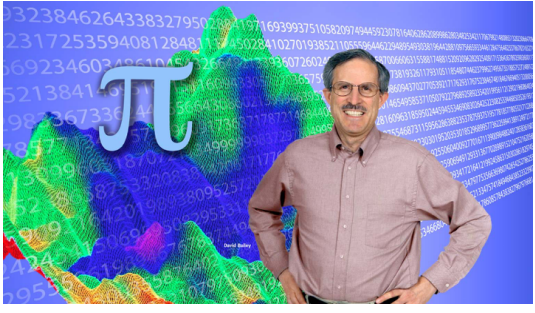
Variable precision in modern floating-point computing

David H. Bailey
Lawrence Berkeley Natlional Laboratory (retired)
University of California, Davis, Department of Computer Science



Questions to examine in this talk

- How can we ensure accuracy and reproducibility in floating-point computing?
- What types of applications require more than 64-bit precision?
- What types of applications require less than 32-bit precision?
- What software support is required for variable precision?
- How can one move efficiently between precision levels at the user level?

Commonly used formats for floating-point computing

Formal name	Nickname	Number of bits				Digits
		Sign	Exponent	Mantissa	Hidden	
IEEE 16-bit	“IEEE half”	1	5	10	1	3
(none)	“ARM half”	1	5	10	1	3
(none)	“bfloat16”	1	8	7	1	2
IEEE 32-bit	“IEEE single”	1	7	24	1	7
IEEE 64-bit	“IEEE double”	1	11	52	1	15
IEEE 80-bit	“IEEE extended”	1	15	64	0	19
IEEE 128-bit	“IEEE quad”	1	15	112	1	34
(none)	“double double”	1	11	104	2	31
(none)	“quad double”	1	11	208	4	62
(none)	“multiple”	1	varies	varies	varies	varies

Numerical reproducibility in scientific computing

A December 2012 workshop on reproducibility in scientific computing, held at Brown University, USA, noted that

> *Science is built upon the foundations of theory and experiment validated and improved through open, transparent communication. ...*
>
> *Numerical round-off error and numerical differences are greatly magnified as computational simulations are scaled up to run on highly parallel systems. As a result, it is increasingly difficult to determine whether a code has been correctly ported to a new system, because computational results quickly diverge from standard benchmark cases. And it is doubly difficult for other researchers, using independently written codes and distinct computer systems, to reproduce published results.*

- V. Stodden, D. H. Bailey, J. M. Borwein, R. J. LeVeque, W. Rider and W. Stein, "Setting the default to reproducible: Reproducibility in computational and experimental mathematics," `http://www.davidhbailey.com/dhbpapers/icerm-report.pdf`.

Reproducibility problems in a Large Hadron Collider code

- The 2012 discovery of the Higgs boson at the ATLAS experiment in the LHC relied crucially on the ability to track charged particles with exquisite precision (10 microns over a 10m length) and high reliability (over 99% of roughly 1000 charged particles per collision correctly identified).
- Software: five million line of C++ and Python code, developed by roughly 2000 physicists and engineers over 15 years.

Recently, in an attempt to speed up the calculation, researchers found that merely changing the underlying math library resulted in some collisions being missed or misidentified.

Questions:

- How extensive are these numerical difficulties?
- How can numerically sensitive code be tracked down?
- How can the library be maintained, producing numerically reliable and reproducible results?

Other numerical reproducibility issues and remedies

- Berkeley Lab climate atmospheric modeling code: Researchers found that when their code was merely ported from one system to another, or run with a different number of processors, computed results quickly diverged from a benchmark run. Had they ported the code correctly? Which run was "correct"?
- Solution: Researchers found that almost all of the numerical variability disappeared by employing double-double arithmetic (roughly 31-digit arithmetic) in two key summation loops. Total run time was unchanged.
- Livermore Lab computational physics code: Researchers working with some large computational physics applications reported similar difficulties with reproducibility.
- Solution: Researchers solved the problem by employing a "pseudo-double-double" software scheme in several critical summation loops.

More examples of high precision applications will be presented later in the talk.

- Y. He and C. Ding, "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications," *J. Supercomputing*, vol. 18 (2001), pg. 259–277.
- R. W. Robey, J. M. Robey, and R. Aulwes, "In search of numerical consistency in parallel programming," *Parallel Computing*, vol. 37 (2011), pg. 217–219.

How to enhance numerical reproducibility and solve accuracy problems

1. Employ an expert numerical analyst to examine every algorithm employed in the code, to ensure that only the most stable and efficient schemes are being used.
2. Employ an expert numerical analyst to analyze every section of code for numerical sensitivity.
3. Employ interval arithmetic for large portions of the application (which will greatly increases run time and code complexity).
4. Employ higher-precision arithmetic, assisted with some "smart" tools to help determine where extra precision is needed and where it is not.

