UNUM SYSTEMS FOR BETTER COMPUTATIONAL ARITHMETIC

Prof. John L. Gustafson
A*STAR-CRC and National University of Singapore
john.gustafson@nus.edu.sg

October 25, 2016
1st Annual CROSS Research Symposium
Outline

• The need to fix floating-point
• Upward compatible: Type 1 unums
• The Wrath of Kahan
• Clean break: Type 2 unums
• Beating floats at their own game
• The need to fix floating-point
• Upward compatible: Type 1 unums
• The Wrath of Kahan
• Clean break: Type 2 unums
• Beating floats at their own game
Everyone accepts lousy arithmetic

Divide 1 by 3 on calculator:

Multiply by 3, get mistake:

“Computers don’t make mistakes”

Yes they do. And at incredibly high speeds.
Remember the Pentium® Divide Bug?

\[
\frac{1}{10} = 0.10000002384185791015625
\]

- Discovered 1994, after chip out 1 year
- One in 9 billion divides were wrong
- Intel initially denied need to fix it
- Intel eventually spent $0.475 billion fixing it

I fooled you. That’s not the Pentium Divide Bug. That’s standard floating-point arithmetic!
When rounding error killed 38 people

- First Gulf War
- Patriot missed Scud
- Struck Army barracks
- Left on for 100 hours
- Cumulative float error
- Guidance off by .43 sec
Quick Tutorial on Rounding Error

“0.1” as a standard float is 0.10000002384185791015625, rounded. Errors start from the moment you enter input data!

Patriot Missile example: accumulating seconds, 0.1 at a time, for 100 hours, will be off by 0.43 seconds

Also,
\[(a + b) + c \neq a + (b + c)\]
FAIL

Also,
\[a \times (b + c) \neq a \times b + a \times c\]
FAIL

So floating point math flunks algebra! This is a problem for parallel programmers. Are different answers a bug or a rounding error?
It’s not just about correctness

• Current floating-point math wastes energy, power, time, and storage, by using excess precision everywhere. Our calculations have become obese.
• Widespread issue, beyond just HPC; precision excess is prevalent in search, games, graphics, financial calculations, speech recognition…
• Floats are hard to use because programming bugs and rounding errors look alike! Inhibits use of parallel processing.
• Numerical analysts are becoming fewer and fewer as our computations become ever more ambitious.
Problem: power and heat

- Huge heat sinks
- 20 MW limit for exascale
- Google’s electric bill
- Mobile device battery life
- Heat intensity means bulk
- Bulk limits speed
## The “Memory Wall”

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy consumed</th>
<th>Time needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit multiply-add</td>
<td>200 pJ</td>
<td>1 nsec</td>
</tr>
<tr>
<td>Read 64 bits from cache</td>
<td>800 pJ</td>
<td>3 nsec</td>
</tr>
<tr>
<td>Move 64 bits across chip</td>
<td>2000 pJ</td>
<td>5 nsec</td>
</tr>
<tr>
<td>Execute an instruction</td>
<td>7500 pJ</td>
<td>1 nsec</td>
</tr>
<tr>
<td>Read 64 bits from DRAM</td>
<td><strong>12000 pJ</strong></td>
<td><strong>70 nsec</strong></td>
</tr>
</tbody>
</table>

Notice that 12000 pJ at 3 GHz = 36 watts!

One-size-fits-all overkill 64-bit precision wastes energy, storage, bandwidth.
Float Disaster: The Ariane 5

- 64-bit float measured speed
- 16-bit guidance system; *oops*
- $0.7$ billion gone in seconds

Why do *programmers* have to manage storage sizes when computers are much better at doing it?
Decades of asking “How do you know your answer is correct?”

