ETAPHASE

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| **1. REPORT DATE** *(DD-MM-YYYY)*<br>02/21/2019 | **2. REPORT TYPE**<br>Final Technical Report | **3. DATES COVERED** *(From - To)*<br>1/1/2017 - 12/31/2019 |

| | |
|---|---|
| **4. TITLE AND SUBTITLE**<br>TRADES uDESIGN Seedling Final Report: unums, posits, and quires<br>Accomplishments to Date and Suggested Next Steps | **5a. CONTRACT NUMBER**<br>HR00111790007 |
| | **5b. GRANT NUMBER** |
| | **5c. PROGRAM ELEMENT NUMBER** |
| **6. AUTHOR(S)**<br>Dr. John L. Gustafson, Dr. Ruth Ann Mullen, Dr. Theodore Omtzigt | **5d. PROJECT NUMBER** |
| | **5e. TASK NUMBER** |
| | **5f. WORK UNIT NUMBER** |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**<br>Etaphase, Incorporated<br>8201 164th Avenue NE<br>Suite 200<br>Redmond, WA 98052-7615 | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**<br>National Aeronautics and Space Agency, Langley Research Center<br>1 NASA Dr., Hampton, VA 23666<br><br>Defense Advanced Research Projects Agency<br>Defense Sciences Office, Arlington, VA | **10. SPONSOR/MONITOR'S ACRONYM(S)**<br>NASA, DARPA, DSO<br><br>**11. SPONSOR/MONITOR'S REPORT NUMBER(S)** |

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution unlimited.

**13. SUPPLEMENTARY NOTES**
DARPA-funded research: BAA 16-39, DSO TRAnsformative DESign (TRADES)

**14. ABSTRACT**
Etaphase's uDESIGN Seedling was tasked to prove the accuracy in linear algebra of a new, highly efficient number representation for computers known as unum arithmetic. Unums enable computers to operate with approximately half as many bits without loss of accuracy. Unums eliminate floating point rounding errors, enabling computers to provide provably correct, exact or rigorously-bounded answers. DARPA-funded accuracy tests were performed on a hardware-friendly version of unums invented just prior to the formal January 2017 kick-off of the TRADES Program. These hardware-friendly unums, known as "Type III unums," comprise "posits," "valids," and "quires."

**15. SUBJECT TERMS**
12. Mathematical and Computer Sciences: precision, accuracy, computer, software, computer number systems, LINPACK, geometric representations, FFT, Hilbert matrix, artificial intelligence, deep learning, tensor mathematics, hardware, FPGA, ASIC, GPU, interval arithmetic, floating point.

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
|---|---|---|---|---|---|
| **a. REPORT** | **b. ABSTRACT** | **c. THIS PAGE** | | | Ruth Ann Mullen, PhD |
| Unclassified | Unclassified | Unclas-sified | UU Unclassified Unlimited | 55 | **19b. TELEPHONE NUMBER** *(Include area code)*<br>(206) 414 - 9283 |

21 February 2019

# "TRADES uDESIGN Seedling Final Report: unums, posits, and quires"

Final R&D Status Report

*Reporting Period:* 1 January 2017 – 31 December 2018

*Sponsored by*
Defense Advanced Research Projects Agency (DOD)
Defense Sciences Office (DSO)

*Issued by*
DARPA
Contract No. HR00111790007

*Agent*
Defense Advanced Research Projects Agency (DOD)
Defense Sciences Office (DSO)

| | |
|---|---|
| **Contract Number:** | HR00111790007 |
| **Proposal Number:** | DARPA-16-39-TRADES-FP-005 |
| **Contractor Name and PI:** | Etaphase, Incorporated, PI Ruth Ann Mullen, PhD |
| **Contractor Address:** | 8201 164th Ave NE, Redmond, WA 98052 |
| **Contract Period of Performance:** | 1 January 2017 – 31 December 2018 |
| **Total Contract Amount:** | **$305,739** |
| **Total Amount Expended:** | **$305,739** |

# TRADES uDESIGN Seedling
# Final Report:
# unums, posits, and quires

∞
10
–1
11
01
1
00
0
Better Answers.
Fewer Bits.

*Accomplishments to-date and Suggested Next Steps*

John Gustafson[1,2]

Ruth Ann Mullen[1]

Theo Omtzigt[3]

DARPA Program Manager: Dr. Jan Vandenbrande

January 25, 2019

[1] Etaphase
[2] National University of Singapore and A*STAR
[3] Stillwater Computing

## uDESIGN Seedling Final Report Table of Contents

**Commented [JG1]:** I don't understand why "quire size" shows up on the line for 3.4.1, but I don't know how to do auto-generation of the ToC. Anyway, make sure it gets fixed for the final version.

## 1. EXECUTIVE SUMMARY

Etaphase's uDESIGN Seedling was tasked to prove the accuracy in linear algebra of a new, highly efficient number representation for computers known as unum arithmetic[1]. Unums enable computers to operate with approximately half as many bits without loss of accuracy. Unums eliminate floating point rounding errors, enabling computers to provide provably correct, exact or rigorously-bounded answers. DARPA-funded accuracy tests were performed on a hardware-friendly version of unums[2] invented just prior to the formal January 2017 kick-off of the TRADES Program, known as "Type III unums," comprising "posits" "valids," and "quires. " [3][4][5]

Unums round only when explicitly commanded to do so; posits round automatically and thus provide a drop-in replacement for floats[2][3]. Valids are their interval arithmetic form, pairs of values that rigorously bound answers. Quires are used in both posits and valids, leveraging an accumulator to provide an exact dot product. Together, the posit and quire number representation for computers (collectively referred to as "posits") enables computers to operate with fewer bits without loss of accuracy, and to provide provably correct answers, as needed, without having to worry about floating point errors. Any form of interval arithmetic, including valids, requires revised algorithms to prevent intervals from growing so pessimistic (too-loose bounds) that the answers become useless; in contrast, quires can be used to *automatically* protect against the accumulation of rounding errors and do not require users to change their algorithms or even understand basic numerical analysis.

Halving the number of bits advantageously reduces memory hardware requirements, while speeding up memory-bound calculations and reducing energy costs associated with memory access. And since this new number representation can be used to provide provably correct answers, it will substantially expedite both the automation of coding and the performance optimization of software systems. The productivity of software engineers will be increased, as they will no longer continually need to worry if their computations are tainted by floating point errors. Calculations will run faster, with fewer capital resources, and with lower energy requirements.

The hardware-friendly nature of posits and quires means that less circuitry is required in a posit processing unit (PPU) than in an IEEE float processing unit (FPU), even if comparing posits and floats with the same number of bits. If reduced-size posits are used, the circuitry savings they provide is considerably more dramatic.

The lower power requirement and smaller silicon footprint for posits means that posit operations per second (POPS) supported by a chip can be significantly higher than floating point operations per second (FLOPS) using similar hardware resources. Next-generation computers designed for posits will perform more calculations per watt and per dollar with posits, while delivering superior answer quality. The energy sensitivities of battery-powered computers, GPU accelerators, and Deep Learning processors make the application of posits to these classes of computers particularly compelling.

Key features of this new number representation strongly recommend their adoption throughout the DoD and commercial computer systems.

Specifically, posits

- provide superior accuracy, dynamic range, and closure
- provide bitwise-reproducible answers, enabling computer programs operating on real numbers to produce, at long last, the same answers on different systems
- provide better answers with same number of bits (or equally good answers with fewer bits)
- use a simpler, more elegant design to reduce silicon cost, energy, and latency
- have much simpler exception handling
- require neither interval arithmetic nor variable size operands
- never overflow to infinity and never underflow to zero; they follow D. Knuth's guidelines
- map monotonically to computer integers (comparison operations are the same as for integers)
- use tapered accuracy for more information-per-bit (higher Shannon entropy)
- have only one rounding mode (round to nearest, the usual)
- allow faster addition because 2's complement format eliminates conditional branches
- simplify multiplication because there are no "subnormals"
- have perfect reciprocals for all integer powers of 2
- allow the sigmoid activation function computation in a single clock cycle, for Deep Learning

In performing the research on unum arithmetic for TRADES, we have made the following breakthroughs relevant to large-scale multi-physics simulations and HPC in general (including AI):

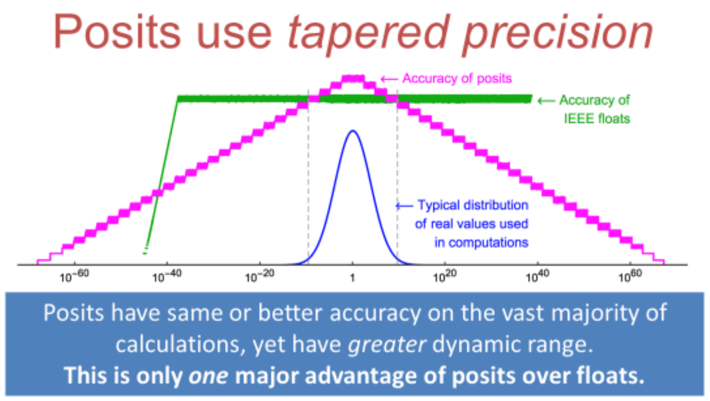
- A practical way to produce exact answers to linear algebra problems using fewer bits than consumed by error-prone floating-point numbers.
- An approach to the general problem of multiplication and addition operations on multidimensional arrays (tensors) that produces perfectly reproducible, correctly-rounded results even when the approach rearranges the computation to better suit the hardware (for parallel execution and to fit a cache hierarchy, say). The quire (exact dot product) of posit arithmetic makes array transformations immune to the rounding errors that plague floating-point numbers (because of the way that floating-point numbers suffer from failing to follow the associative or distributive laws of algebra.
- Constructive reverse error analysis for the first time; numerical analysis teaches that rounding produces an answer that is the exact answer to a *different problem*, and attempts to prove that the different problem is close to the one that was posed. The quire allows the different problem to be constructed and presented to the user along with the computed answer. The user can then be the judge of whether the different problem is close enough for the application at hand.

The combination of improved-quality number representation and the mathematics of array transformation make it possible to attack larger multi-physics problems through automatic numerical error control and automatic optimization of data placement for maximum speed, thereby assisting with the main goal of the TRADES program. The constructive reverse error analysis eliminates the problem that instability is invisible in rounded calculations, preventing a user from blindly accepting an answer to an ill-posed problem.

## 2. Overview: What are posits?

Posits were invented based on the way unum arithmetic was being used by early adopters: as a drop-in replacement for floating-point arithmetic (floats), where the explicit round-to-nearest-exact operation of unums was being applied frequently to mimic the rounding errors of floats. Unums were never designed to be used this way, but Gustafson responded, just before the TRADES period of funding began, by inventing a completely new and mathematically consistent way to represent points on the real number line in a fixed number of bits.

A key part of the design is based on the observation that floats have low Shannon entropy, if the sequence of floats in a program is regarded as a signal. As the figure below shows, the typical distribution of real value magnitudes in real programs is close to 1, with decreasing frequency for very large and very small magnitudes.


Posits use *tapered precision*

Posits have same or better accuracy on the vast majority of calculations, yet have *greater* dynamic range.
**This is only *one* major advantage of posits over floats.**

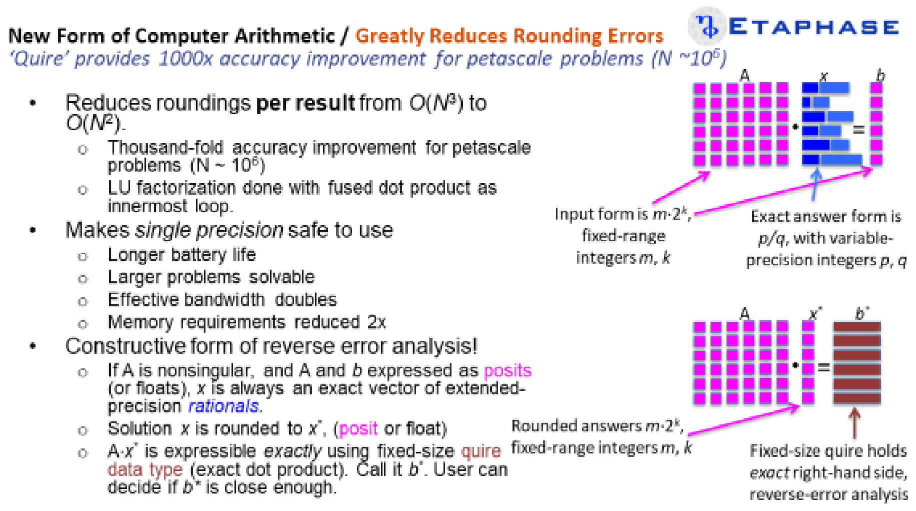
Float format is designed to have flat accuracy over a vast range, with tapered precision only for the very smallest numbers (gradual underflow, or “subnormal floats”). The reason for this distribution is both theoretical, practical, and sociological: If a calculation starts with a flat distribution of magnitudes, after many millions of calculations with those numbers, the Central Limit Theorem shows they will form a Gaussian distribution like that shown with the blue curve above. Algorithms are designed to avoid overflow and underflow, so existing software already is designed to keep exponents near the center of the range. And finally, humans tend to re-scale their problems (by selecting appropriate units, for instance) to make numbers easier to read, write, and understand.

Posits exploit this distribution by maximizing accuracy (higher density of represented values) near the center of the range, using a format that creates a tent-like accuracy as a function of the magnitude. This allows them to increase both accuracy *and* dynamic range over floats.

The design of posits also makes practical for the first time the idea of providing an *exact dot product*, using a fixed-point register called a “quire.” To compute dot products exactly using a fixed-point exact accumulator means the accumulator must have enough bits to store both the smallest value squared and the largest value squared, plus a few bits to guard against overflow. The Karlsruhe school of accurate computing applied the exact dot product to 64-bit floats, but this requires a very large (67-long) vector of 64-bit integers to serve as the accumulator, a total of 4,288 bits and not an integer power of 2 in size[6]. As a result, the powerful techniques of "Karlsruhe Accurate Arithmetic" were rejected by the IEEE 754 committee as well as the more recent IEEE 1788 committee deciding standards for *interval arithmetic*.

**Commented [JG2]:** I found a reference that uses this larger number, and that should probably be added as a reference here: https://hal.archives-ouvertes.fr/hal-01488916v2/document

The quire, for 32-bit posits, is only 512 bits. This is a very hardware-friendly size because it is the width of a cache line and also the width of an SSE instruction in modern x86 processors currently using 512-bit vector instruction sets such as Intel's Advanced Vector Extensions 512 (AVX-512) [7]. Hence, current chips are already designed to quickly move and manipulate quire-sized quantities. The figure below shows some breakthrough capabilities made possible with the quire, and discovered only as the result of the TRADES program.



## New Form of Computer Arithmetic / Greatly Reduces Rounding Errors

*'Quire' provides 1000x accuracy improvement for petascale problems (N ~$10^6$)*

- Reduces roundings **per result** from $O(N^3)$ to $O(N^2)$.
  - Thousand-fold accuracy improvement for petascale problems (N ~ $10^6$)
  - LU factorization done with fused dot product as innermost loop.
- Makes *single precision* safe to use
  - Longer battery life
  - Larger problems solvable
  - Effective bandwidth doubles
  - Memory requirements reduced 2x
- Constructive form of reverse error analysis!
  - If A is nonsingular, and A and $b$ expressed as posits (or floats), $x$ is always an exact vector of extended-precision *rationals*.
  - Solution $x$ is rounded to $x^*$, (posit or float)
  - $A \cdot x^*$ is expressible *exactly* using fixed-size quire data type (exact dot product). Call it $b^*$. User can decide if $b^*$ is close enough.

