Unum Arithmetic: Better Math with Clearer Tradeoffs

Nic Brummel, John L. Gustafson, Andrew Klofas, Carlos Maltzahn, Andrew Shewmaker

Abstract—To raise performance beyond Moore's law scaling, *Approximate Computing* reduces arithmetic quality to increase operations per second or per joule. It works on only a few applications. The quality-speed tradeoff seems inescapable; however, *Unum Arithmetic* simultaneously raises arithmetic quality yet *reduces the number of bits required*. Unums extend IEEE floats (type 1) or provide custom number systems to maximize information per bit (type 2). Unums achieve Approximate Computing cost savings *without* sacrificing answer quality.

1 Introduction

Approximate Computing encompasses a range of ideas intended to reduce numerical quality in return for higher speed [1]. Such techniques include aggressive clipping of trailing fraction bits, alternative math libraries that compute to lower accuracy with fewer clock cycles, hardware that does not propagate a carry bit when adding, and so on. All of these ideas presume that better answers and higher computing costs go hand-in-hand.

In early 2015, the *unum* (universal number) data type was introduced as an extension of IEEE floats [2]. Unlike floats, it does not round, overflow to $\pm\infty$, or underflow to 0, but instead uses a single bit at the end of the fraction field (the *ubit*) to indicate if a number is an exact float or in the open interval between floats. The improved mathematical behavior allows unums to be safely used down to precisions as low as *four bits*, and they adjust in precision one bit at a time, up to thousands of bits. In some applications, unums provide tightly bounded answers using fewer bits than floats. In contrast with approximate computing, they offer a way to simultaneously improve answer quality while reducing the number of bits needed.

That form of unum is now referred to as type 1; the "unum type 2" completely breaks from IEEE floats to design a system for representing real numbers (and sets of real numbers) that maximizes information per bit with a Shannon-like maximum entropy approach [3]. A user (or a compiler) can customize the number system to fit the requirements of an application for accuracy and dynamic range, which allows unprecedented answer quality from a reduced number of bits.

- *N.B. is with the Applied Math & Statistics Department, University of California, Santa Cruz, CA 95064. E-mail: brummell@soe.ucsc.edu*
- *J.G. is with the A*STAR-CRC and NUS, 1 Fusionopolis Way #17-01 Connexis, Singapore 138632. E-mail: dcsjlg@nus.edu.sg.*
- *A.K. is with the Applied Math & Statistics Department, University of California, Santa Cruz CA 95064. Email: aklofas@ucsc.edu.*
- *C.M. is with the Computer Science Department, University of California, Santa Cruz, CA 95064. E-mail: carlosm@ucsc.edu.*
- *A.S. is with the HPC division, LANL, Los Alamos, NM 87544. E-mail: shewa@lanl.gov.*

2 Technology Description

Conventional floats have three bit fields: sign bit, exponent, and mantissa. We prefer the term "fraction" for mantissa. The IEEE Standard allows a choice between 16, 32, 64, 80, and 128 precisions, but for any precision, the Standard fixes the number of exponent and fraction bits, resulting in inefficiency for any particular application.

Type 1 unums allow adjustable exponent and fraction sizes at the bit level, all the way down to a single bit. Fig. 1 shows an example of the economization typical of a unum, in this case representing Avogadro's number.

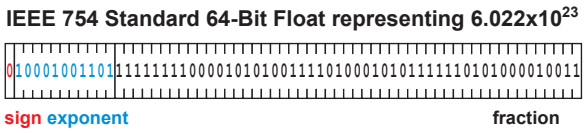

Fig. 1. IEEE Format versus Unum Type 1 Format

Unums offer the same trade-off versus floats as variable-width versus fixed-width typefaces: Harder for the design engineer and more logic for the computer, but superior for everyone else in terms of usability, compactness, and overall cost. Many examples are shown in [2]; here is a new example designed to stress destructive cancellation, a common source of accuracy loss in application programs. It demonstrates that economy need not reduce accuracy or dynamic range: With a budget of 32 bits per number, compute

$$x = \left(\frac{27/10 - e}{\pi - (\sqrt{2} + \sqrt{3})} \right)^{67/16}$$

starting from computer representations of the integers, π, and e. The correct value to seven places is 302.8827···.

IEEE 32-bit binary floats, with a dynamic range of over 2.8×10^{83} and a nominal precision of 7.2 decimal digits, produce the answer 302.9124···, correct to only three digits. In terms of Units in the Last Place (ULPs), the answer is off by 973 ULPs, even though the expression only requires 9 operations. The format provides no warning to the user regarding this loss of accuracy.

Approximate arithmetic, where (for example) we improve hardware speed by using only 29 bits instead of 32 and truncate numbers instead of rounding, produces the answer 303.0412···, which is wrong by about 0.297. Now only the first two digits are correct.

Interval arithmetic values $[a, b]$, where a and b are IEEE 16-bit floats and thus fit in the 32-bit budget, offer the hope of a rigorous bound on the answer, a sort of automatic error analysis. Each endpoint has only 3.3 decimal digits of precision and a dynamic range of 1.0×10^{12}, a considerable sacrifice of computing vocabulary for the benefit of a bound. It produces the very uninformative answer that x lies in the interval [18.21875, 33056.], a bound far too loose to have practical value.