- “(Laughter) “What do you mean?” (This is the most common response)
- “We used double precision.”
- “It’s the same answer we’ve always gotten.”
- “It’s the same answer others get.”
- “It agrees with special-case analytic answers.”
The Sleipner Oil Platform Disaster

- Error in structural analysis; collapsed to ocean floor
- ~$1 billion loss (in 2014 dollars)

August 23, 1991

August 24, 1991

Oops.
Terminology Reminders

- **Precision** = Digits available to store a number (“32-bit” or “4 decimal”, for example)
- **Accuracy** = Number of valid digits in a result (“to three significant digits”, for example)
- **ULP** = Unit of Least Precision.

Precise but not accurate: \( \pi = 3.1400000000001 \)
Accurate but not precise: \( 3.1 < \pi < 3.2 \)

Precision is not a goal.
Precision is the means, **accuracy** is the end.
Precision versus Accuracy

High precision.
Low accuracy.

Low precision.
High accuracy.
Happy 102\textsuperscript{th} Birthday, Floating Point

1914: Leonardo Torres y Quevedo proposed a relay-based computer with what we now call floating point: A fraction part and a scaling factor.
More Early Float History

- 1943: First built into actual machine by Zuse (using electromechanical relays)
- 1951: IBM introduces vacuum tube computer with floats, the IBM 701
- ~1985: IEEE committee creates a standard for floats, “IEEE 754”

We are overdue for something better.
COMPUTERS THEN
COMPUTERS NOW
ARITHMETIC THEN

SINGLE

1 8 23

DOUBLE

1 11 52

EXTENDED

1 15 "H" 64
ARITHMETIC NOW

Single

- Sign bit: 1
- Exponent: 8
- Mantissa: 23

Double

- Sign bit: 1
- Exponent: 11
- Mantissa: 52

Extended

- Sign bit: 1
- Exponent: 15
- Mantissa: "H"

1970: 30 sec per page

2016: 30 sec per page

We use faster technology for better prints, not to do low-quality prints in milliseconds.
What’s wrong with IEEE 754?

- It’s a *guideline*, not a *standard*
- No guarantee of identical results across systems
- No way to express most of the real number line
- Invisible rounding errors; “inexact” flag is useless
- Breaks laws of algebra
- Overflows to infinity, underflows to zero
- Exponents usually too large; not adjustable
- Flat precision across vast range, then falls off cliff
- Wasted bit patterns; “negative zero,” too many NaN values
- Subnormal numbers are headache
- Divides are hard
- Decimal floats are too expensive, no 32-bit
- Wobbling precision
- … and so on.
• The need to fix floating-point
• **Upward compatible: Type 1 unums**
• The Wrath of Kahan
• Clean break: Type 2 unums
• Beating floats at their own game
A New Number Format: The Unum

- Universal numbers
- Integers ➔ floats ➔ unums
- No rounding error
- No overflow to infinity
- No underflow to zero
- They obey algebraic laws!
- Fewer bits than floats
- But… they’re new
- Some people don’t like new

“You can’t boil the ocean.”

—Former Intel exec, when shown the unum idea
“The computer cannot give you the exact value, sorry. Use this value instead. It’s close.”
A Key Idea: The Ubit

We have *always* had a way of expressing reals correctly with a finite set of symbols.

Incorrect: \( \pi = 3.14 \)

Correct: \( \pi = 3.14\ldots \)

The latter means \( 3.14 < \pi < 3.15 \), a *true statement*.

Presence or absence of the “…” is the *ubit*, just like a sign bit. It is 0 if exact, 1 if there are more bits after the last fraction bit, not all 0s and not all 1s.
Floats only express discrete points on the real number line.

Use of a tiny-precision float highlights the problem.
The ubit can represent exact values or the range *between exacts*. Unums cover the entire extended real number line using a finite number of bits.
Three ways to express a big number

Avogadro’s number: \( \sim 6.022 \times 10^{23} \) atoms or molecules

**Integer (80 bits):**

```
0 11111111000010101010011110100101111101010000100101001100000000000000000000000
```

- **Sign**
- **Lots of digits**

**IEEE Standard Float (64 bits):**

```
0 10001001101111110000101010011101010100010101111110101000010011
```

- **Sign**
- **Exponent (scale)**
- **Fraction**

**Unum (29 bits):**

```
0 11001101 11111100001 111 1011
```

- **Sign**
- **Exp.**
- **Frac.**
- **Ubit**
- **Exp. size**
- **Frac. size**

*Self-descriptive “utag” bits track and manage uncertainty, exponent size, and fraction size*
Four reasons unums can sometimes use fewer bits