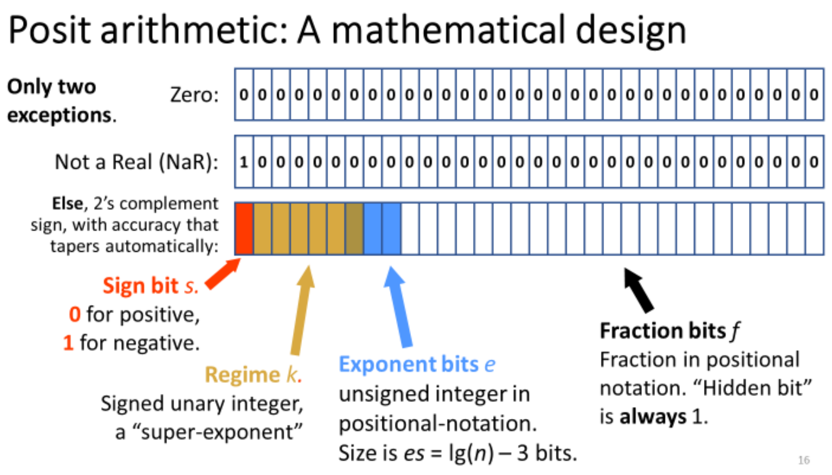
ETAPHASE
A
x
b
Input form is $m \cdot 2^k$, fixed-range integers $m$, $k$
Exact answer form is $p/q$, with variable-precision integers $p$, $q$
A
$x^*$
$b^*$
Rounded answers $m \cdot 2^k$, fixed-range integers $m$, $k$
Fixed-size quire holds *exact* right-hand side, reverse-error analysis

Since the TRADES design problems require solving petascale sparse linear systems involving billions or even trillions of variables, a dot product can easily be a million elements long. Because a float multiply-add operation rounds, its answer will be off by as much as half a Unit in the Last Place (0.5 ULP), and on average will be off by 0.25 ULP. If the roundings are statistically independent, the dot product of two million-long vectors will be off by about 250 ULPs, an unreported loss of many decimals of accuracy. This is one reason why almost all linear algebra for HPC is done using 64-bit floats, not 32-bit floats. However, 32-bit posits have no problem computing the dot product *exactly* in the quire with no possibility of integer overflow, and then converting the sum to a posit with a single rounding, hence an average error of only 0.25 ULP. That's 1000 times as accurate, or three more decimals of accuracy than floats. We have reported the exact inversion of Hilbert matrices using this powerful technique, despite the notoriously high condition numbers of Hilbert matrices. (Side comment: If an exact accumulator is built for 32-bit *floats*, it requires nine or ten 64-bit integers instead of integer power-of-two sizes, which will complicate hardware and slow performance considerably. It is a fallacy to think that floats can use the same quire trick as posits, because the IEEE 754 committee has repeatedly and resoundingly rejected the exact dot product concept ever since the Standard was first established in 1985.)

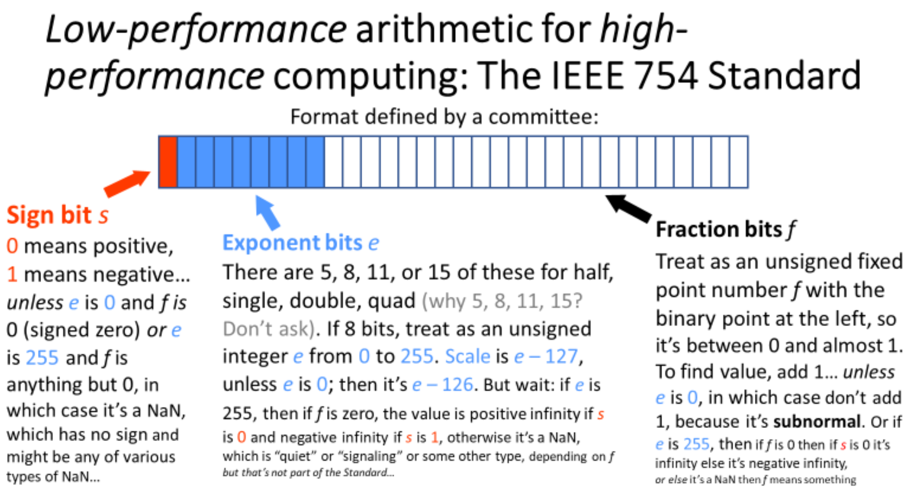
For some linear algebra problems, it is even possible to reduce the precision to 16-bit posits, a fourfold improvement. An unexpected result of our TRADES work was the solution of the classic LINPACK problem (100 equations in 100 unknowns, using random matrix entries and double-precision arithmetic) using only 16-bit posits. Furthermore, while every one of the 100 results for 64-bit IEEE floats showed the accumulation of many rounding errors, the 16-bit posit result was *exact* for every value. This was made possible through use of the quire. (The quire for 16-bit posits is only 128 bits, a mere pair of integer registers.) Note that even a twofold reduction in precision reduces the power requirement per operation by almost 2×, but it also runs in less than half the time; since energy is power × time, the energy reduction can easily be 4×. A further effect is that the memory moves closer to the functional units; if it spilled into L3 cache, for example, perhaps it now stays entirely in L2 cache, which is faster and lower-power per bit moved.

The third major bullet point in the figure above shows another TRADES-funded breakthrough: constructive reverse error analysis. While classic numerical analysis asserts that some methods (like Gaussian elimination) provide the exact solution to a problem "close to" the one that was posed, this rather disingenuously disguises wildly incorrect solutions caused by ill-conditioned systems. The quire makes it possible for the first time to construct what the computed result actually solves. If trying to find an *n*-vector $x$ that satisfies A$x$ = $b$ where A is a given *n*-by-*n* matrix and $b$ is a given vector, we will usually get an approximation, $x_a$. The computation of A$x_a$ = $b_a$ can be performed exactly, using the quire, since it is simply *n* independent dot products. Presenting $b_a$ as well as $x_a$ restores honesty to the computation, by informing the computer user what actual problem was solved with mathematical correctness.

For the curious, the diagrams below contrast the simplicity of posit format…


Posit arithmetic: A mathematical design
Only two exceptions.
Zero:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Not a Real (NaR):
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Else, 2's complement sign, with accuracy that tapers automatically:
Sign bit s. 0 for positive, 1 for negative.
Regime k. Signed unary integer, a "super-exponent"
Exponent bits e unsigned integer in positional-notation. Size is es = lg(n) – 3 bits.
Fraction bits f Fraction in positional notation. "Hidden bit" is always 1.
16

With float format, which at times resembles the US Tax Code in complexity and arcane exception rules:


*Low-performance* arithmetic for *high-performance* computing: The IEEE 754 Standard

Format defined by a committee:

**Sign bit *s***
0 means positive, 1 means negative… *unless e* is 0 and *f is* 0 (signed zero) *or e* is 255 and *f* is anything but 0, in which case it's a NaN, which has no sign and might be any of various types of NaN…

**Exponent bits *e***
There are 5, 8, 11, or 15 of these for half, single, double, quad (why 5, 8, 11, 15? Don't ask). If 8 bits, treat as an unsigned integer *e* from 0 to 255. Scale is *e* – 127, unless *e* is 0; then it's *e* – 126. But wait: if *e* is 255, then if *f* is zero, the value is positive infinity if *s* is 0 and negative infinity if *s* is 1, otherwise it's a NaN, which is "quiet" or "signaling" or some other type, depending on *f* *but that's not part of the Standard…*

**Fraction bits *f***
Treat as an unsigned fixed point number *f* with the binary point at the left, so it's between 0 and almost 1. To find value, add 1… *unless e* is 0, in which case don't add 1, because it's **subnormal**. Or if *e* is 255, then if *f* is 0 then if *s* is 0 it's infinity else it's negative infinity, *or else* it's a NaN then *f* means something

As a final remark about posit design, it is not simply a matter of having more fraction bits for the most commonly-used numbers. Posits are also much more likely to have *no* rounding error when performing a single-argument operation like $1/x$, $x^2$, etc., or a two-argument operation like $x+y$, $x/y$. For this reason, we see many examples where 32-bit posits are 50 to 400 times more accurate in their final result than 32-bit floats.

## 3. TRADES Accomplishments: uDESIGN Seedling

### 3.1. Posit Software Implementations

Concurrently with Gustafson's invention of posits just prior to the program kickoff, he did some initial accuracy-testing in *Mathematica*.

The first subsequent software implementation was performed on the uDESIGN Seedling program by Isaac Yonemoto and presented at the TRADES Kick-Off meeting (MS0:0001AA) as a Julia-based software demonstration (MS1:0001AB for exact-value inputs), posted at this MS1 GitHub link, submitted for review to IV&V (MS2:0001AC), and presented by Etaphase's Isaac Yonemoto to several key stakeholders at the 6-month TRADES Review at NASA-Ames. An archival version of the demonstration has been posted for subsequent viewing at this "asciinema" link.

Etaphase software implementations in Julia, C, and C++ created in large part during the course of this program are posted at the following links:

- julia: https://github.com/interplanetary-robot/SigmoidNumbers (in which posits were originally called "sigmoid numbers" by Yonemoto)
- Hybrid julia/C/C++ posit implementation: posted at FastSigmoid: This is a hybrid julia/C/C++ package which implements sigmoid numbers (aka Type III unums), backed by IEEE floating point. This is in contrast to [http://github.com/interplanetary-robot/SigmoidNumbers] which

implements it as strictly binary operations. This library has been rigorously validated in templated C++ libraries for scientific applications. It supports 8-, 16-, and 32-bit operations.
- Julia/Verilog: https://github.com/interplanetary-robot/Verilog.jl

In a parallel effort at Singapore's Agency for Science, Technology, and Research (A*STAR), Gustafson together with his colleague Dr. Cerlane Leong developed SoftPosit in C with C++ and Python support. SoftPosit, designed to be a drop-in replacement for Berkeley's SoftFloat, generally works almost as fast as SoftFloat for the same precision. (SoftPosit is slightly faster than SoftFloat for multiply, divide, and square root, and slightly slower for add and subtract due to the fact that many float adds and subtracts destroy all significance in one of the input operands and are thus able to exit the routine early.) Basic operations take about 30 clock cycles, not that much slower than early computers that lacked special support for floating-point operations. If an application caches so poorly that most memory references must spend ~200 clock cycles getting data out of DRAM, then it may get performance with SoftPosit very similar to what it would get with native hardware support.

Additional implementations by other teams working globally are flourishing. A comprehensive set of links currently maintained by A*STAR at PositHub provides the links for the following, ever-growing list of languages in which posits have been implemented globally:

- several additional C++ implementations (Posit Research's GitHub by Theo Omtzigt, Clement Guerin's GitHub, Emanuele Ruffaldi's GitHub)
- NumPy
- TensorFlow
- java (NUS's Cerlane Leong's GitHub),
- several Python implementations (NUS's SoftPosit-Python, SpeedGo's GitHub, Ken Mercado's GitHub, University of Washington's Bill Zorn),
- C# for .NET
- Racket
- Rust
- Verilog
- Octave.

### 3.2. Time-complexity Characterization

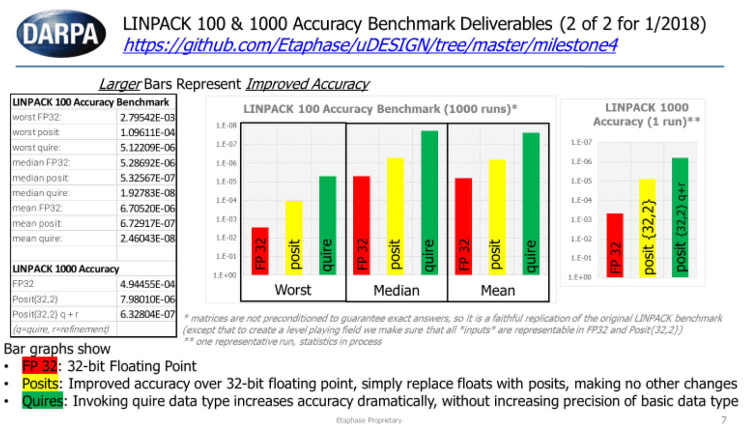
Characterization of time-complexity of posits (MS3:0001AD) as a function of increasingly large array sizes indicated exponential growth in time-to-compute with the order *N* of an *N*-by-*N* linear algebra matrix. The time complexity appears to grow on the order of 100× per *N*. This trend appears to be insensitive to the precision of the posit number system. The results on time-complexity of posits are posted at this GitHub link and in the associated monthly report at this TRADES Sharepoint Link.

### 3.3. Benchmarking: Linear Algebra (LINPACK)

One thing that floats do well is to guess answers *x* to a system of linear equations, A *x* = *b* where A is an *n*-by-*n* matrix and *b* is a given vector of *n* values. (At least, they do well *under certain conditions*, and people have learned to live with those restrictions).

The LINPACK benchmark fills A with random numbers between –1 and 1, then assigns the sum of each row to the values of *b*, so that the answer should be *x* = {1, 1, …, 1}. Etaphase's Benchmarking *milestones* included the following: Initial Benchmarking (MS4:0001AE), incorporation into benchmarking of Inexact Value Inputs (MS5:0001AF), Accuracy Benchmarking (MS6:0001AG), and Advanced Benchmarking (MS7:0001AH).

This series of benchmarking tests provided compelling proof of the accuracy of posits in linear algebra. Initial Benchmarking presented at the January 2018 TRADES Review provided earliest evidence of the accuracy of posits. These initial benchmarking efforts, using the built-in linear solver that comes with the julia language, proved the improved accuracy of posits and quires over floats. An apparent lack of perfect accuracy in the quire was shown to be due to an inaccuracy in julia's linear algebra solver, rather than due to an inaccuracy in the quires. Accuracy results for posits, displayed as yellow bars in the figure below, prove that for a thousand runs of LINPACK 100 (where 100 indicates a 100-by-100 matrix), posits consistently provide over an order of magnitude accuracy improvement over floating point (for the same bit size, in this case 32 bits). A similar result shows over an order of magnitude accuracy improvement for posits with a LINPACK 1000 test of a 1,000-by-1,000 array. From this same test, accuracy with quires (results displayed by the green bars in the figure below) proved more than an additional order of magnitude improved in accuracy as compared to posits alone. This interestingly differed from the expected perfect accuracy of quires, pointing out the importance of using appropriately-chosen, rigorously correct solvers in order to prove the absolute accuracy of quires.


LINPACK 100 & 1000 Accuracy Benchmark Deliverables (2 of 2 for 1/2018)
https://github.com/Etaphase/uDESIGN/tree/master/milestone4

*Larger* Bars Represent *Improved Accuracy*

| LINPACK 100 Accuracy Benchmark | |
|---|---|
| worst FP32: | 2.79542E-03 |
| worst posit | 1.09611E-04 |
| worst quire: | 5.12209E-06 |
| median FP32: | 5.28692E-06 |
| median posit | 5.32567E-07 |
| median quire: | 1.92783E-08 |
| mean FP32: | 6.70520E-06 |
| mean posit | 6.72917E-07 |
| mean quire: | 2.46043E-08 |
| | |
| **LINPACK 1000 Accuracy** | |
| FP32 | 4.94455E-04 |
| Posit{32,2} | 7.98010E-06 |
| Posit{32,2} q + r | 6.32804E-07 |
| (q=quire, r=refinement) | |

* matrices are not preconditioned to guarantee exact answers, so it is a faithful replication of the original LINPACK benchmark (except that to create a level playing field we make sure that all *inputs* are representable in FP32 and Posit{32,2})
** one representative run, statistics in process

Bar graphs show
- FP 32: 32-bit Floating Point
- Posits: Improved accuracy over 32-bit floating point, simply replace floats with posits, making no other changes
- Quires: Invoking quire data type increases accuracy dramatically, without increasing precision of basic data type

Etaphase Proprietary 7

Demonstration of the perfect accuracy for which quires were designed was proven as part of the Advanced Benchmarking Milestone (MS7:0001AH). A rigorously correct linear solver was written in *Mathematica*. The histogram charts displayed below show computed solutions of the above-described LINPACK accuracy benchmark for 100-by-100 arrays using 16-bit floats (on left) and 16-bit posits (on right)[8]. As the arrays are chosen such that the correct answer is unity, these histograms clearly display the LINPACK accuracy of 16-bit posits as compared to 16-bit floats.

