonlinear)

Note:

- This means easy, but not necessarily faster, divisions.
- First bit (sign) is additive indicator ("left/right")
- Second bit (exponent sign) is multiplicative indicator ("upper/lower")
- One can construct other fraction maps for specific effects.

[8] P. Lindstrom, S. Lloyd, J. Hittinger. **Universal Coding of the Reals - Alternatives to IEEE Floating Point**. Proceedings of the Conference for Next Generation Arithmetic: 5, 2018.