- Exponent smaller by about 5 – 10 bits on average
- Trailing zeroes in fraction compressed away, saves ~2 bits
- Values close to 1 are far more common in real programs
- Cancellation removes bits and the need to store them
Long-distance communication was precious in 1836, so…

- Morse Code: Use shortest bit strings for commonest data

- Compare that concise efficiency with ASCII:
“Variable bit size is too expensive”

- The utag serves as a linked-list pointer for packing
- “Chapter 7: Fixed-size unum storage” pp. 93–102
- Energy/power savings still possible with unpacked form

A typeface example of space savings from variable size:

Courier, 16 point

“Unums offer the same trade-off versus floats as variable-width versus fixed-width typefaces: Harder for the design engineer and more logic for the computer, but superior for everyone else in terms of usability, compactness, and overall cost.” (page 193)

Times, 16 point

“Unums offer the same trade-off versus floats as variable-width versus fixed-width typefaces: Harder for the design engineer and more logic for the computer, but superior for everyone else in terms of usability, compactness, and overall cost.” (page 193)
The Warlpiri unums: 1, 2, many…
• The need to fix floating-point
• Upward compatible: Type 1 unums
• The Wrath of Kahan
• Clean break: Type 2 unums
• Beating floats at their own game
The Wrath of Kahan

- Berkeley professor William Kahan is the father of modern IEEE Standard floats
- Also the authority on their many dangers
- Every idea to fix floats faces his tests that expose how new idea is even worse


Can unums survive the wrath of Kahan?
Example: Kahan’s “Smooth Surprise”

Find minimum of \( \log(|3(1-x)+1|)/80 + x^2 + 1 \) in \( 0.8 \leq x \leq 2.0 \)

Plot, test using \textit{half a million} double-precision IEEE floats. Shows minimum at \( x = 0.8 \).

\textbf{FAIL}

Even if you test \textit{every float in the range}, it will fail to detect the minimum!
Why did Kahan want to debate me?

- *The End of Error* had dozens of reviewers, including David Bailey, Horst Simon, Gordon Bell, John Gunnels…
- Kahan has had the manuscript since November 2013 but ceased email conversation about its content in July 2014
- Then this happened (Amazon.com):
Kahan’s biggest blind spot of all

Remember: There is nothing floats can do that unums cannot. ■

The last line of my book, p. 413, and emphasized throughout

- Unums are a *superset* of IEEE floats. Not an “alternative.”
- We need not throw away float algorithms that work well.
- Rounding can be *requested*, not forced on users. **Unums end the error of mandatory, invisible substitution of incorrect exact values for correct answers.**
- Float methods are a good way to deal with “The Curse of High Dimensions” in many cases, like getting a starting answer for $Ax = b$ linear systems in polynomial time.
Another classic Kahan example

Find the area of a triangle with sides $a$, $b$, $c$ where $a$ and $b$ are only 3 ULPs longer than half the length of $c$.

Try the formula $\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{a+b+c}{2}$

IEEE Quad Precision (128 bits, 34 decimals): Let $a = b = 7/2 + 3 \cdot 2^{-111}$, $c = 7$.

If $c$ is 7 light years long, 3 ULPs is ~1/200 the diameter of a proton. The correct area is about 55 times the surface area of the earth. To 34 decimals:

$3.14784204874900425235885265495507 \ldots \times 10^{-16}$ square light years.
Quad-precision float result

- IEEE Quad float gets 1 digit right:
  \[ 3.634814908423321347259205161580577\ldots \times 10^{-16}. \]
- Error is about 15 percent, or 252 peta-ULPs.
- **Result does not admit any error, nor bound it.**
- Kahan’s approach: Sort the sides so \( a \geq b \geq c \) and rewrite the formula as

\[
Area = \frac{\sqrt{(a + (b + c))(c - (a - b))(c + (a - b))(a + (b - c))}}{4}
\]

This is within 11 ULPs of the correct area, but it takes **hours** to figure out such an approach.