Invoking the quire, solving for the residual and adding that to the posit provides the *exact* answer when using a mathematically-correct solver rather than the built-in Julia solver.

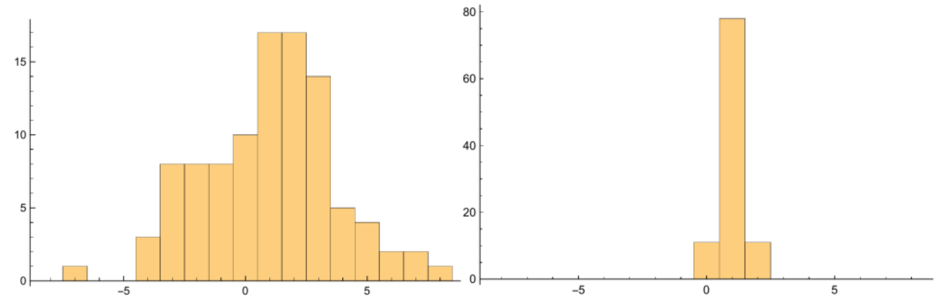


*Figure 1: 100-by-100 LINPACK Accuracy Benchmark for 16-bits, showing histogram of solutions for 16-bit floats (left) and 16-bit posits (right). Correct answer is unity. Left: 16-bit IEEE Floats (a.k.a. Half-Precision). Right: 16-bit posits (before solving for the residual to get the exact answer).*

With 16-bit floats, we cannot even get two decimals of worst-case accuracy, even when applying correction techniques. The correction technique can only help so much, because you really need the exact dot product (which has been repeatedly proposed to the IEEE 754 committee and repeatedly rejected, as mentioned above).

Increasing the precision of the floats to 64-bits improves the accuracy of the 100-by-100 LINPACK, but it still gets the wrong answer for every single one of the 100 entries in the answer vector. Using *16-bit posits* provides the correct answer for every single entry. The choice is therefore quite clear between posits and floats as to which is number system provides better accuracy: posits!

Despite the use of residual correction, posits would almost certainly be *much* faster time to solution, while using only 1/4 the storage. Solving $N$-by-$N$ dense linear systems takes order $N^3$ work, but residual correction only requires order $N^2$ work.

Increasing the matrix size to 300-by-300 using 16-bit posits again provided the exact answer. In contrast, the use of 16-bit floats caused the matrix to look singular at this low precision. That is, all entries came up "Indeterminate" when using 16-bit floats to solve a 300-by-300 matrix. Solving the 300-by-300 matrix with floats required increasing the precision to 32 bits.

Further increasing the matrix size to 1000-by-1000, 32-bit posits provided the correct answer with two refinement iterations, whereas 32-bit floats only provided 1.6 decimals of accuracy.

The associated *Mathematica* Notebook[8] provides additional details including the entries of the matrices and intermediate steps.

## 3.4. Benchmarking Beyond LINPACK

### 3.4.1. Hilbert Matrix Inversion with and without LDL Solver

Posits were additionally benchmarked against a notoriously ill-conditioned matrix. The *Hilbert matrix* is so well-known as a tough problem for floating-point arithmetic; it is the canonical ill-conditioned matrix, and it has serious applications, so it is not simply a problem contrived to be a challenge. It arises in trying to form least-squares polynomial approximations to arbitrary functions.

The usual Hilbert matrix $H$ is a square matrix with entries $h_{i,j} = 1 / (i + j - 1)$. For example, the 4-by-4 Hilbert matrix is

$$H = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}$$

The row vectors are nearly parallel, increasingly so as the dimension of the matrix increases; this means a conventional Gaussian elimination solver, even with partial pivoting, will tend to produce large rounding errors in computing the inverse. With quires, the exact mathematical inverse, conveniently comprising exact integers, was obtained.

Even without using the quire for residual correction, posits were found to be between 30 and 432 times more accurate than floats, depending on the order of the matrix and the choice of solver.

Posits (with quire help) were benchmarked against floats for inversion of these ill-conditioned Hilbert matrices. A posit-enabled LDL Solver invoking the quire was shown to provide perfect (exact) results for the inverse of the Hilbert matrix. Floats cannot do this. Details are described in [9].

### 3.4.2. Integration of Posits with B-spline fit routine

A second demonstration of posit accuracy outside of basic linear algebra was the posit-enabling of B-spline polynomial fitting. Etaphase coordinated with both MIT's Bob Haimes who originally wrote the B-spline fitting code to work with conventional IEEE 754 Floating Point and Dr. Theo Omtzigt, an Etaphase colleague at Stillwater Computing who wrote a C++ posit library that is particularly well-tested and that interestingly allows posit sizes that are not an integer power of two. As posit-enabling of the MIT code was formally outside scope of Etaphase's seedling, Dr. Omtzigt performed the work on a "pro bono" basis. His conclusion was that while the posit<48,3> was sufficiently strong for this iterative algorithm, posit<40,3> was not sufficiently strong.

This work positions MIT's Haimes to benchmark posits against floats in his B-spline fit code.

Steps followed in posit-enabling MIT's B-spline code were described by Omtzigt as follows:

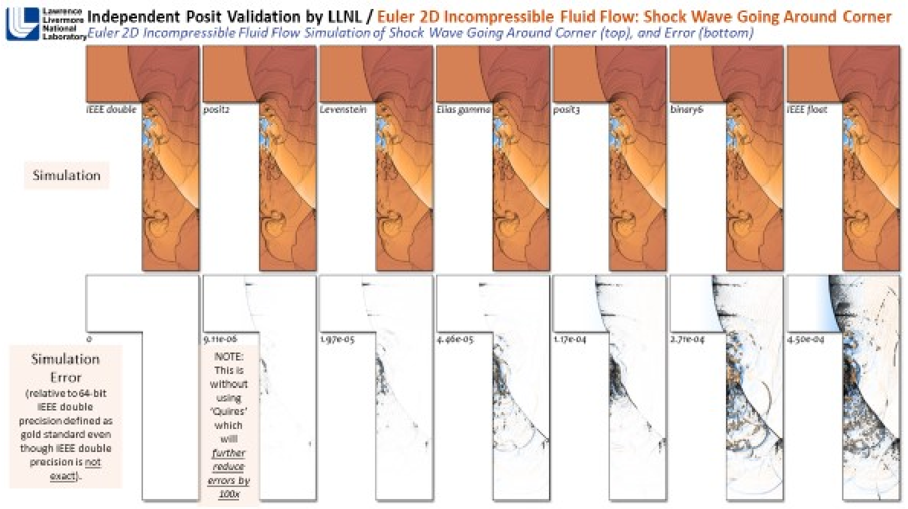
> "Robert Haimes provided a code drop that contained the spline code and a test data set. The goal was to posit-enable the spline code. We needed two modifications to enable posits. The first step involved making the overall code a pure C++ code. The test driver was a C program that was calling a C++ spline function. The new structure had to be a pure C++ program with a templated spline function. The second problem we had to resolve was more subtle, and involved removing a type-conversion that was blocking the templated posit type. Simply making the code adhere to basic arithmetic requirements sufficed to enable the posit type. The resulting code change after that to enable posits was literally a single line of code, as demonstrated here:

```
using Scalar = sw::unum::posit<40,3>;
stat = spline2dAppr<Scalar>(endc, imax, jmax, xyz, uknot, vknot, vdata, wesT, easT,
south, snor, north, nnor, tol, &idata[0], &rdata);
```

"The subsequent experimentation to compare posits to double precision demonstrated that a 40-bit posit has same convergence rate and accuracy as a 64-bit double precision IEEE floating point. This is a direct replacement of a double with a posit<40,3>. No modifications of the algorithm were done. This code consists of hundreds of interpolations which could be made more accurate when transformed into quire-enabled expressions, which would further reduce the required size of the posit representation."

### 3.4.3. Beyond Linear Algebra

A particularly early demonstration of posits in a nonlinear physics application was independently performed by LLNL's Peter Lindstrom[10] subsequent to some initial guidance from Gustafson and Yonemoto. The physics was non-linear fluid dynamics of turbulent gas flow in a shock wave around a corner. Results displayed in the figure below were presented as a keynote at the 2018 Conference on Next Generation Arithmetic (CoNGA). The LLNL team's shock fluid flow computer accuracy test considered IEEE double precision to be "ground truth." Simulation errors with the "posit2" number representation (without invoking the quire) were shown to be consistently smaller than with the alternative number representations. Invoking the quire would provide an estimated further 100× reduction in errors.



## 3.5. Hardware Implications

Proving the anticipated energy savings and computational speedup of posits relative to floats requires the development of posit-enabled hardware optimized for energy-efficient performance with posits. While hardware developments were outside the scope of Etaphase's uDESIGN Seedling, the seedling's final milestone MS8:0001AJ, entitled "Hardware Architecture Implications," tasked Etaphase to evaluate implications of hardware architecture on unum performance.

This milestone was originally worded to address a challenge associated with Type I unums (before the discovery of the substantially more hardware-friendly Type III unums known as posits, quires, and valids). Type III unums have fixed size, and use almost precisely the same circuit elements as IEEE floats, but simplified.

MS8:0001AJ was a modest proposal since, as it was originally written, it would have merely enhanced the library to indicate what hardware costs can be. With the help of the international community doing research on Type III unums, we have blown past this milestone with the design of a "Parameterized Posit Arithmetic Hardware Generator"[11]. Additionally, the cost of the quire register is now addressed in "RISC-V for posits," which indicates in detail the extensions needed to incorporate both posit and quire types into a RISC-V processor, following the format of other RISC-V extension definitions.

In this way, the "Hardware Architecture Implications" milestone was both met and substantially exceeded. Multiple institutions currently in the process of creating posit-enabled hardware are identified in the previously- reported Appendix to this document. The first sale of a posit-enabled FPGA occurred in March 2018 (Calligo Technologies to A*STAR).

## 4. Global Developments: How are posits implemented currently?

4.

### 4.1. Artificial Intelligence (AI): Posits Implementation Status in AI

A Facebook hardware design team experimented with the use of low-precision posits for AI applications[12]. They were able to drive the precision down to 5-bit posits, and the custom hardware they built is six times as power-efficient as the usual 16-bit data types used for Deep Learning. Surprisingly, they went public with this result instead of keeping it to themselves as a competitive advantage.

Even more surprising, Google engineers have responded by including posit support as well. In a paper submitted to the upcoming CoNGA conference, they announce that they will be adding posits to their "B-floats" and fixed-point formats used in TensorFlow, and presumably their TPU as well.

Researchers at the Rochester Institute of Technology have written multiple papers showing the excellent fit of the posit accuracy "tent" shape to the histogram of values used in AI, both for Deep Learning and for inference. NUS researchers confirmed the RIT results for inference, using the MNIST benchmark to achieve a 99.16% accuracy rate with 9-bit posits. MNIST using 32-bit floats has an accuracy rate of 99.11%. None of these results exploits the quire for accumulations; that should enable even higher accuracy, and even lower precision posits can meet the needs of AI with a major savings of energy and power requirements compared to the IEEE 754 16-bit floats used by Nvidia, or the 16-bit "B-floats" used by Google.

**Commented [JG3]:** I understand "754b" probably refers to the binary standard separate from the decimal version of IEEE floats, but this hasn't been defined anywhere, so just use 754. There is no 16-bit decimal float anyway, so there is no possibility of confusion.

### 4.2. IoT: Current status of posits in Internet of Things (IoT) Development Efforts

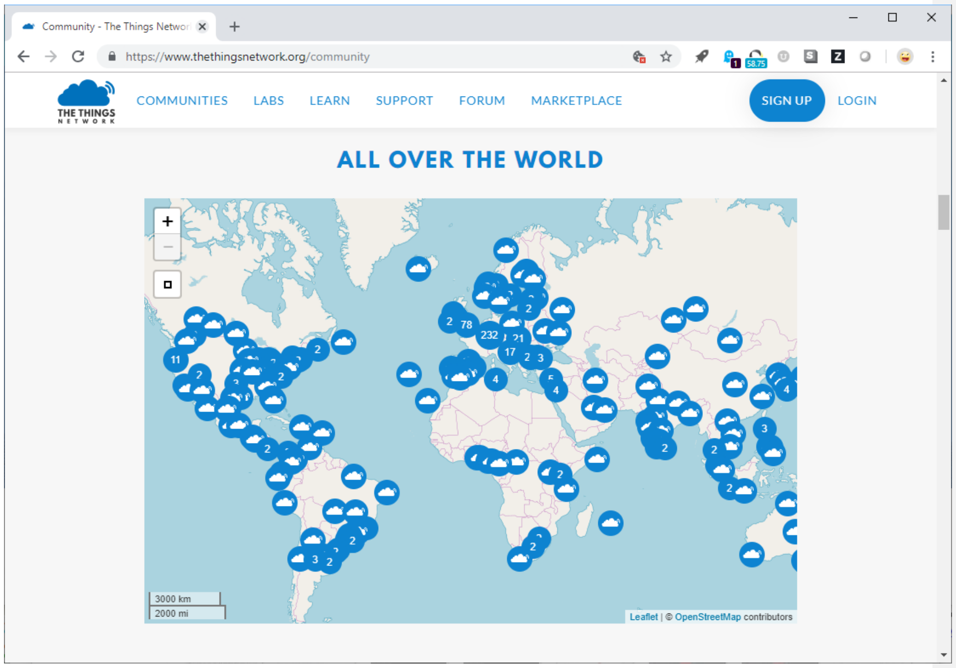
The Internet of Things, and the Industrial Internet of Things, also known as Industry 4.0, are expected to drive the next phase of information technology, where assets and processes are instrumented to generate digital information that can be transformed by sensor fusion and artificial intelligence processes to generate actionable intelligence to optimize a plurality of decision-making processes for business, defense, and other enterprises.

The energy efficiency of posits as compared to IEEE floating point is expected to drive differentiated solutions that will have large scale benefits.

*What is posit status today in Commercial IoT Development Efforts?*

Industrial incumbents, such as Robert Bosch and Huawei, in 2018, already had research projects underway to study and quantify the benefits of posits in their embedded platforms. Unfortunately, the results are kept secret to benefit European and Chinese interests.

Stillwater Supercomputing is developing a posit-enabled IoT gateway that is built on eFPGA platforms from Xilinx, Intel, Achronix, and Flex Logix. The goal is to deliver a 'smart' IoT gateway that uses posits in the sensor fusion algorithm to create collective intelligence applications of attached sensor networks. This platform is being developed for two different use cases. The first use case is that of an intelligent node in a global IoT network, the ThingsNetwork (https://www.thethingsnetwork.org/).


Community - The Things Network
https://www.thethingsnetwork.org/community
THE THINGS NETWORK
COMMUNITIES
LABS
LEARN
SUPPORT
FORUM
MARKETPLACE
SIGN UP
LOGIN
ALL OVER THE WORLD
3000 km
2000 mi
Leaflet | © OpenStreetMap contributors

This will be a LoRaWANsensor network (Long Range Wide Area Network) across the world, enabling commercial entities to add value in the form of easy-to-use applications. Temperature monitoring for commercial freezers in food preparation environments, precision irrigation systems, and air-quality monitoring are examples of anticipated early commercial use-cases. Stillwater is engaged in expanding this to forest resource management (in collaboration with Bowdoin College with initial focus on Maine forests in the vicinities of Portland, and Brunswick).