Unum arithmetic (Type 1) has adjustable ranges; with three bits for the exponent field length and five for the fraction field length, unums have a precision of up to 9.9 decimals and a dynamic range of 2.4×10^{86}, quite a bit more than offered by 32-bit floats. The number of bits per unum varies from 12 to 49 as needed. If we cap the range at 36 bits maximum, which keeps the *average* size per unum below 32 bits, we get the result that x is provably inside (302.75, 303). The average unum size of about 29 bits saves a modest amount of storage and bandwidth but providing a reasonably accurate answer that also conveys invaluable information of an absolute *bound* on the result. Notice that the Approximate Arithmetic example above gives an answer that lies *outside* this rigorous bound.

There are many other examples for which unums provide bounds that are tighter than the inaccuracy of floats, using fewer bits. Unums are not restricted to applications that tolerate approximate computing techniques.

3 SC Challenges Addressed

A unum environment will simultaneously increase programmer productivity and make hardware more cost-effective by addressing the following challenges:

1. Mostly-automatic numerical error analysis
2. Higher performance on bandwidth-limited codes
3. Power/heat-dissipation reduction
4. Better accuracy with less storage requirements
5. Numerical instability made visible to user
6. Elimination of roundoff errors that are indistinguishable from coding errors
7. Bitwise identical results across platforms
8. Parallelization of codes without numerical effects

4 Novelty

Unum arithmetic is less than three years old; some public descriptions were made prior to the publication of primary text on the subject in February 2015, but it is quite recent that a viable alternative to the 1980s-era float format has been found and tested extensively. A groundswell of activity has started to develop libraries and convert languages and applications to use the new format.

Type 2 unums are even more novel, having been developed only as of February 2016. MIT researchers recently announced a Julia language prototype for type 2 unums [5]. Julia appears to be the ideal language for unums.

Some quotes from Amazon.com reviews of [2] indicate the level of novelty of the idea:

"…a bold and brilliant proposal for a revolutionary number representation system, unum, for scientific (and potentially all other) computers. …The book is a call to action for the next stage: implementation and testing that would lead to wide-scale adoption."—*Gordon Bell*

"…a sketch of what perhaps might be the future of computing." —*Ulrich Kulisch, U. of Karlsruhe*

"…an extraordinary reinvention of computer arithmetic and elementary numerical methods from the ground up… These changes are not just marginal technical improvements… they lead to what can only be described as a radical re-foundation of elementary numerical analysiswith new methods that are free of rounding error, fully parallelizable, fully portable, easier for programmers to master, and often more economical of memory, bandwidth, and power than comparable floating point methods." —*David Jefferson, LLNL*

5 Maturity

Software for unums is maturing rapidly; multiple unum environments now exist in C, C++, Julia [4], and Python. The Python and Mathematica environments are particularly complete environments for type 1 unums. The authors are presently creating an open source C repository for type 2 unums. DARPA has funded unum development efforts and has shown a strong interest in funding application development based on unum arithmetic.

Native hardware is only in the planning stages, making it difficult to benchmark unums against floats. REX Computing has stated plans to support unums in a future version of its Neo chip. The LULESH proxy code at Lawrence Livermore National Laboratory has been ported to use unums instead of floats.

Outgoing IEEE President Tom Conte has urged the creation of an IEEE Standard for the unum data type; the inventor of the format believes this is premature.

6 Risks

Because unums are a superset of floats, they can already run any existing program that currently works with floats. The only downside of using them in that way is that they might offer no improvement but slightly increase instead of decrease the bits required.

When used as bit-efficient interval bounds, there is a risk of falling into some of the pitfalls of traditional interval arithmetic that result in bounds too loose to be useful. A variety of ways to avoid these pitfalls are described in [2], but much more experimentation is needed to know whether tight bounds can always be achieved without expertise.

7 Conclusion

As HPC attempts ever more ambitious tasks with ever less access to people with numerical analysis expertise, approximate computing may be a step in the wrong direction. A result *provably bounded* to five decimals may have far more value than a guess that displays 15 decimals, all of which *might* be wrong. The laudable goal of using fewer bits per number can instead be achieved through new formats that maximize information per bit, and that allow the computer to bear more of the numerical analysis burden. □

REFERENCES

[1] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys (CSUR)*, Vol. 48 Issue 4, May 2016

[2] J.L. Gustafson, *The End of Error—Unum Computing*, Boca Raton: CRC Press Taylor & Francis Group, 2015.

[3] J.L. Gustafson, "A Radical Approach to Computation with Real Numbers," presented at *Multicore World 2016*, New Zealand: www.johngustafson.net/presentations/Multicore2016-JLG.pptx, February 2016

[4] S. Byrne, "Implementing Unums in Julia," https://github.com/JuliaComputing/Unums.jl, 2016

[5] J.W. Merrill, "Unums 2.0 for Julia," https://github.com/jwmerrill/Pnums.jl, 2016