It also uses twice as many operations, but that’s not the issue: it’s the **people cost** of the approach.
Unum approach to the thin triangle

- Use no more than 128 bits per number, but *adjustable*
- Exponent can be 1 to 16 bits (wider range than quad)
- Fraction can be 1 to 128 bits, plus the hidden bit (higher precision than quad)
- Result is a *rigorous bound* accurate to 31 decimals:

\[ 3.14784204890042523588526549455070\ldots \times 10^{-16} < \text{Area} < 3.14784204890042523588526549455139\ldots \times 10^{-16} \]

The size of that bound is the area of a square 8 nanometers on a side.

*No need to rewrite the formula.*
# Summary of comparison

<table>
<thead>
<tr>
<th>Format Capabilities</th>
<th>Quad-precision IEEE floats</th>
<th>Unums, {4,7} environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Range</td>
<td>~6.5×10⁻⁴⁹⁶⁶ to 1.2×10⁴⁹³²</td>
<td>~8.2×10⁻⁹⁹⁰³ to ~2.8×10⁹⁸⁶⁴</td>
</tr>
<tr>
<td>Precision</td>
<td>~34.0 decimal digits</td>
<td>~38.8 decimal digits</td>
</tr>
</tbody>
</table>

**Results on thin triangle**

<table>
<thead>
<tr>
<th></th>
<th>Quad-precision IEEE floats</th>
<th>Unums, {4,7} environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum bits used</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Average bits used</td>
<td>128</td>
<td>90</td>
</tr>
<tr>
<td>Result</td>
<td>Area = 3.6481490842332134725920516 × 10⁻¹⁶ &lt; Area &lt; 3.147842048749004252358852654945507×10⁻¹⁶</td>
<td>3.147842048749004252358852654945514×10⁻¹⁶</td>
</tr>
<tr>
<td>Type of information loss</td>
<td>Invisible error, very hard to debug</td>
<td>Rigorous bound, easy to debug if needed</td>
</tr>
<tr>
<td>Error / bound size</td>
<td>~4×10¹⁵ meters²</td>
<td>~6×10⁻¹⁷ meters²</td>
</tr>
</tbody>
</table>
"The Lord of the Reals... Does NOT Share Power."
• The need to fix floating-point
• Upward compatible: Type 1 unums
• The Wrath of Kahan
• **Clean break: Type 2 unums**
• Beating floats at their own game
Unums Type 2: Break with IEEE, to get:

- All operations equally fast, even $x^y$
- No penalty for decimal instead of binary
- Easy to build with 2016-era chip technology
- Computation with mathematical rigor
- Robust set representations with a fixed number of bits
- A cleaner way to deal with exception cases

**BUT:**
- Harder to scale to high precision
- Higher average number of bits than Type 1

Strategy: Get ultra-low precision right, then work up.
All projective reals, using 2 bits

“±∞” is “the point at infinity” and is unsigned.

Think of it as the reciprocal of zero.
Absence-Presence Bits

0 (open shape) if absent from the set, 1 (filled shape) if present in the set.

Rectangle if exact, oval or circle if inexact (range)

Red if negative, blue if positive

$2^4 = 16$ possible subsets of the extended reals.
Sets become \textit{numeric quantities}