A substantially higher data-rate use case for the intelligent IoT gateway is a computer vision 'brain' for deployments in autonomous vehicles, roaming robotics applications, and retail applications, such as Amazon's cashier-free stores. This involves a collection of computational science components that will be enhanced with posits to deliver power-savings for this high-performance embedded application. The algorithm we are posit-enabling is YOLO, You Only Look Once, which is a building block in the self-driving car vehicle detection data set by Udacity.

The YOLO architecture is a fully convolutional neural network (FCNN). It passes the image ($n \times n$) once through the FCNN and outputs a prediction of ($m \times m$) elements. The YOLO architecture splits the input image in an $m \times m$ grid, and for each grid element generates two bounding boxes and class probabilities for those bounding boxes. Note that bounding box is likely to be larger than the grid itself. From the paper:

*We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities.*

*A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection. First, YOLO is extremely fast. Since we frame detection as a regression problem, we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency.*

*Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.*

*Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.*

*Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision.*

*Our system divides the input image into an $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.*

*Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $Pr(Object) * IOU$ . If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.*

*Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, Pr(Classi |Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.At test time we multiply the conditional class probabilities and the individual box confidence predictions*

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU} = \Pr(\text{Class}_i) * \text{IOU}$$

*which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in* the *box and how well the predicted box fits the object.*

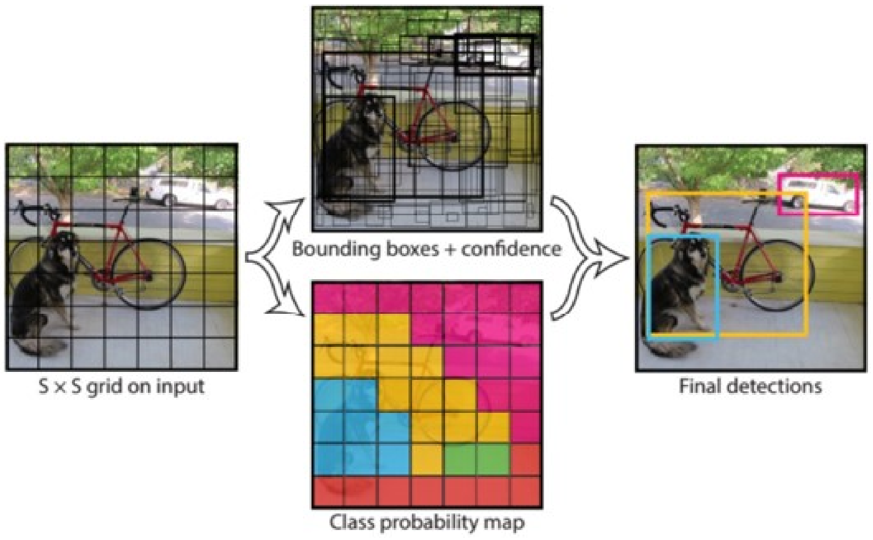


*Figure 2: The Model. Stillwater's YOLO Fully Convolutional Neural Network (FCNN) models detection as a regression problem. It divides the image into an S x S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S x S x (B*5 + C) tensor.*

## *What is new in Stillwater's approach and why does Stillwater think it will be successful?*

Stillwater expects that posits will enable a much more efficient hardware pipeline because posits enable a much smaller data path. Taking maximum advantage of posits demands complete reimplementation of the YOLO algorithm. Stillwater has a collaboration with the Chalmer's Self-Driving Truck team at the University of Gotenburg, Sweden, (https://www.chalmers.se/en/news/Pages/Self-driving-trucks-2016.aspx), the goal of which is to deliver a posit-enabled visual system for their self-driving truck research.

## 4.3. HPC (Singapore)

Singapore researchers tested 16-bit posits for Fast Fourier Transforms (FFTs) and found that they could achieve forward-inverse transforms that restored every bit of an original 11-bit signal perfectly[13]. This is not achievable with 16-bit floats, which is why almost all signal and image processing programs are forced to use 32-bit precision when using floats. Since FFTs are clearly a bandwidth-limited operation, this has the potential to double the performance of a broad range of signal and image processing tasks while reducing storage and energy requirements.

Singapore works with the Square Kilometre Array project based in New Zealand, Australia, and South Africa. SKA consultant Peter Braam (creator of the Lustre file system) has discovered that their ambitious radio telescope design will be taking in 200 petabytes per second of data coming over the horizon as the Earth rotates, which will consume 12 megawatts moving the data from DRAM memory into the processors. The entire power budget for their computer is 10 megawatts, creating a crisis. Braam has analyzed the use of posits instead of floats and discovered that FFTs using posits will radically reduce the data sizes required and enable the billion-dollar project to go to completion. Posits will also save the program $18 million, Braam estimates.

Singapore's National Supercomputing Centre (NSCC) has as its most important workload the GROMACS molecular dynamics program. Singapore researchers spoke with the developers of GROMACS, who have trimmed the float precisions down to 32 bits wherever they can, and the GROMACS developers believe 16-bit posits can yield sufficient accuracy. FFTs are a major component of GROMACS, as they are with many computational chemistry codes; the other major component is dot products. Posits excel at both operations, so this approach bears watching.

The rate of posit adoption in HPC is astonishingly high, given that posits were only introduced publicly in February 2017[4], less than two years ago.

## 4.4. Other Posit Adoption Worldwide

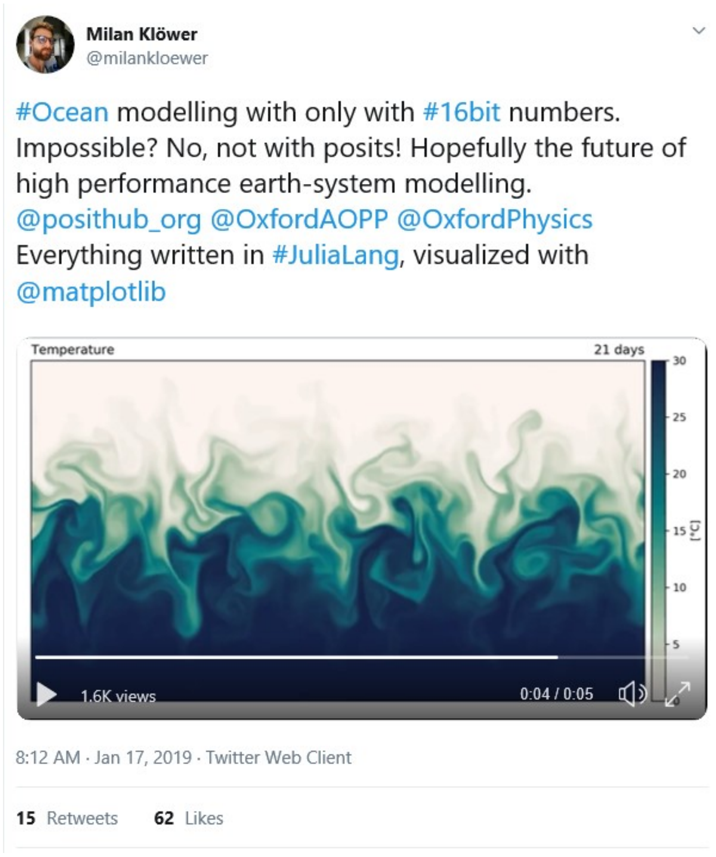
Peter Hofstee, designer of the Cell processor and one of IBM's highest-ranking engineers, endorses posits and is urging that future POWER processors be made dual-mode to support posits. He estimates that the cost of supporting both forms of arithmetic will be one additional clock cycle in the pipeline at most, but that the throughput will be the same.

Satoshi Matsuoka, of the Tokyo Institute of Technology and now Japan's leading supercomputer expert in charge of the Post-K design, believes Fujitsu should add support for posit arithmetic to its A64FX version of the ARM processor. Matsuoka is primarily interested in posits for AI, but also recognizes what they can do for HPC applications in general.

The European Processor initiative plans to pursue an ARM-based processor designed in Barcelona as an alternative to Intel's processors, and an accelerator designed by French company Kalray. Kalray's CTO, Benoit DuPont de Denichen says the initial accelerator will use floats, but the follow-on accelerator will use posit format.

Researchers at Oxford University have recently discovered that weather and climate models, which are multi-physics applications that typically use 64-bit float arithmetic, can run successfully using 16-bit posits. Their paper is pending publication and presentation at CoNGA (Conference on Next-Generation

Arithmetic). Displayed below is a screenshot of the team's preview Tweet of this upcoming paper, announcing that posits enable ocean modeling with 16 bit numbers.



Milan Klöwer
@milankloewer

#Ocean modelling with only with #16bit numbers. Impossible? No, not with posits! Hopefully the future of high performance earth-system modelling. @posithub_org @OxfordAOPP @OxfordPhysics
Everything written in #JuliaLang, visualized with @matplotlib

8:12 AM · Jan 17, 2019 · Twitter Web Client

15 Retweets 62 Likes

Micron Technology has a hardware-software team at their Boise headquarters designing posit circuitry and math library software. Public statements indicate they may want to create "smart memory" that stores floats by converting them to lower-precision posits, converting them back to floats on loads from memory.

## 5. Suggested Next Steps

### 5.1. Standardization

*What are you trying to do? Articulate your objectives using absolutely no jargon.*

We aim to shorten the widespread adoption of posit to five years, by coordinating with NIST to lead the official standard definition. There are several ways to achieve this.

*How is it done today?*

A Draft Posit Standard exists [Appendix B], along with an international committee of experts contributing and insuring agreement on decisions. Its page count is about 1/8 that of the IEEE 754 Standard because of sweeping simplifications and adherence to mathematical principles instead of human preferences.

*What are the limits of the current Draft Posit Standard?*

It is unfinished in the areas of language and debugger support, and the rules for formatted (human-readable) posit output.

*What is new in your approach and why do you think it will be successful?*

The committee has been chosen to avoid undue influence by commercial computer makers who have an interest in bending the rules to more closely match their existing product plans or to grant themselves competitive advantages, since those kinds of commercial interests led to many of the shortcomings of the IEEE 754 Standard that persist to this day. Instead of going through the IEEE standardization process, we recommend going directly to NIST to create an ANSI Standard definition for posit arithmetic. The brevity and lack of contention in the Draft Posit Standard should make this possible. DARPA may be able to lend its influence to make this happen.

Gustafson has created a document describing an RISC-V extension to support 32-bit posits, patterned after documents that describe extensions for 32-bit and 64-bit floating-point arithmetic. It can be found at the following URL:

https://posithub.org/docs/RISC-V/RISC-V.htm

*Who cares? If successful, what difference will it make?*

With standards in place, companies such as Intel and IBM can be confident in altering their product roadmaps to allow a mode switch in future floating-point units in their chips that will allow either posit or float operations. This parallels how other transitions have been handled historically, such as IBM's transition from big-endian to little-endian storage, or Intel's transition from 32-bit to 64-bit addressing. Over the period of perhaps a decade, both number systems will coexist since backward compatibility will still be needed, but customers will want the advantages of the posit format and will need the posit format to achieve competitive performance.

## 5.2. Software Development

### 5.2.1. Math Libraries

*What are you trying to do? Articulate objectives using absolutely no jargon*

We aim to build the posit-enabled math libraries, which will reduce to five years the time required for widespread adoption of posits.

*How is it done today?*

Posits are designed as drop-in replacements for floats. Multiple implementations of posits now exist in support of the basic arithmetic operations like plus-minus-times-divide. This suffices for conversion of linear algebra and FFTs.

*What are the limits of current practice?*

In general, algorithms rely on language support for math libraries to perform such operations as exp, cos, sin, tan, log, and so on. The Draft Posit Standard requires that these functions be rounded correctly for *all* input arguments, something that most vendors do not presently do because their approximation methods pay a steep performance price if they require perfection. The so-called "Table-Maker's Dilemma" is that transcendental functions can happen to produce values that lie extremely close to the tiebreaking value between rounding up or rounding down, which seemingly then requires millions of bits of accuracy to resolve.

**Commented [JG5]:** I had to double-check this, but it can indeed require tens of millions of bits per variable to compute elementary functions to the required accuracy, by traditional methods. Mathematica actually does this, which is why it can be used as the Gold Standard. What people fail to notice is that the Gold Standard only has to be computed once in designing a math library to make sure everything is correct; it does *not* need to be computed every time the functions are called.

*What is new in your approach and why do you think it will be successful?*

Gustafson has recently shown that a new approximation technique, the "Minefield Method," can produce perfectly-rounded math functions with fast, relatively low-order polynomials. This eliminates the tradeoff between speed and correctness, making that part of the Draft Posit Standard quite practical to implement. However, there is considerable work to be done, similar to the investment of time that has gone into creating the math library routines for floats of various precisions. Once C, Fortran, and other languages have standard math libraries for posit functions, posits can truly become a drop-in replacement for floats. A DARPA investment to develop those libraries can shorten to five years the wide-spread adoption of for wide-ranging applications in HPC.

### 5.2.2. Geometric Representations

*What are you trying to do? Articulate objectives using absolutely no jargon.*

Demonstrate the accuracy of posits for geometric model representations having physically important design features covering six orders of magnitude in span of scale.

*How is it done today, and what are the limits of current practice?*

One of the most important kernels in computational geometry is to determine on which side of a line a point lies. The task is equivalent to taking a cross product (or a 2-by-2 determinant) and examining the sign of the result. When the point is close to the line, floats frequently get an incorrect result because of accumulated rounding errors from the multiply-multiply-subtract operation needed. This leads to "light leaks" in graphics programs, incorrect or irreproducible crash simulations, and many other difficulties.

### *What is new in your approach?*

Posits perform the 2-by-2 determinant in the quire, free of rounding error, so the sign is always correct when the quire is converted back to a posit. This is a potential game-changer for the entire field of computational geometry since it drastically reduces the precision needed to get excellent results.

Sandia's tpetra and Trilinos systems, which were designed to support enormous spans of scale in computational geometry, are now poised, as a result of the TRADES uDESIGN Seedling, to become posit-enabled.

### *Why do you think it will be successful?*

A pro bono posit-enabling of MIT's B-spline program covering a limited span of scale was performed by Stillwater's Theo Omtzigt in collaboration with MIT's Bob Haimes during Etaphase's DARPA TRADES uDESIGN program. This demonstration was basic proof-of-concept performed on desktop machines, for a geometric representation covering a very limited span of scale.

We also think that posit-enabling of these systems will be successful because of our success with posit-enabling MIT's B-spline software referenced above. As well, we have spoken with Sandia's SME on these systems and he has agreed that it is feasible.

### *Who cares? If successful, what difference will it make?*

The entire computational geometry ecosystem from airplane design to augmented reality systems and simulations will see 2× and more reduction in both memory capital costs and memory access power budgets. Computational geometry calculations of critical national importance currently underway at national labs such as Sandia, will immediately see a 2× or more efficiency improvement from the posit-enabling of tpetra and Trilinos.

### *What are the risks?*

A risk is that the speed-up advantage of posits may not be realizable until the tests can be run on posit-enabled hardware (described below).

### *How much will it cost?*

The effort can be kicked-off with a $200,000 extension to the current uDESIGN Seedling.

### 5.2.3. Multi-physics CAD

*What are you trying to do? Articulate our objectives using absolutely no jargon.*

We aim to posit-enable the simulation and physical optimization of complex geometries, starting with electrostatics and electromagnetics before incorporating mechanics. We will then leverage lessons-learned from other groups who have posit-enabled fluid dynamics and heat flow to add these two additional physics to our multi-physics model.

*How is it done today?*

LLNL's Peter Lindstrom has benchmarked the precision of posits in a nonlinear fluid dynamics application involving the propagation of a shock fluid flow wave around a sharp corner.

As mentioned in Section 4 above, there is a group in the UK currently working on the application of posits to weather modeling. This Oxford weather and climate forecast model based on posits mentioned in section 4.4, will evidently be the first posit-enabled multi-physics model.