{
\begin{itemize}
\item The empty set, \{ \}
\item All positive reals \((0, \infty)\)
\item Zero, 0
\item All nonnegative reals, \([0, \infty)\)
\item All negative reals, \((-\infty, 0)\)
\item All nonzero reals, \((-\infty, 0) \cup (0, \infty)\)
\item All nonpositive reals, \((-\infty, 0]\)
\item All reals, \((-\infty, \infty)\)
\item The point at infinity, \(\pm \infty\)
\item The extended positive reals, \((0, \infty]\)
\item The unsigned values, \(0 \cup \pm \infty\)
\item The extended nonnegative reals, \([0, \infty]\)
\item The extended negative reals, \([-\infty, 0)\)
\item All nonzero extended reals \([-\infty, 0) \cup (0, \infty]\)
\item The extended nonpositive reals, \([-\infty, 0]\)
\item All extended reals, \([-\infty, \infty]\)
\end{itemize}

\textit{“SORNs”: Sets Of Real Numbers}

Closed under \(x + y\), \(x - y\), \(x \times y\), \(x \div y\) \textit{and...} \(x^y\)

Tolerates division by 0.

\textit{No indeterminate forms.}

Very different from \textit{symbolic ways of dealing with sets.}
No more “Not a Number”

\[\sqrt{-1} = \text{empty set}: \quad \square \quad \square \quad \square \quad \square \]

\[0 / 0 = \text{everything}: \quad \square \quad \square \quad \square \quad \square \]

\[\infty - \infty = \text{everything}: \quad \square \quad \square \quad \square \quad \square \]

\[1^\infty = \text{all nonnegatives}, [0, \infty]: \quad \square \quad \square \quad \square \quad \square \]

etc.

Answers, as limit forms, are sets. We can express those!
Now include +1 and –1

The SORN is 8 bits long.

This is actually enough of a number system to be useful!
Example: Robotic Arm Kinematics

Example of Real Constraints: inverse kinematics of an elbow manipulator

\[\begin{align*}
s_2c_5s_6 - s_3c_5s_6 - s_4c_5s_6 + c_2c_6 + c_3c_6 + c_4c_6 &= 0.4077; \\
c_1c_2s_5 + c_1c_3s_5 + c_1c_4s_5 + s_1c_5 &= 1.9115; \\
s_2s_5 + s_3s_5 + s_4s_5 &= 1.9791; \\
c_1c_2 + c_1c_3 + c_1c_4 + c_1c_2 + c_1c_3 + c_1c_2 &= 4.0616; \\
s_1c_2 + s_1c_3 + s_1c_4 + s_1c_2 + s_1c_3 + s_1c_2 &= 1.7172; \\
s_2 + s_3 + s_4 + s_2 + s_3 + s_2 &= 3.9701; \\
s_i^2 + c_i^2 &= 1 \quad (1 \leq i \leq 6)
\end{align*}\]

Notice all values must be in \([-1,1]\) →

12-dimensional nonlinear system (!)
“Try everything”… in 12 dimensions

Every variable is in $[-1, 1]$, so split into $[-1, 0)$ and $[0, 1]$ and compute the constraint function to 3-bit accuracy.

- ■ = violates constraints
- ■ = compliant subset

$2^{12} = 4096$ sub-cubes can be evaluated in parallel, in a few nanoseconds.

Note: The test uses slightly more than 3 bits for the function evaluation
There is nothing special about 2. We could have added 10 and 1/10, or even $\pi$ and $1/\pi$, or any exact number.

(Yes, $\pi$ can be numerically exact, if we want it to be!)
Note: sign bit is in the usual place

The sign of 0 and ±∞ is meaningless, since

0 = –0 and ±∞ = –±∞.
Negation is trivial

To negate, flip horizontally.

Reminder: In 2’s complement, flip all bits and add $1$, to negate. *Works without exception*, even for $0$ and $\pm\infty$. (They do not change.)
A new notation: Unary “/”

Just as “−” can be put before \( x \) to mean \( 0 − x \), unary “/” can be put before \( x \) to mean \( 1/x \).

Pronounce it “over \( x \).” Teach in elementary school?

\[
− (−x) = x, \text{ just as } / (/x) = x.
\]
\[
x − y = x + (−y), \text{ just as } x ÷ y = x × (/y)
\]

Unums Type 2 are lossless under negation and reciprocation.

Operations \(+ − × ÷\) are on equal footing.
Reciprocation is trivial, too!

To reciprocate, flip vertically.

Reverse all bits but the first one and add 1, to reciprocate. Works without exception. +1 and –1 do not change.
The last bit serves as the *ubit*

ubit = 0 means exact.

ubit = 1 means the open interval between exact numbers.

“uncertainty bit”.

Example: This means the open interval (½, 1). Or (get used to it), (/2, 1).
Divide by 0 mid-calculation and still get the right answer

What is \( \frac{1}{1/x + \frac{1}{2}} \) for \(-1 < x \leq 2\)?

10-unum SORN for \( x = (-1, 2] \)

Lossless SORN for \( 1/x = [\frac{1}{2}, -1) \)

Divide by 0 is an ordinary operation!
Add $\frac{1}{2}$, reciprocate again

Add $\frac{1}{2}$

lossless SORN for $\frac{1}{x} + \frac{1}{2} = [1, -\frac{1}{2})$

Reciprocate

lossless SORN for $\frac{1}{\left(\frac{1}{x}+\frac{1}{2}\right)} = (-2, 1]$
Back to kinematics, with exact $2^k$

Split one dimension at a time. Needs only 1600 function evaluations (microseconds).

Display six 2D graphs of $c$ versus $s$ (cosine versus sine… should converge to an arc)

Here is what the *rigorous bound* looks like after one pass.

Information = uncertainty.

Uncertainty = answer volume.

Information increases by $1661 \times$
Make a second pass

Still using ultra-low precision

Starting to look like arcs (angle ranges)

457306 function evaluations ($\mu$secs, using parallelism)

Information increases by a factor of $3.7 \times 10^6$
A third pass allows robot decision

Transparency helps show 12 dimensions, 2 at a time.

Starting to look like arcs (angle ranges).

6 million function evaluations (a few msec)

Information increases by a factor of $1.8 \times 10^{11}$

Remember, this is a **rigorous bound** of all possible solutions. Gradient-type searching with floats can only **guess**.
Move up to 8-bit unums

What is the best use of one byte for real-valued calculations?

Start with kindergarten numbers:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Unite with that set divided by 10 to center the set about 1:
0.1, 0.2, 0.3, …, 0.9,
1, 2, 3, …, 9, 10

This has the classic problem with decimal IEEE floats: “wobbling precision.” Actually, it should be called wobbling accuracy.
Reciprocal closure cures wobbling accuracy!

Unite set with the reciprocals of the values, guaranteeing closure:

0.1, /9, 0.125, /7, /6, 0.2, 0.25, 0.3, /3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, /0.9, 1.25, /0.7, /0.6, 2, 2.5, 3, /0.3, 4, 5, 6, 7, 8, 9, 10

That's 30 numbers. Budget for exact unums is $2^6 = 64$, so room for 34 more exact values.

No “kinks”!
Flat precision makes table generation and fused operations easier.

Imagine: custom number systems for *application-specific arithmetic*

- Look-up tables simplify if precision is flat
- Power of 10 determined via integer divide, instead of having a separate bit field
8-bit unum means 256-bit SORN

Ultra-fast parallel arithmetic on arbitrary subsets of the real number line.

Ops can still finish within a single clock cycle, with a tractable number of parallel OR gates.
Typically, you only need 16 bits to express the SORN.

Connected sets remain connected under $+ - \times \div$, even division by zero!

Run-length encoding of a contiguous block of 1s amongst 256 bits only takes 16 bits.

- **00000000 00000000** means all 256 bits are 0s
- **xxxxxxxx 00000000** means all 256 bits are 1s (if any x is nonzero)
- **00000010 00000110** means there is a block of 2 1s starting at position 6

Trivial logic still serves to negate and reciprocate compressed form of value.
Table look-up background

In 1959, IBM introduced its 1620 Model 1 internal name “CADET.”

All math was by table look-up.