*What are the limits of current practice?*

Posits have not yet been applied to the physical simulation and optimization of mechanics, electrostatics, or electromagnetics.

*What is new in your approach and why do you think it will be successful?*

As a result of the TRADES program, we now know how to refine the solution to the linear systems that arise in CAD and other applications, which then allows the use of fewer bits per data point without sacrificing confidence in the answer.

Preliminary evidence from early results of the above-referenced Oxford weather and climate forecast indicates that 16-bit posits may be up to the challenge of such complex, chaotic simulations.

*Who cares? If you are successful, what difference will it make?*

Examples of complex multi-physics simulations of critical importance to our nation's security include electronic and photonic integrated circuits, voltage multipliers, high data rate wireless communications systems for operation between fast-moving platforms, radar systems, aircraft design, rocket propulsion, and the kinds of simulations for which organizations such as Sandia are responsible.

DARPA support can shorten to five years the availability of posit-enabled multi-physics software simulators for electromagnetics.

*What are the risks?*

The development of these multi-physics simulations may not be embraced by commercial CAD companies until posit-enabled hardware becomes widely adopted.

### 5.2.4. Automation of Computer Resource Optimization

*What are you trying to do? Articulate your objectives using absolutely no jargon.*

We aim to automate optimization of the following computer resources: processors, network and memory. Our goals include beneficially enabling software systems to automatically adapt themselves to constantly-changing hardware resources, whether the changes are due to obsolescence of original computer hardware systems, or due to variabilities in the availability of computer resources. This would facilitate the maintenance and upkeep of software systems associated with legacy DoD platforms, and would improve the adaptability of computational resources to the ever-changing performance characteristics of communications channels (which have data rates and reaches dependent on factors such as rain, smoke, and fog); processor performance (with lower processor performances available locally, and extraordinarily high performance processors available remotely in the event that a sufficiently high bandwidth communications channel is available to access the processor(s); and memory (with potentially very sensitive dependence not only on the available communications channels, but also on the amount of power and energy available to move data between processor and memory).

*How is it done today, and what are the limits of current practice?*

Currently, software optimization is primarily a human effort requiring extraordinarily high levels of expertise in computational mathematics and deep understanding of available computer hardware resources.

In most HPC and data center applications, significant hardware upgrades require a complete re-writing and/or re-optimization of associated software systems. Limitations of this current practice include challenges with finding sufficiently expert computational scientists to fine-tune software programs to perform optimally when presented with new hardware resources. And the use of human software engineers introduces significant prospects both for human error and for software security risks.

One of the main reasons that software does not maintain its planned usefulness for as long as originally anticipated is that programs make implicit assumptions about the relative cost of operations; when hardware changes improve those costs unevenly, the program must be rewritten to once again adapt to a new set of assumptions. For example, when matrix-matrix multiplication is done by "tiling" the matrix and using block-based algorithms, the blocking decomposition is tuned to the cache size and speed, as well as to network bandwidth and latency. The tuning has not been parameterized and automated because there has not been a sufficiently accurate and complete general model of computer performance of tensor operations. Dr. Lenore Mullin mathematically demonstrated the feasibility of perfect performance prediction for tensor operations[14][15]. This leads to the possibility that programs can be made optimal analytically. The failure of computers to follow the associative and distributive laws of arithmetic has limited the application of Mullin's tensor mathematics to specific case studies, carefully reviewed in each case by an expert computational mathematician continually watching out for floating point errors. This has limited the application of Mullin's tensor mathematics to specific problems of particularly high interest to DoD.

**Commented [JG6]:** I think the document consistently uses "Dr." when introducing people for the first time, if they have a PhD.

The above-referenced problems are significantly-compounded when the hardware upgrade is on the processors, network, and/or memory of a highly-specialized DoD system, for which there are even fewer individuals having the appropriate level of expertise to perform the software optimization required in the event that a hardware upgrade (or repair) brings with it any changes in the processor speed, network, or memory resources.

*What is new in your approach, and why do you think it will be successful?*

We express the computer resources as a tensor (processors, network, and memory), leverage the associative and distributive properties of posits, and apply computational mathematics to the optimization of the distribution of computation across those resources.

A key to the approach is to unburden the compiler from the task of producing a “one size fits all” mapping of any array algorithm to the vast range of computer architectures that exist. That role instead shifts to the “installer” program, a modernized version of the classic Unix “makefile,” that contains the sophisticated mathematical models of indexing and data motion complexity. This has the potential to save massive amounts of wasted programmer time rewriting software every few years, saving DARPA and everyone else much if not most of their software costs.

*What are the mid-term and final “exams” to check for success?*

The approach can be prototyped on simple multiprocessor servers, and then advanced to more dynamically-reconfigurable systems such as data centers and the complex distributed computational resources relevant to DoD missions.

### 5.2.5. Artificial Intelligence

*What are you trying to do? Articulate your objectives using absolutely no jargon.*

We seek to demonstrate posit-enabled accelerators for Artificial Intelligence applications.

*How is it done today, and what are the limits of current practice?*

Google and Microsoft are implementing proprietary approaches to reducing the number of bits required in AI applications. Both Google [16] and Microsoft have designed and are deploying at-scale domain specific processors to aid in tensor processing specifically for Deep Learning applications in their clouds.

Additionally, mobile chip makers such as Apple, Samsung, and Qualcomm all have announced inference engines. It is very clear that the leaders in the industry have progressed to building custom hardware solutions to strategically differentiate their services.

*What’s new in your approach and why do you think it will be successful?*

We will demonstrate the power of posits in DoD-specific applications and platforms having performance requirements different than Facebook.

*Who cares?*

DoD needs increasingly smart weapons systems in order to minimize loss of life while at the same time maintaining dominance on land, in the air, and by sea.

*If you’re successful, what difference will it make?*

Smarter AI systems available to DoD will increase the nation’s security, while minimizing loss of life.

## 5.3. Hardware Development

Etaphase has identified a powerful plan to build a US-based team positioned to shorten to five years the normal 20-year time span required for commercial adoption of new computer number systems. We suggest a five-year program to design, build, and deploy energy-efficient posit-enabled chips. We suggest the following parallel development of both FPGA-based and ASIC approaches:

- The FPGA approach will be led by Theo Omtzigt with a focus on early adoption of posits in IoT systems, where the energy savings of posits are particularly compelling and where rapid and wide-spread anticipated commercial market adoption promises to helpfully drive down costs of hardware for DoD moving forward (at least once the initial large-scale demonstrations are performed by DoD).
- The ASIC Approach for which we have a potential customer with interest in using a posit-enabled processor in a commercial 5G application will leverage the deep experience of the former REX Computing team and will provide DoD with a path to early adoption in microwave radar and communications systems.

Accelerated insertion into DoD systems provides the following advantages expected to have beneficial strategic implications cascading throughout the defense establishment:

- energy savings: 2× (and more) energy savings
- concomitant battery weight reduction
- improved execution speed of 2× or more for bandwidth limited calculations,
  elimination or dramatic reduction of floating-point errors, streamlined, machine-driven software optimization including dynamic resource allocation via tensor optimization across processor, memory, and network, leading to substantial reductions in human coding errors

### 5.3.1. FPGA-based posit-enabled tensor accelerator (PETA) for HPC

Scientific discovery and engineering optimization through advanced computing is increasingly a key strategy; in both corporate and national strategic interest. Organizations such as Google and Baidu have invested in custom high-performance computing hardware to drive their products and services that differentiate through artificial intelligence. About a decade ago, Google characterized the workloads in their data centers and discovered that the efficiencies provided by custom hardware for just 10% of their common workloads would generate a positive return on the significant (>$100M) investment required. The Google Tensor Processor Unit is now in its third generation, and other information processing service companies like Amazon, Baidu, and Alibaba now all have custom hardware to create strategic differentiation and efficiency in their service offerings. The US national high-performance computing industry is no longer the premier segment of computational science; that center has shifted towards China.

The demands for scientific discovery and engineering optimization require a similar approach as the commercial companies have set out for artificial intelligence in the form of efficient high-performance computing solutions. However, due to the high cost of scientific computing and computational engineering infrastructure, this high-performance computing segment is progressively defined by multigroup collaborations that bring together a vast set of subject matter experts to solve next-generation problems. For more than a decade it has been known that IEEE floating point is making these collaborations costly and ineffective because collaborating groups cannot reproduce results due to a shortcoming in IEEE floating point stemming from its numerical rounding rules. The *posit* number system was designed to solve this problem by defining rigorous numerical rounding rules and a flexible mechanism to defer any rounding during long, computational sequences.

*What are you trying to do? Articulate your objectives using absolutely no jargon.*

Our goal is to transition the entire high-performance computing industry to the posit number system so that computational scientific and engineering experiments can be delivered with reproducibility, optimally fast execution speed, less resource requirements (esp. less memory, bandwidth, energy, etc.), efficiency, accuracy, correctness, and cost-effectiveness.

*How is it done today, and what are the limits of current practice?*

Today, collaborative efforts that depend on reproducibility of computational results depend on arbitrary precision software emulation techniques, which tend to be four to five orders of magnitude slower than IEEE floating point hardware, significantly reducing their usefulness and applicability.

*How is it done today and what are the limits of current practice?*

The proliferation of high-performance computing into real-time and embedded use cases amplifies a major shortcoming of the standard floating-point number system: floating-point addition and multiplication are not associative.

High-performance task-level parallel systems introduce different execution orders of the original equations causing nondeterministic reordering of intermediate results. When such systems are inspected, the non-deterministic reordering makes reproduction of the failure difficult if not impossible. Rounding errors are then indistinguishable from logic errors in the programming of the parallel execution.

Several solutions to the non-associativity of floats were proposed as early as 1986, starting with the work of Ulrich Kulisch[17]. Kulisch leveraged a 'super-accumulator' that accumulates intermediate results of a computational path at full precision. The actual rounding decision is made explicit by language constructs under control of the programmer. The fundamental problem with the Kulisch approach is caused by the structure of floating point: a fixed-point representation of the result of a floating point multiply requires $1 + 2 \times (2ebits + mbits)$, where $ebits$ and $mbits$ are the number of bits in the exponent and mantissa respectively. To be able to accumulate $2k$ products we would add $k$ bits to the accumulator. For single precision and double precision floats the approximate size of these superaccumulators would be 640 bits, and 4288 bits respectively. Modern implementations of the Kulisch accumulator idea can be found in "A Hardware Accelerator for Computing an Exact Dot Product"[18].

**Commented [JG7]:** Note that this is now consistent with the previous statement of the size, which used 4,224 bits. 4,288 bits is the correct number.

Another approach is to use arbitrary precision arithmetic. An example is the GNU Multiple Precision Floating-Point Reliably (MPFR[19]). The upside is that very difficult computation problems in computational geometry and optimization become feasible, but the downside is that the common case is slowed down by *three orders of magnitude*. ExBLAS[20] is a software approach that isn't as slow as arbitrary precision but is still at least an order of magnitude slower than native execution. ExBLAS uses the super-accumulator approach coupled with a clever trick to compute the rounding error of each operation. By keeping track of how error accumulates in the basic linear algebra subroutines they can create reproducible results. RepoBLAS[21] instead focuses on performance and relaxes the exactness constraint to deliver reproducible results in task-parallel execution environments. Floating-point based arithmetic error control is complicated by the structure of the number systems. Super-accumulators grow very large due to the disproportional dynamic range of floats compared to their precision; also, the IEEE rules lead to the super-accumulator being slightly larger than an integer power of 2 bits in size, creating hardware inefficiency.

*What is new in your approach and why do you think it will be successful?*

Posits have demonstrated that they solve the reproducibility problem cleanly, consistently, and using fewer resources than methods based on IEEE floating point. The use of the quire is the key component that makes it possible to deliver deferred rounding, and the size of the quire for posits tends to be an order of magnitude smaller than for IEEE double precision floating point. Any posit hardware that implements the quire to deliver reproducible computational science is expected to be an order of magnitude more efficient as compared to IEEE floating point hardware.

*Who cares? If you are successful, what difference will it make?*

The computational science and engineering communities that depend on collaboration to progress the state-of-the-art need high-performance reproducible computing. Posit-based supercomputers will be able to make these collaborations more effective, enabling subject matter experts to focus on their domain without having to deal with the numerical errors of IEEE floating point. This will improve the productivity of these teams as well as the quality of their work. Research teams will be able to reproduce their computational results regardless of the size of their computing infrastructure.

*What are the risks?*

The risk in moving the high-performance computing hardware industry from IEEE floating point to posit-based number systems is in the transformation of the numerical software. All current scientific and engineering software uses IEEE floating point, and any changes to the underlying number system will require new software validation efforts. History has shown that new computing hardware requires several generations before the software catches up. Case in point has been the shift to use GPUs in scientific computing. The first research papers that demonstrated the benefit of using GPUs where published in 2000, but the first general purpose 64-bit hardware accelerator that generalized this for scientific computing was introduced in 2005 (the ClearSpeed CSX600), followed by 64-bit accelerators from traditional graphics chip vendors by 2007 (the NVIDIA Tesla GPU). It took another five years for enough computational libraries to be developed and validated before general, non-specialized computational research teams started to use these hardware accelerators. The resulting artificial intelligence revolution unfolded over the next five years, as early adopters of GPU hardware tapped into several orders of magnitude better performance as compared to general purpose CPUs. We address this risk by including in the proposed program several powerful demonstrations of reproducibility with posits.

An example mid-term software test might apply one of Gustafson's most recent technical breakthroughs, the Minefield Method, which allows perfectly-rounded elementary functions to be computed at uniformly high speed (a solution to the classic Table-Maker's Dilemma). While the approach has been proved to work, it now needs to be applied to each of the dozens of functions people expect to find in a standard library.

*How much will it cost?*

Posit-enabling a 1000 server cluster using FPGA-based solutions is roughly $10M, approximately cost-equivalent to GPU-enabling that same cluster. This FPGA-based solution will advantageously enable continuous engineering modifications as the posit-based solution is fine-tuned, followed by delivery of a robust posit-based solution in three generations, while at the same time delivering immediate value to the industry. Once the software transformation is complete, an ASIC implementation to deliver yet another order of magnitude performance benefit would be recommended.

While the cost to build a state-of-the-art ASIC for computational science ranges from $25M to $100M depending on the manufacturing process selected, the FPGA ecosystem enables state-of-the-art process nodes (10-7 nm) to be used for posits, while amortizing the manufacturing cost across many different applications. While FPGA hardware is typically about an order of magnitude less efficient as compared to a custom ASIC, the overall system cost for a prototypical FPGA-based high-performance computing installation will be substantially more cost-competitive. As indicated before, the risk in moving an industry to a new number system is in transforming the software, and thus being cost effective is more important than being performance efficient in order to accelerate the software transformation.

*How long will it take?*

The first FPGA-based solutions are already in prototype stage at several universities, start-ups, and large incumbents outside the HPC industry (Robert Bosch, Facebook, and Huawei). It would therefore require less than a quarter for production systems to be available capable to engineer software transformations to take advantage of the posit number system. The goal would be to have two or three computer-aided engineering applications (G+SMO, TRADES) for additive manufacturing to be available delivering reproducible results on medium-sized clusters, by the end of 2019. Each year thereafter, we would expect an exponential growth of available applications that are transformed to use posit-based reproducible computational methods.

*What are suggested mid-term and final “exams” to check for success?*

The program can be staged for testing first with a single FPGA or small cluster before proceeding to build-out larger clusters capable of accurately running design optimizations of more complex systems covering larger spans of multi-physics scales.

Mid-term and final “exams” to check for success focused on reproducibility could include the application of the FPGA hardware to a carefully-chosen series of software tests. In a recent technical breakthrough of Gustafson’s, the Minefield Method, perfectly-rounded elementary functions can be computed at uniformly high speed (a solution to the classic Table-Maker's Dilemma). While the approach has been proved to work, it now needs to be applied to each of the dozens of functions people expect to find in a standard library.