Customers decided CADET stood for “Can’t Add, Doesn’t Even Try.”
Table look-up can use ROM

- Read-Only Memory needs very few transistors. ~3x denser than DRAM, ~14x denser than SRAM.
- Billions of ROM bits per chip is easy.
- Imagine the speed… all operations take 1 clock! Even $x^y$.
- 1-op-per clock architectures are much easier to build.
- Single argument-operations require tiny tables. Trig, exp, you name it.

Low-precision rigorous math is possible at 100x the speed of sloppy IEEE floats.
What about, *decent* precision?
Is 3 decimals enough for a *result*?

IEEE half-precision has ~3 decimal accuracy.
9 orders of magnitude, $6 \times 10^{-5}$ to $6 \times 10^4$.
Many bit patterns wasted on NaN, negative zero, etc.
Can 16-bit unums do better, and *express decimals exactly*?

$2^{16} = 65536$ bit patterns.
8192 in the “u-lattice”.
Start with \{1.00, 1.01, ..., 9.99\}.
Unite with reciprocals.
While set size < 16384:
unite with $10 \times$ set.
Clip to 16384 values centered at 1.
Unite with negatives.
Unite with open intervals.
What is the *dynamic range*?
Answer: $9^{+}$ orders of magnitude

$0.389 \times 10^{-5}$ to $0.389 \times 10^{5}$

This is 1.5 times larger than the range for IEEE half-precision floats.

This is the Mathematica code for generating the number system.

```
nbits = 16;
digits = 3; set = Range[10^{digits-1}, 10^{digits-1}] / 10^{digits-1};
set = Union[set, 10 / set];
set = Union[set, set / 10];
While[Length[set] < 2^{nbits-2},
  set = Union[set, set / 10, set * 10]];
Off[General::infy]
m = [Length[set] / 2];
set = Union[{0, 1 / 0},
  Take[set, {m - 2^{nbits-3} + 1, m + 2^{nbits-3} - 1}]];
set = Union[set, -set];
Length[set]
32768
```

Notice:
No subnormals.
No NaNs.
No “negative zero.”
Software-defined number formats

- Extend FPU with unums type 1
  - 64-bit unpacked unum (10 decimals)
  - Optional IEEE, but *deterministic* rules
  - Fused operation set for both
- Unum type 2 for ultrafast low precision
  - Standard format in ROM
  - User-defined table option
  - Compiler can generate tables

**Future CPU**

<table>
<thead>
<tr>
<th>ALU</th>
<th>Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPU</td>
<td>IEEE floats</td>
</tr>
<tr>
<td>Unums 1 extensions</td>
<td></td>
</tr>
<tr>
<td>ULU</td>
<td>Unums 2</td>
</tr>
<tr>
<td>User-defined tables</td>
<td></td>
</tr>
</tbody>
</table>

Histogram real values used by application; select $u$-lattice to maximize information per bit (much like Shannon maximum entropy concept)
For more information…

• Aimed at *general* reader; Kahan hates its casual style
• Complete prototype environment is available as free *Mathematica* notebook through publisher (MIT open source license)
Work in Progress—Python

- [https://github.com/jrmuizel/pyunum](https://github.com/jrmuizel/pyunum)
- Complete! But not fast.
- Line-by-line translation of Mathematica prototype
- Needs to avoid using $g$-layer for most arithmetic
- Main author: Jeff Muizelaar [jrmuizel@gmail.com](mailto:jrmuizel@gmail.com)
- Brings up the need for a test suite for distribution
- Anthony di Franco is working on improving the port, [di.franco@gmail.com](mailto:di.franco@gmail.com)
Work in Progress—Julia

Put unum data types into Julia (funded by A*STAR)
Alan Edelman, MIT
Viral Shah, Julia Computing
Deepak Vinchhi, Julia Computing
Isaac Yonemoto, REX Computing
Three programmers in India, TBD
Scott Jones, Gandalf Software, Inc.
Jeffrey Sarnoff, Diadem Special Projects
Job Van Der Zwan, julia-users@googlegroups.com
Thomas Breloff, Cointegrated Technologies
Daniel O’Malley, Los Alamos National Laboratory
Waldir Pimenta, Universidade do Minho
Zenna Tavares, MIT
Steven G. Johnson, MIT
Stefan Karpinski, Julia Computing
Work in Progress—misc.