FPGA-based implementations of posits leverage existing FPGA hardware to achieve the promised performance per watt improvements, without having to first invest in the development of new chips.

**Commented [JG8]:** While “W” is capitalized as an abbreviation for a watt, the word “watt” is not unless it is someone’s surname. Similarly with volt, ampere, tesla, henry, etc.

### 5.3.2. FPGA-based posit-enabled tensor accelerator (PETA) for DoD IoT

While FPGA-based posit-enabled tensor accelerators (FPGA-PETA) can beneficially be developed for HPC applications as discussed above, an application having substantially more sensitivity to performance per watt, size, and cost is IoT.

*Who cares?*

While commercial providers will develop proprietary IoT sensor networks for international use in revenue-generating applications, DoD will want customized incorporation of posits into high-availability IoT applications able to operate at lower power levels and with less energy than that available to enemies leveraging anticipated commercial systems. Example commercial and DoD missions include video encode-decode in mobile devices; sensor fusion in autonomous vehicles, robotic, and embedded industrial systems; and the smart sensors and analytic gateways used in both the IoT, and the Industrial Internet of Things (IIoT).

*If you are successful, what difference will it make?*

Posit-enabled IoT systems will operate at lower power and consume less energy. This will improve the availability of sensor data while extending the reach of the network, with lower-weight batteries and smaller form factors than would be possible without posits.

*What's new in your approach and why do you think it will be successful?*

Gustafson's work on tapered number systems[1] regains control over efficient and productive error control. He coined the term universal numbers, or *unums* for short. Unums come in several types, the Type III unums, a.k.a. posits[5], are the basis of Stillwater Computing's adaptive tensor processing architecture.

ERROR-FREE LINEAR ALGEBRA: Posits offer higher precision than floats at the same size due to tapering. This allocates fraction bits where a typical computation needs them most: numbers of modest magnitude. However, the improved accuracy of posits does not provide error-free linear algebra in the same way that simply going to the next bigger float doesn't resolve numerical issues in an algorithm. The addition of a quire to the posit standard enables rounding control for arbitrary computational paths and graphs. The quire is equivalent to the super-accumulator of Kulisch. In 2008, the IEEE 754 standard[22] added a fused multiply-add. Fusing is defined as deferring the rounding of a computational path until the last assignment operation.

For example, a floating-point fused multiply-add, or FMA, takes the exact result of a multiplication and uses it as input to the addition, avoiding a rounding step. This leads to less rounding error than the rounded multiplication followed by the addition. However, it increases irreproducibility due to variability of compilers and hardware. Some processors do not have a fused multiply-add instruction, some processors use higher internal precision during computation, and different compilers may emit different instruction sequences. To gain control over rounding error, the rounding decision must be programmer-visible.

In the posit standard the following fused instructions can (optionally) be performed using the quire, or supported with separate hardware:

| | |
|---|---|
| Fused multiply-add | $(a \times b) + c$ |
| Fused add-multiply | $(a + b) \times c$ |
| Fused multiply-multiply-subtract | $(a \times b) + (c \times d)$ |
| Fused sum | $\sum a_i$ |
| Fused dot-product | $\sum a_i b_i$ |

The usage model for the quire is to be the intermediate accumulator for a programmer-defined computational path. In code form:

```
posit<16,1> a,b,c, x;
quire<16,1> q;
q = a * b;
    … rest of the computational path
q += c;  // last step in the computation
x = q;  // final rounding step
```

This generalization of the fused instructions of the IEEE Floating Point standard is particularly valuable for the construction of math libraries. The fused dot-product is the key innovation in our posit-enabled adaptive tensor processor; it is elevated to an atomic instruction with its own stream control.

POSIT QUIRES AND QUIRE SIZE

The size of a posit quire is a function of the number of bits in the posit and the size of the exponent field. In mathematical terms, the smallest magnitude nonzero value that can arise after a multiply is $minpos^2$. Every other product is an integer multiple. The largest possible product value is $maxpos^2$, and thus a fixed-point integer big enough to hold these extremes is: $maxpos^2 / minpos^2$. As with the discussion of the super-accumulator in the introduction, we need to accumulate some non-trivial number of these products, and for $k$ additional bits you would be able to accumulate $2^k$ products. A value of $k = 30$ would accommodate roughly a billion products. For custom hardware we can dial this number to particular workload demands, but for the Draft Posit Standard, the quire size in bits is half the square of the posit size (32 bits for 8-bit posits, 128 bits for 16-bit posits, and 512 bits for 32-bit posits). This is large enough to accommodate the dynamic range needed and also supply additional bits to protect against overflow.

As a rule of thumb, when leveraging the quire, a posit has roughly the same precision as a float twice its size, so 32-bit posits compete with 64-bit floats, and 16-bit posits compete with 32-bit floats. No use case has yet been shown to require 64-bit posits, probably because the quire for 32-bit posits is already capable of 150-decimal precision.

GENERAL PURPOSE QUIRES: To enable quires for explicit rounding control they need to be visible to the programmer. For traditional load-store instruction set architectures (ISA) this implies that we need to create a special quire namespace and connect that namespace to the individual instructions that need to participate. Modern microprocessors can move 128 bits per clock cycle to or from DRAM, so even moving a block of quires can be done at four clock cycles per quire (about 60 GB/sec), though this bandwidth is shared by multiple cores. To sustain these throughputs, all high-performance linear algebra algorithms use some form of blocking to prime the caches with the next set of operands. Depending on the DRAM parameters and the cache line size, a typical optimized data flow would bring in 2D blocks of 8–16 lines. If we design a quire register set and ISA for fused dot products, to support the blocking of memory, we would need to create a register file of at least 8 and preferably more addressable quire registers.

### *How is hardware acceleration done currently without posits?*

Google's Tensor Processing Unit™ [16] is a notable example of the benefits of custom hardware acceleration for tensor operators. They focused on the dense matrix/vector operations present in neural network training and inference workloads. The data flow that this class of tensor operators need is centered about the weight matrices.

### *What are the limits of current practice?*

Without posits and quires, IoT will continue to require more bits (and more energy) than can be practically supported in distributed, high-availability IoT systems.

### *What are you trying to do? Articulate your objectives using absolutely no jargon.*

We want to build posit-enabled hardware for high-availability IoT applications in which battery weight and high availability are driving metrics.

### *What is new in your approach and why do you think it will be successful?*

As we have seen in the previous section, special handling of the quire in conjunction with blocked data movement to and from memory via the fused dot product assures error-free linear algebra. The corresponding hardware accelerator architecture for error-free posit-based linear algebra is illustrated in Figure 3.

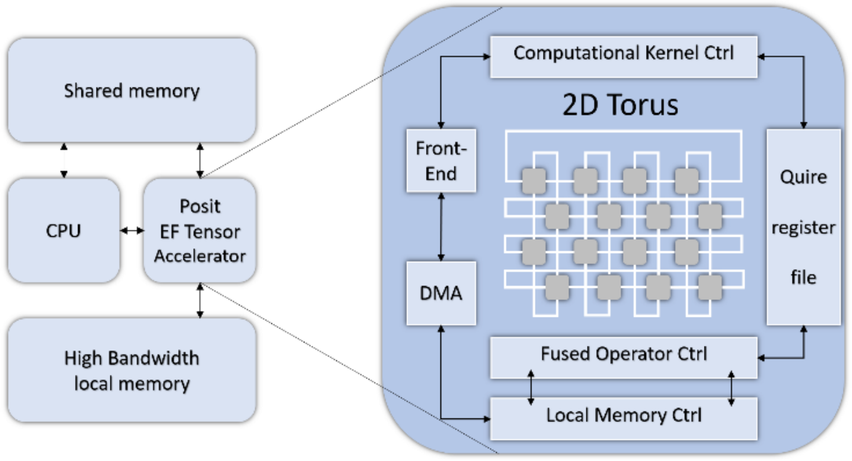


*Figure 3: Error-free Tensor Accelerator*

The error-free tensor processor is built around a scalable torus fabric of posit arithmetic units. The Front-end and DMA (Direct Memory Access) engines orchestrate tensor data flow movement to support fused dot-product streams. They use a block-oriented memory access pattern to improve DRAM efficiency and access latency. As blocks arrive at the fused-operator controller, the contained fused operator operand sets identify the respective quires that need to receive the intermediate results. Under the direction of a computational kernel, the DMA, fused operator controller, and quire register file coordinate a specific stream schedule that leaves the quires stationary until they are assigned to their final destination, at which point they are evicted from the quire register file. This avoids having to move quires to and from memory: instead, we move dot-product streams past the quires.

## ERROR-FREE DISTRIBUTED MEMORY

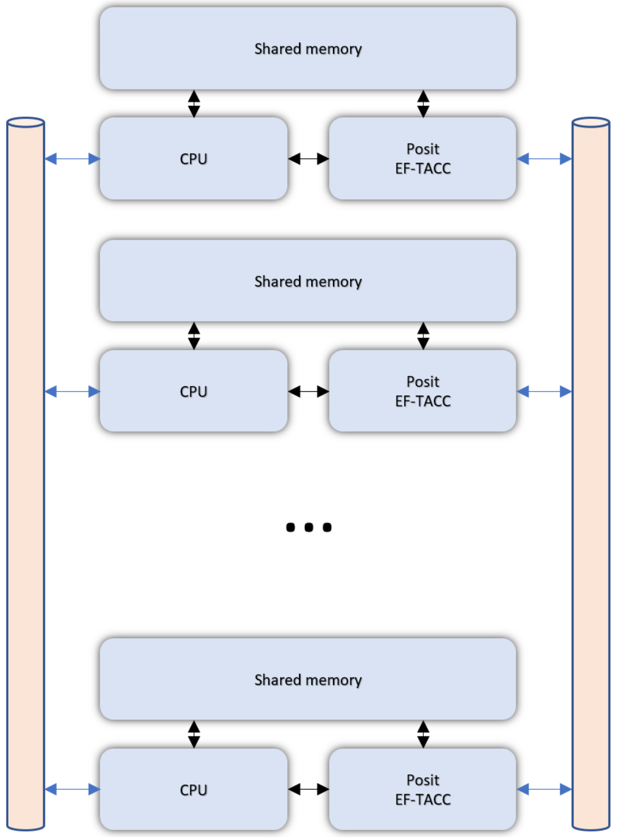


*Figure 4: Distributed memory error-free linear algebra cluster*

The error-free posit tensor accelerator was designed to scale up and down to accommodate large-scale HPC applications and low-latency real-time embedded applications. This is accomplished by a traditional distributed memory design with an additional fine-grain data flow accelerator network. The scalable cluster design is shown in Figure 4. We leverage a fast serial processor to address workloads that have a significant serial component and are governed by Amdahl's law (strong scaling). The design couples that with a scalable tensor hardware accelerator to address workloads that are governed by Gustafson's law (weak scaling). The CPU side is connected via an IP network to support coarse grain distributed memory operations to support very large in-memory data structures and global transformations such as global addressing mods and reductions. On the tensor accelerator side, computation is organized in fine-grain data flow to support the quire accumulation. The accelerators are also connected by a network, but instead of an IP network, this network is a mesh network to support fine-grain data flow computations and transformations such as sorts and transposes.

### 5.3.3. Posit-Enabled ASIC

The plan here is to leverage lessons-learned from a posit-capable ASIC developed by REX Computing to build a posit-enabled ASIC particularly well-suited to application in RF and microwave digital signal processing applications starting with DoD communications and radar systems, before proceeding on a commercialization path which will leverage rapid market adoption in 5G applications to lower costs for DoD moving forward.

*What are you trying to do? Articulate your objectives using absolutely no jargon.*

Etaphase aims to create a posit-enabled, ASIC-based accelerator.

*How is it done today, and what are the limits of current practice?*

Currently, GPUs are optimized for floating point. This current practice is limited by the poor accuracy and inefficiencies of floating point. This current practice wastefully requires twice as many memory bits, causing memory-bandwidth-limited calculations to slow down and increasing the energy necessary to communicate with memory.

*What is new in your approach and why do you think it will be successful?*

Posit-enabled ASICs are new. We think it will be successful as Gustafson has a design the basic functionality of which has been proven in its first chip iteration.

*Who cares? If you are successful, what difference will it make?*

Etaphase has identified a first potential commercial customer in a 5G network application. This application would leverage the posit-enabled ASIC hardware against a posit-enabled FFT. This application has enormous potential to make a difference in DoD wireless communications systems as well as in DoD phased array radar systems. And posit-enabled design automation promises to substantially streamline creation of custom ASICs for wide-ranging DoD applications beyond communications and radar, to include Artificial Intelligence and HPC.

*What are the risks?*

It is always hard to know how many design iterations might be required before succeeding in the fabrication of a high-functioning chip. This risk has been substantially mitigated by the successful test of a first design/fab/test for basic functionality.

*How much will it cost?*

ASIC Development efforts typically range between $25 million and $100 million. With the right team and with the design that we already have in-hand, we expect that our first prototype can be built and tested for $25 million.

*How long will it take?*

We propose a four-year program with an optional fifth year for final lab testing in a chosen DoD application.

## Bibliography

[1] J. L. Gustafson, *The End of Error: Unum Computing*, no. November. 2015.

[2] J. L. Gustafson, “Type III unum invention report.”

[3] J. L. Gustafson, (Etaphase), and I. T. (Etaphase) Yonemoto, “Unum Design (uDESIGN) Seedling Kick-Off,” in *DARPA TRADES Kick-Off Meeting*.

[4] J. L. Gustafson and I. T. (Etaphase) Yonemoto, “Beyond Floating Point: Next-Generation Computer Arithmetic,” *Stanford EE Computer Systems Colloquium 4:30 PM, Wednesday, February 1, 2017 NEC Auditorium, Gates Computer Science Building*. [Online]. Available: http://web.stanford.edu/class/ee380/Abstracts/170201.html.

[5] J. L. Gustafson and I. T. Yonemoto, “Beating Floating Point at its Own Game: Posit Arithmetic,” *Supercomput. Front. Innov. Vol 4, No 2*, 2017.

[6] Y. Uguen and F. De Dinechin, “Design-space exploration for the Kulisch accumulator,” *HAL Arch.*, no. <hal-01488916v2>, 2017.

[7] Intel, “Intel Advanced Vector Extensions 512 (Intel AVX-512).” [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/avx-512-overview.html. [Accessed: 27-Dec-2018].

[8] J. L. Gustafson, “Posit Arithmetic for LINPACK-Type Solvers,” 2018.

[9] J. L. Gustafson, “Inverting Hilbert Matrices with Posit Arithmetic,” 2018.

[10] P. Lindstrom, S. Lloyd, and J. Hittinger, “Universal Coding of the Reals : Alternatives to IEEE Floating Point,” 2018.

[11] R. Chaurasiya, J. Gustafson, R. Shrestha, J. Neudorfer, S. Nambiar, and K. Niyogi, “Parameterized Posit Arithmetic Hardware Generator,” in *IEEE International Conference on Computer Design (ICCD)*, 2018, vol. 6B182.

[12] J. Johnson, “Rethinking floating point for deep learning,” *arXiv*, vol. 1811 . 017.

[13] J. L. Gustafson and et al, “Reversible FFTs Using Posit Arithmetic,” 2018.

[14] L. M. R. Mullin, “A Mathematics of Arrays December 1988,” Syracuse University, 1988.

[15] A. Acar, A. Anandkumar, L. Mullin, S. Rusitschka, and V. Tresp, “Dagstuhl Reports: Tensor Computing for Internet of Things,” 2016.

[16] N. P. Jouppi, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, C. Young, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, N. Patil, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Patterson, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, G. Agrawal, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, R. Bajwa, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, S. Bates, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W.

Wang, E. Wilcox, D. H. Yoon, S. Bhatia, and N. Boden, “In-Datacenter Performance Analysis of a Tensor Processing Unit,” *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 1–12, 2017.

[17] U. W. Kulisch and W.L. Miranker, “The arithmetic of the digital computer: a new approach,” *SIAM Rev.*, vol. 28, no. 1, 1986.

[18] J. Koenig, D. Biancolin, J. Bachrach, and K. Asanovic, “A Hardware Accelerator for Computing an Exact Dot Product,” *IEEE 24th Symp. Comput. Arith.*, pp. 114–121, 2017.

[19] L. Fousse, G. Hanrot, V. Lefevre, P. Pelissier, and P. Zimmermann, “MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding,” *ACM Trans. Math. Softw.*, vol. 33, no. 2, 2007.

[20] R. Iakymchuk, S. Collange, D. Defour, and S. Graillat, “ExBLAS: Reproducible and Accurate BLAS Library,” 2015.

[21] P. Ahrens, H. D. Nguyen, and J. Demmel, “Efficient reproducible floating point summation and BLAS,” 2015.

[22] “IEEE Standard for Binary Floating-Point Arithmetic,” 1985.

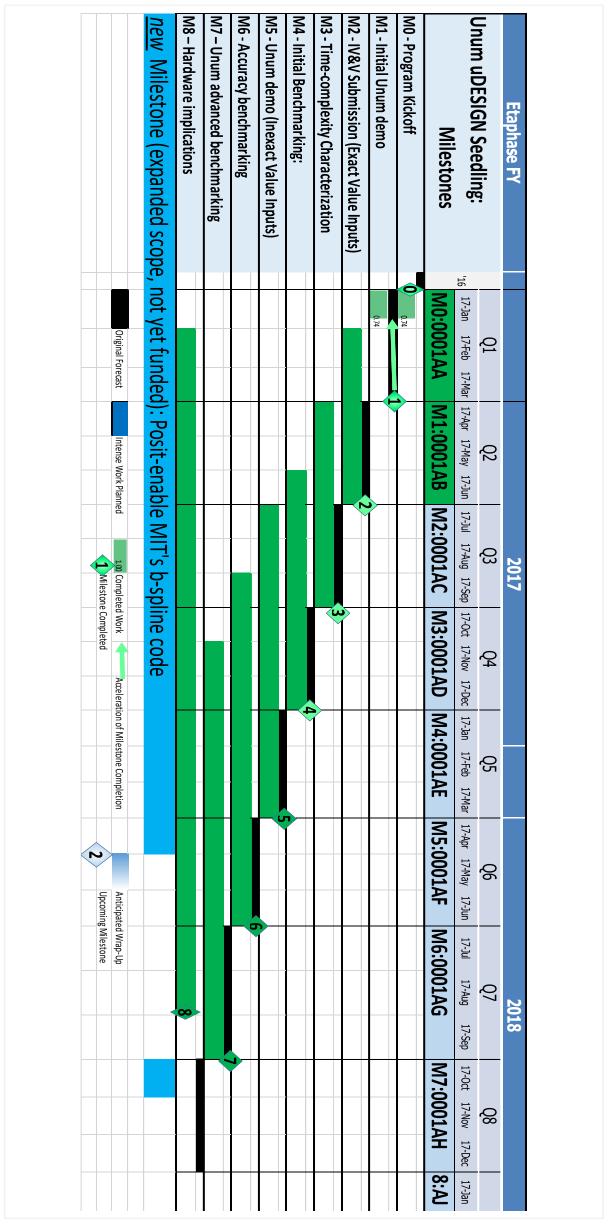
## Appendix A: Survey of Posit Hardware and Software Development Efforts (June 2018, updates online)

| Project | Type | Precisions | Quire Support? | Speed | Testing | Notes |
|---|---|---|---|---|---|---|
| Calligo Tech “Posit Numeric Unit (PNU)” (Vijay Holimath) | FPGA; first working posit hardware | 32 | Claimed, not yet tested | ~0.5 Mpops/s | Extensive tests, not exhaustive. No known bugs. | Single-op accelerator approach; allows direct execution of C codes written for floats. + – × tested; ÷ √ claimed |
| A*STAR “posit4.nb” (John Gustafson) | *Mathematica* notebook | All | Yes | < 80 kpops/s | Exhaustive for low precisions. No known bugs. | Original definition and prototype. Most complete environment for comparing IEEE floats and posits. Open source (MIT license). Many examples of use, including linear solvers |
| Stillwater Technologies (Theo Omtzigt) | C++ template library | All | Yes | ~2 Mpops/s | Extensive testing, but some bugs in the “correct answer” source | Open source. Ports in progress to TensorFlow, linear algebra libraries, other applications |
| A*STAR “SoftPosit” (Cerlane Leong) | C library based on Berkeley SoftFloat | 8, 16, published and complete; 32 needs √ and integer conversions | Yes | ~60 to 110 Mpops/s on x86 core (Broadwell) | 8: Exhaustive; 16: Exhaustive except FMA ,quire 32: extensive subset tests. No known bugs. | Fastest and most comprehensive C library for posits presently. Open source (MIT license) Designed for plug-in comparison of IEEE floats and posits. |
| IBM-TACC (Jianyu Chen) | Specific-purpose FPGA | 32 | Yes | 16–64 Gpops/s | Only one case tested, need major software modification to multiply different numbers | Does 128-by-128 matrix-matrix multiplication (SGEMM) using quire. Dense matrix-matrix multiplier. Task-specific and data-specific design. |

| Project | Type | Precisions | Quire Support? | Speed | Testing | Notes |
|---|---|---|---|---|---|---|
| "Sigmoid Numbers"(Isaac Yonemoto)** | Julia library | All <32, all ES | Yes | Unknown | No known bugs (posits), Division bug (valids) | Leverages julia's templated mathematics standard library, can natively do matrix and tensor operations, complex numbers, FFT, DiffEQ. Support for valids. |
| "FastSigmoid" (Isaac Yonemoto) | Julia and C/C++ library | 8, 16, 32, all ES | No | Unknown | Known bug in 32-bit multiplication | Used by LLNL in shock fluid dynamics case studies. |
| untitled Verilog effort (Isaac Yonemoto) | Julia and Verilog | 8,16,32, ES=0 | No | Unknown | Comprehensively tested for 8-bit, no known bugs | Intended for Deep Learning applications Addition, Subtraction and Multiplication only. A proof of concept matrix multiplier has been built, but is off-spec in its precision. |
| Secret Effort Social Network | VLSI | 5? | Unknown | Unknown | Unknown | Proprietary in-house design for Deep Learning workloads |
| Bosch (Rohit Buddhiram Chaurasiya) | Verilog generator for VLSI, FPGA | All | No | Similar to floats of same bit size | Limited testing; no known bugs | May soon be used in commercial products to reduce bit size needed |
| Speedgo (Chung Shin Yee) | Python library | All | No | ~20 Mpops/s | Extensive; no known bugs | Open source (MIT license) |
| IIT-Madras "SHAKTI" (V. Kamakoti, Sugandha Tiwari) | RISC-V processor with posit ALU | Unknown | Unknown | Unknown | Unknown | Difficult to engage via email |
| Manish Kumar Jaiswal | HDL posit generator | Unknown | No | NA | Unknown | Incorrect (truncated) rounding for now; plans to correct |
| "PySigmoid" Ken Mercado | Python library | All | Yes | < 20 Mpops/s | Unknown | Open source (MIT license). Easy-to-use interface. Neural net example. Comprehensive functions support. |

| Project | Type | Precisions | Quire Support? | Speed | Testing | Notes |
|---|---|---|---|---|---|---|
| Lombiq Technologies (Zoltán Lehóczky and team) | C# with Hastlayer for hardware generation | 32 initially, adjustable with effort | Yes | Not fast | Partial | Requires Microsoft compiler |
| Jorge Aparicio | Rust library | 8 (*es* = 1) | No | Unknown | Unknown, but probably exhaustive | Very small subset of posit environment |
| Emanuele Ruffaldi | C++ library | 4 to 64 (any *es* value); “Template version is 2 to 63 bits” | No | Unknown | A few basic tests | 4 levels of operations working with posits. Special support for NaN types (nonstandard) |
| “bfp (Beyond Floating Point)” Clément Guérin | C++ library | Any | No | Unknown | Bugs found; status of fixes unknown | Supports + – × ÷ √ reciprocal, negate, compare |
| A*STAR “posithub.org” (Cerlane Leong) | JavaScript widget | Convert decimal to posit 6, 8, 16, 32; generate tables 2–17 with es 1–4. | NA | NA; interactive widget | Fully tested | Table generator and conversion |
| Tokyo Tech (Artur Podobas) | FPGA | 16, 32, extendable | No | “2 GHz”, not translated to Mpops/s | Partial; known rounding bugs | |
| REX Computing (Thomas Sohmers) | FPGA version of the “Neo” VLIW processor with posit numeric unit | 32 | No | ~1.2 Gpops/s | Extensive; no known bugs | No divide or square root. First full processor design to replace floats with posits. |

## Appendix B: Etaphase uDESIGN Seedling Timeline and Milestone Chart



| SUBCLIN | Milestone List and Chart | ACRN | MONTH |
|---|---|---|---|
| 0001AA | MS 0: Program Kickoff. | AA | 0 |
| 0001AB | MS 1: Initial Unum demo. A working implementation of a generalized Ubox method for Type 1 Unums for data on N-dimensions subject to a constraint.<br>Deliverable: Reporting/Software/Demonstrator. | AA | 3 |
| 0001AC | MS 2: IV&V Submission (Exact Value Inputs): A working implementation of linear equation solving using Type 1 Unums and the Ubox method on exact value inputs using simple test data. Submission of software to IV&V team.<br>Deliverable: Reporting/Software/Demonstrator. | AA | 6 |
| 0001AD | MS 3: Time-complexity Characterization: Characterization of time-complexity of the Ubox search method for higher-order systems of linear equations.<br>Deliverable: Reporting/Software/Demonstrator. | AA | 9 |
| 0001AE | MS 4: Initial Benchmarking:<br>o Demonstrate results from Level 2 BLAS linear equation benchmark.<br>o Demonstrate results from LINPACK 100 & 1000 linear equation benchmarks<br>o Submission of updated software to IV&V team.<br>Deliverable: Reporting. | AA | 12 |
| 0001AF | Characterization of equation solutions on inexact value inputs with guidance and parameters helpful for solving results from inexact value calculations.<br>Deliverable: Reporting/Software/Deliverable. | AA | 15 |
| 0001AG | MS 6: Accuracy benchmarking: Accuracy benchmarking on Unum equation solving and identification of "trouble regions" where Unum-based equation solving yields accurate results over traditional IEEE floating point calculations. Compilation of problem inputs and submission to the IV&V team.<br>Deliverable: Reporting/Software/Deliverable. | AA | 18 |
| 0001AH | MS 7: Unum advanced benchmarking: Revision of unum library to perform calculations with zero heap-fetching operations in memory. Comparison with standard BigFloat and performance benchmarking against Level 2 BLAS and LINPACK 100/1000 datasets in both Gaussian elimination only and UBOX-enhanced modes. Submission of updated software to IV&V team.<br>Deliverable: Reporting/Software. | AA | 21 |
| 0001AJ | MS 8: Evaluate implications of hardware architecture on unum performance: Revision of unum library to optionally track actual and "virtual" bit-wise execution costs and memory movement latencies. Identification of key bottlenecks in BLAS/LINPACK where hardware architectural design can most effectively yield valuable performance enhancements. Submission of updated software to IV&V team.<br>Deliverable: Reporting/Software. | AA | 24 |

# Draft Standard for Posit Arithmetic

(Updated 1 January 2019)

Sponsor

**Agency for Science, Technology and Research (A*STAR)**

**Abstract:** This standard specifies the storage format, operation behavior, and required mathematical functions for posit arithmetic in computing environments. It describes the binary storage used by the computer and the human-readable character input and output for posit representation. They may be realized in software or hardware or any combination of the two. A system that meets this standard is said to be *posit compliant* and will produce results that are identical to any other posit compliant system.

**Keywords:** Arithmetic, binary, exponent, format, fraction, NaR, rounding, posit, quire, regime, unum

**Participants**

The following people in the Posit Working Group contributed to the development of this standard:

**John Gustafson**, *Chair*

| | | |
|---|---|---|
| Gerd Bohlender | Siew Hoon Leong (Cerlane) | Theodore Omtzigt |
| Shin Yee Chung | Geoff Jones | Andrew Shewmaker |
| Vassil Dimitrov | Peter Lindstrom | Isaac Yonemoto |

## Table of Contents

# 1 Overview

## 1.1 Scope

This standard specifies the storage format and mathematical behavior of posit numbers, and the functions a posit arithmetic system must support. It includes a description of how real-valued results are to be rounded and how exceptions are handled.

## 1.2 Purpose

This standard provides a system for computing with real numbers represented in a computer using fixed-size binary values. Deviations from mathematical behavior (including loss of accuracy) are kept to a minimum while preserving the ability to represent a wide dynamic range of values. All features are accessible by programming languages; the source program and input data suffice to specify the output exactly on any computer system, similar to the way integer arithmetic produces bitwise-identical results.

## 1.3 Inclusions

This standard specifies:

- Formats for binary data, for computation and data interchange
- Addition, subtraction, multiplication, division, and comparison operations
- Combination (fused) operations that are correctly rounded
- Mathematical functions such as logarithm, exponential, trigonometric, and hyperbolic functions
- Conversions between different posit formats
- Conversions of other number representations to and from posit format
- Exception handling when a result is not a real number (NaR).

## 1.4 Requirements vs. Recommendations

All descriptions herein are requirements of the behavior of the system. The decision of how to satisfy the requirements (using any combination of hardware and software) is up to the implementer of this standard, but all functionality must be provided and behave as described for a system to be posit-compliant.

## 1.5 Programming Environment

A programming environment may claim to be compliant with this standard if it supports at least one of the four precisions (8, 16, 32, 64) completely. If it includes more than one precision, then it must also provide the ability to convert between those precisions.

## 2 Definitions, abbreviations, and acronyms

**Correct rounding:** This standard's method of converting an infinitely precise real value to a posit. A value so converted is said to be correctly rounded.

***es*:** Exponent size; the maximum number of bits 0, 1, 2, 3, … that are available for expressing the exponent.

**exponent:** The part of the power-of-two scaling determined by the exponent bits.

**exponent bits:** A field of bits within a posit that, in combination with the regime bits, determines the power-of-two scaling of the significand.

**format:** A set of bits and the definition of their meaning.

**fraction:** The component of a posit containing its significant binary digits after the binary point; $0 \leq$ fraction $< 1$.

**hidden bit:** an implicit 1 bit before the MSB of the fraction

**lg:** Logarithm base 2

**LSB:** Least Significant Bit

***maxpos*:** The largest real value expressible as a posit.

***minpos*:** The smallest positive real value expressible as a posit.

**MSB**: Most significant bit

**NaR:** Not a real. A value that has infinite magnitude, is indeterminate, is multi-valued, or requires an imaginary component to express (like $\sqrt{-1}$).

***ps*:** Posit size; the precision of a posit format, (8, 16, 32, or 64 bits).

***pintmax*:** The largest consecutive integer expressible as a posit.

**posit:** A real number that is exactly representable using a fixed number of bits in the format described in this standard, or a NaR.

**precision:** The number of bits available for expressing a quantity.

**quire:** A fixed-point format capable of storing sums of products of posits without rounding, up to some large number of such products.

**regime:** A subfield of a posit consisting of some number of identical bits terminated by the opposite bit or the end of the number, that contributes to the specification of the power-of-two scaling of the fraction.