- Lawrence Livermore National Lab
  - Team of 11 working on applications
- University of California Santa Cruz
  - Carlos Maltzahn established open source community for unums
- Karlsruhe Institute of Technology
  - Ulrich Kulisch has published a formal unum definition
- IEEE
  - Current President Tom Conte has proposed creating a unum IEEE Standard
  - I think it’s too early, but I appreciate the support!
- NTU project to do Neural Networks with Type 2, FPGA
Future Unums 2 Directions

- Create 32-bit and 64-bit unums with new approach; table look-up still practical?
- Compare with IEEE single and double
- General SORNs need run-length encoding.
- Build C, D, Julia, Python versions of the arithmetic (Julia version is mostly done)
- Test on various workloads, like
  - Deep learning
  - N-body
  - Ray tracing
  - FFTs
  - Linear algebra done right (complete answer, not sample answer)
  - Other large dynamics problems
• The need to fix floating-point
• Upward compatible: Type 1 unums
• The Wrath of Kahan
• Clean break: Type 2 unums
• Beating floats at their own game
Beating Floats at their Own Game

Suppose we ignore all the ubit math and other “rigorous bound” aspect of unums. We can still make massive improvements to methods that round.

Binary and decimal floats cut a crooked path through the real number line:

**accplot[binarylattice]**
Minimum accuracy = 1.29113 decimals.
Loss of decimals from ideal = 0.214017

**accplot[decimallattice]**
Minimum accuracy = 0.52139 decimals.
Loss of decimals from ideal = 0.733882
Beating Floats at their Own Game

Eliminating accuracy “wobble” gives better accuracy and dynamic range.

Minimum accuracy = 2.36438 decimals.
Loss of decimals from ideal = 0.0489216

Plus, decimal I/O, perfect reciprocals, fast division...
Closure plot for multiply, divide

- = Exact result
- = Inexact
  (single ULP range)

Embedded - are where the power of 2 and the power of 5 differ by more than 4.
### Accuracy on a 32-bit budget

Compute: \[
\left( \frac{27/10 - e}{\pi - (\sqrt{2} + \sqrt{3})} \right)^{67/16} = 302.8827196 \ldots
\]

with \( \leq 32 \) bits per number.

<table>
<thead>
<tr>
<th>Number Type</th>
<th>Dynamic Range</th>
<th>Answer</th>
<th>Error/Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE 32-bit float</td>
<td>( 2 \times 10^{83} )</td>
<td>302.912\ldots</td>
<td>0.0297</td>
</tr>
<tr>
<td>Rational arithmetic</td>
<td>( 2 \times 10^{9} )</td>
<td>302.955\ldots</td>
<td>0.0725</td>
</tr>
<tr>
<td>Interval arithmetic</td>
<td>( 10^{12} )</td>
<td>[18.21875, 33056.]</td>
<td>3.3\times10^{4}</td>
</tr>
<tr>
<td>Type 1 unums, 32-bit max. size</td>
<td>( 4 \times 10^{83} )</td>
<td>(288.320)</td>
<td>32</td>
</tr>
<tr>
<td>Type 1 unums 32-bit avg. size</td>
<td>( 4 \times 10^{83} )</td>
<td>(302.75, 303.)</td>
<td>0.25</td>
</tr>
<tr>
<td>Type 2 unums</td>
<td>( 10^{99} )</td>
<td>(302.887\ldots)</td>
<td>0.0038</td>
</tr>
</tbody>
</table>
Summary

- Fewer bits, better answers possible with unums
- Unums save energy, power, storage, bandwidth, programming effort
- Applies to everything from portable devices to exascale supercomputers
- Unums could show up in real systems in just a couple of years
- They might completely obsolete the present way we do calculations

This idea is not patent protected.

MIT Open Source license.