**sign:** The value +1 for positive numbers, –1 for negative numbers. Exception values 0 and NaR are unsigned.

**sign bit:** The MSB of a posit or quire. For non-exception values, a 0 bit indicates a positive value and a 1 bit indicates a negative value.

**significand:** The implicit 1 bit followed by the fraction bits; $1 \leq$ significand $< 2$.

**universal number:** *See:* **unum**.

**unum:** universal numbers express real numbers (posits) and ranges of real numbers (valids).

**unum seed:** *See:* **useed**.

***useed*:** A value obtained by starting with 2 and squaring repeatedly *es* times: 2, 4, 16, 256, … that determines the way the projective real circle of unums gets populated.

# 3 Posit and quire formats

## 3.1 Overview

### 3.1.1 Formats

This clause defines posit formats, which are used to represent a finite set of real numbers. Posit formats are specified by their precision, *ps*. For each posit format, there is also a quire format of size $ps^2/2$ that is used to contain exact sums of products of posits. Dynamic range, accuracy, quire size, and format are determined solely by the precision.

There are four precisions described in this standard: 8, 16, 32, and 64. We sometimes refer to the four corresponding formats as posit8, posit16, posit32, and posit64.

### 3.1.2 Compliance

An implementation is compliant with this standard if it supports full functionality of at least one precision (8, 16, 32, or 64). If the implementation supports more than one precision, then it must support conversions between the precisions that it supports.

### 3.1.3 Represented data

Within each format, a posit represents either NaR, or a number of the form $m \times 2^n$, where $m$ and $n$ are integers limited to a range symmetrical about and including zero. The maximum $m$ range is $-2^p < m < 2^p$ where $p = ps - \lg(ps) + 1$ is the maximum number of significant digits (bits).

The smallest positive posit, *minpos*, is $2^{ps(2-ps)/8}$ and the largest positive posit, *maxpos*, is the reciprocal of *minpos*, or $2^{ps(ps-2)/8}$. Every posit is an integer multiple of *minpos*. Every real number maps to a unique posit representation; there are no redundant representations.

The quire represents either NaR or an integer multiple of $minpos^2$, represented as a 2's complement binary number with $2^{ps^2/2}$ bits. This enables it to add a list of posits or a list of exact products of posits without rounding error and thereby satisfy the associative and distributive laws of algebra up to some minimum length. Sums of lists longer than that minimum are capable of integer overflow.

Posits can exactly express all integers $i$ in a range $-pintmax \le i \le pintmax$; outside that range, integers exist that cannot be expressed as a posit without rounding to a different integer.

The values for the posit formats are summarized in Table 3.1.

**Table 3.1** Properties of posit formats

| property | posit8 | posit16 | posit32 | posit64 |
|---|---|---|---|---|
| Maximum significand size, bits | 6 | 13 | 28 | 59 |
| Maximum exponent bits, *es* | 0 | 1 | 2 | 3 |
| *minpos* | $2^{-6}$ $\approx 1.5\times10^{-2}$ | $2^{-28}$ $\approx 3.7\times10^{-9}$ | $2^{-120}$ $\approx 7.5\times10^{-37}$ | $2^{-496}$ $\approx 4.9\times10^{-150}$ |
| *maxpos* | $2^{6}$ $= 6.4\times10^{1}$ | $2^{28}$ $\approx 2.7\times10^{8}$ | $2^{120}$ $\approx 1.3\times10^{36}$ | $2^{496}$ $\approx 2.0\times10^{149}$ |
| *pintmax* | 8 | 256 | $2^{22}$ | $2^{52}$ |
| Quire size, bits | 32 | 128 | 512 | 2048 |
| Exact sum quire limit | 32767 | $2^{43}-1$ | $2^{151}-1$ | $2^{559}-1$ |
| Exact dot product quire limit | 127 | 32767 | $2^{31}-1$ | $2^{63}-1$ |

The exact sum quire limit and exact dot product quire limit are the number of additions or multiplication-additions up to which the quire cannot overflow. Up to these limits, the quire obeys the associative law of addition and the distributive law.

## 3.2 Binary interchange format encoding

### 3.2.1 Posit format encoding

All posits have just one encoding in a binary interchange format shown in Figure 3.1. The four fields are

a) Sign bit *S*
b) Regime *R* consisting of *r* bits identical to $R_0$, terminated by $1 - R_0$ ($r + 1$ bits total length) or the end of the posit (*r* bits total length).
c) Exponent *E* represented by *es* exponent bits; truncated bits have value 0.
d) Fraction *F* represented by *f* fraction bits; truncated bits have value 0

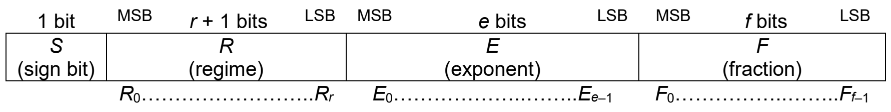


**Figure 3.1** General binary posit format

The meaning of each field is as follows:

a) *S* is its literal value, 0 or 1.
b) *R* is $-r$ if $R_0$ is 0, and $r - 1$ if $R_0$ is 1.
c) *E* is an *es*-bit unsigned integer, with 0 bit padding in the least significant bits if the exponent field has fewer than *es* bits because of the regime length.
d) *F* represents an unsigned integer divided by $2^f$.

NOTE—The regime size can be so long that all fraction bits are truncated. Even longer regimes can also truncate exponent bits. All truncated bits are treated as 0 bits.

The representation *(S, R, E, F)* of the posit and value *v* of the datum represented are inferred from the fields as follows:

a) If $S = 0$ and all other fields contain only 0 bits, then $v = 0$.
b) If $S = 1$ and all other fields contain only 0 bits, then *v* is NaR.
c) If any bits in the *(R, E, F)* are 1, then $v = (1 - 3S + F) \times 2^{(-1)^S(R\times 2^{es}+E+S)}$.

### 3.2.2 Quire format encoding

The quire is a fixed-point 2's complement value of length $ps^2/2$ which is 32, 128, 512, or 2048 bits for the posit sizes 8, 16, 32, and 64 respectively.

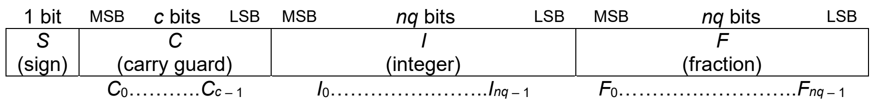

1 bit
MSB
*c* bits
LSB
MSB
*nq* bits
LSB
MSB
*nq* bits
LSB
*S*
(sign)
*C*
(carry guard)
*I*
(integer)
*F*
(fraction)
$C_0$...........$C_{c-1}$
$I_0$........................$I_{nq-1}$
$F_0$...........................$F_{nq-1}$

**Figure 3.2** Binary quire format

The number of bits for the fraction is $nq = \frac{1}{4}ps^2 - \frac{1}{2}ps$. The integer part also has *nq* bits. The carry guard has $c = ps - 1$ bits to guarantee that sums of products cannot overflow, up to $2^{ps-1} - 1$ products.

The representation *(S, C, I, F)* of the quire and value *v* of the datum represented are inferred from the fields as follows:

a) If $S = 1$ and all other fields contain only 0 bits, then *v* is NaR and undefined.
b) For all other cases, the value *v* is the 2's complement signed integer represented by all bits, divided by $2^{nq}$.

# 4 Rounding

## 4.1 Definition and Method

Rounding is the substitution of an expressible posit for any real number that is not expressible as a posit. Operation results are regarded as exact prior to rounding.

The *correct rounding* of a real value *x* is as follows:

a) If *x* is exactly expressible as a posit, it is unchanged.
b) If | *x* | > *maxpos*, *x* is rounded to sign(*x*) × *maxpos*.
c) If 0 < | *x* | < *minpos*, *x* is rounded to sign(*x*) × *minpos*.
d) For all other values, the value is rounded to the nearest binary value if the posit were encoded to infinite precision beyond the *ps* length; if two posit binaries are equally near, the one with binary encoding ending in 0 is selected.

### 4.2 Fused Operations

A fused operation is an expression for which only the final result is rounded; subexpressions are evaluated exactly. Fused operations are distinct from non-fused operations and must be explicitly requested in a posit programming environment.

Expressions that can be written in the form of a dot product can be performed using accumulations in a quire. Exact sums are dot products where one vector consists of all 1s. The fused multiply-add operation $(a \times b) + c$ is a dot product of vectors $(a, 1)$ and $(b, c)$. A complex product $(a + b\,i) \times (c + d\,i)$ can be performed as two fused operations, $a \times c - b \times d$ and $a \times d + b \times c$. A fused add-multiply $(a + b) \times c$ can be written as the dot product of $(a, b)$ and $(c, c)$ and performed with only a single rounding, via the quire.

Other fused operations are math library functions such as **rSqrt** and **hypot** (see below) that produce results that are correctly rounded for all input values.

# 5 Operations

### 5.1 Guiding principles for NaR

If NaR is the input to an operation that normally produces real number output, the result is also NaR. NaR is also output when the mathematical result is a discontinuity, or infinite in magnitude. A test of NaR equal to NaR returns True.

### 5.2 Mathematical functions

The following functions shall be supported, with correct rounding for all input arguments.

#### 5.2.1 Elementary functions of one argument

**negate**(*posit*) is identical to 2's complement integer negation of the binary representation.

**abs**(*posit*) is identical to 2's complement integer absolute value. Note that **abs**(NaR) is the only posit that can produce a result with a sign bit of 1, and **abs**(NaR) produces NaR.

**round**(*posit*) converts *posit* to the nearest posit with integer value, and the nearest even integer if two integers are equally far from *posit.*

**sign**(*posit*) returns 1 if the value of *posit* is positive, and –1 if the value of *posit* is 1. If *posit* is zero or NaR, sign(*posit*) returns 0.

#### 5.2.2 Elementary functions of two arguments

**addition**(*posit1, posit2*) returns *posit1* + *posit2*, rounded.

**subtraction**(*posit1, posit2*) returns *posit1* – *posit2*, rounded.

**multiplication**(*posit1, posit2*) returns *posit1* × *posit2*, rounded.

**division**(*posit1, posit2*) returns NaR if *posit2* = 0, else *posit1* / *posit2*, rounded.

### 5.2.3 Comparison functions of two arguments

All comparison functions are identical to the comparisons of the posit bit strings regarded as 2's complement integers, so there is no need for separate machine-level instructions. The value NaR has the bit string of the most negative integer, so NaR < *posit* returns True if posit is not NaR. The posit environment shall support:

**compareEqual**(*posit1*, *posit2*)
**compareNotEqual**(*posit1*, *posit2*)
**compareGreater**(*posit1*, *posit2*)
**compareGreaterEqual**(*posit1*, *posit2*)
**compareLess**(*posit1*, *posit2*)
**compareLessEqual**(*posit1*, *posit2*)

NOTE—Testing if a posit is NaR is not an exception: **compareEqual**(NaR, *posit*).

#### 5.2.3.1 Functions of one argument

**sqrt**(*posit*) returns NaR if *posit* < 0, else the square root of *posit*, rounded.
**rSqrt**(*posit*) returns NaR if *posit* ≤ 0, else 1/sqrt(*posit*), rounded.
**ceiling**(*posit*) returns the posit representing the greatest integer ≥ *posit*.
**floor**(*posit*) returns the posit representing the smallest integer ≤ *posit*.
**exp**(*posit*) returns $e^{posit}$, rounded.
**expm1**(*posit*) returns $e^{posit} - 1$, rounded.
**exp2**(*posit*) returns $2^{posit}$, rounded.
**exp2m1**(*posit*) returns $2^{posit} - 1$, rounded.
**exp10**(*posit*) returns $10^{posit}$, rounded.
**exp10m1**(*posit*) returns $10^{posit} - 1$, rounded.
**log**(*posit*) returns $\log_e$(*posit*), rounded. Returns NaR if *posit* ≤ 0.
**logp1**(*posit*) returns $\log_e$(*posit* + 1), rounded. Returns NaR if *posit* ≤ –1.
**log2**(*posit*) returns $\log_2$(*posit*), rounded. Returns NaR if *posit* ≤ 0.
**log2p1**(*posit*) returns $\log_2$(*posit* + 1), rounded. Returns NaR if *posit* ≤ –1.
**log10**(*posit*) returns $\log_{10}$(*posit*), rounded. Returns NaR if *posit* ≤ 0.
**log10p1**(*posit*) returns $\log_{10}$(*posit* + 1), rounded. Returns NaR if *posit* ≤ –1.
**sin**(*posit*) returns sin(*posit*), rounded.
**sinPi**(*posit*) returns sin(*π* × *posit*), rounded.
**cos**(*posit*) returns cos(*posit*), rounded.
**cosPi**(*posit*) returns cos(*π* × *posit*), rounded.
**tan**(*posit*) returns tan(*posit*), rounded.
**tanPi**(*posit*) returns tan(*π* × *posit*), rounded.
**asin**(*posit*) returns NaR if |*posit*| > 1, else arcsin(*posit*), rounded.
**asinPi**(*posit*) returns NaR if |*posit*| > 1, else arcsin(*posit*) / *π*, rounded.
**acos**(*posit*) returns NaR if |*posit*| > 1, else arccos(*posit*), rounded.
**acosPi**(*posit*) returns NaR if |*posit*| > 1, else arccos(*posit*) / *π*, rounded.
**atan**(*posit*) returns arctan(*posit*), rounded.
**atanPi**(*posit*) returns arctan(*posit*) / *π*, rounded.

**sinh**(*posit*) returns sin(*posit*), rounded.
**cosh**(*posit*) returns cos(*posit*), rounded.
**tanh**(*posit*) returns tan(*posit*), rounded.
**asinh**(*posit*) returns NaR if |*posit*| > 1, else arcsin(*posit*), rounded.
**acosh**(*posit*) returns NaR if |*posit*| > 1, else arccos(*posit*), rounded.
**atanh**(*posit*) returns arctan(*posit*), rounded.

##### *5.2.3.2 Functions of two posit arguments*

— **hypot**(*posit1*, *posit2*) returns the square root of $posit1^2 + posit2^2$, rounded.
**pow**(*posit1*, *posit2*) returns $posit1^{posit2}$, rounded. NaR is returned if the result is discontinuous in either argument, or has a nonzero imaginary component.
**atan2**(*posit1*, *posit2*)
**atan2Pi**(*posit1*, *posit2*)

### 5.2.4 Functions of a posit argument and an integer argument

— **compound**(*posit*, *n*) returns NaR if $x \leq -1$, else $(1 + x)^n$, rounded
**pown(***posit*, *n*) returns NaR if …,
**rootn**(*posit*, *n*)

# 6 Conversion operations for posit formats

## 6.1 Conversion between different posit formats

Conversion from a posit to a higher-precision posit is exact. Conversion to a lower-precision posit uses correct rounding.

## 6.2 Conversion between posit format and decimal character sequences.

Table 6.1 shows the minimum number of significant decimals needed to express a posit with decimal characters such that it will convert to the same posit using correct rounding.

**Table 6.1** Minimum decimals in significand to preserve posit value

| Posit Precision | **posit8** | **posit16** | **posit32** | **posit64** |
|---|---|---|---|---|
| Decimals | 3 | 5 | 10 | 20 |

## 6.3 Conversion between posit format and integer format.

The supported posit sizes must provide conversion to and from all supported integer sizes. Rounding for converting posits to integers is indicated by the operation name; If the integer is out of range or the posit is NaR, an exception is signaled. Conversion from integers to posits uses correct rounding.

## 6.4 Conversion between posit format and IEEE 754 float format.

When converting a posit to an IEEE 754 float of any type, the rounding is by the rounding mode of the float environment. Zero converts to “positive zero.” NaR converts to quiet NaN.

Conversion from floats to posits uses correct rounding for all floats that represent real values. All forms of infinity and NaN convert to NaR.