#4 is the only practical solution for real-world computing.

Numerical analysis expertise among U.C. Berkeley graduates

Of the 2010 U.C. Berkeley graduating class, 870 were in disciplines requiring technical computing (count by DHB):

- Division of Mathematical and Physical Sciences (Math, Physics, Statistics).
- College of Chemistry.
- College of Engineering (including Computer Science).

Other fields whose graduates will likely do significant computing:

- Finance, biology, geology, medicine, economics, psychology and sociology.

The total count is likely well over 1000; maybe closer to 2000.

Enrollment in numerical analysis courses:

- Math 128A (Introductory numerical analysis required of math majors): 219.
- Math 128B (A more advanced course, required to do serious work): 24.

Conclusion: At most, only 2.4% of U.C. Berkeley graduates who will do technical computing in their careers have had rigorous training in numerical analysis.

The U.C. Berkeley/U.C. Davis "Precimonious" tool

Objective: Develop software facilities to find and ameliorate numerical anomalies in large-scale computations:

- Facilities to test the level of numerical accuracy required for an application.
- Facilities to delimit the portions of code that are inaccurate.
- Facilities to search the space of possible code modifications.
- Facilities to repair numerical difficulties, including usage of high-precision arithmetic.
- Facilities to navigate through a hierarchy of precision levels (32-bit, 64-bit, 80-bit or higher as needed).

The current version of this tool is known as "Precimonious."

- C. Rubio-Gonzalez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey and C. Iancu, "Precimonious: Tuning assistant for floating-point precision," *Proceedings of SC13*, November 2013, `https://www.davidhbailey.com/dhbpapers/precimonious.pdf`.
- C. Nguyen, C. Rubio-Gonzalez, B. Mehne, K. Sen, C. Iancu, J. Demmel, W. Kahan, W. Lavrijsen, D. H. Bailey and D. Hough, "Floating-point precision tuning using blame analysis," *Proceedings of the 38th International Conference on Software Engineering* (ICSE 2016), 14–22 May 2016, `https://www.davidhbailey.com/dhbpapers/blame-analysis.pdf`.

Can we achieve bit-for-bit floating-point replicability between systems?

- Although the IEEE floating-point standards guarantee bit-for-bit results (provided the same rounding mode is used) for individual operations, different systems may produce different higher-level results, because of non-associativity: $(A \oplus B) \oplus C \neq A \oplus (B \oplus C)$.
- Recently some researchers have proposed to establish a standard that would guarantee bit-for-bit replicability, first on a single system and subsequently on a parallel system. For details, contact Michael Mascagni of Florida State University.

Dangers: If this standard becomes the default on computer systems:

- Will it lock into place computations that are seriously inaccurate?
- Will it make it more difficult for a user to even be aware that he/she has a problem with numerical sensitivity and/or inaccuracy?

Be careful what you wish for. You may get it...

Why are code developers reluctant to employ high-precision arithmetic?

- Many are persuaded that physical reality fundamentally does not require high precision, beyond, say, the limits of 64-bit IEEE arithmetic. Invalid.
- Many believe that numerical problems are always due to the usage of inferior algorithms. Invalid.
- Many complain that easy-to-use high-precision arithmetic software facilities are not widely available. Invalid.
- Many regard the usage of high-precision arithmetic as "cheating" or "sinful." Why? Because it is too easy?
- Many are concerned about the 10–50X increase in run time (for IEEE quad). This is a legitimate concern. Can this be improved?

Innocuous example where standard 64-bit precision is inadequate

Problem: Find a polynomial to fit the data $(1, 1048579, 16777489, 84941299, 268501249, 655751251, 1360635409, 2523398179, 4311748609)$ for arguments $0, 1, \cdots, 8$. The usual approach is to solve the linear system:

$$\begin{bmatrix} n & \sum_{k=1}^{n} x_k & \cdots & \sum_{k=1}^{n} x_k^n \\ \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 & \cdots & \sum_{k=1}^{n} x_k^{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} x_k^n & \sum_{k=1} x_k^{n+1} & \cdots & \sum_{k=1}^{n} x_k^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{n} y_k \\ \sum_{k=1}^{n} x_k y_k \\ \vdots \\ \sum_{k=1}^{n} x_k^n y_k \end{bmatrix}.$$

64-bit arithmetic fails to find the correct polynomial using Matlab, Linpack or LAPACK, even if one rounds results to nearest integer.

IEEE quad or double-double arithmetic quickly produces the correct polynomial:

$$f(x) = 1 + 1048577x^4 + x^8 \;=\; 1 + (2^{20} + 1)x^4 + x^8.$$

Innocuous example, cont.

The result on the previous page can be obtained with double precision using Lagrange interpolation or the Demmel-Koev algorithm. But few outside of expert numerical analysts are aware of these schemes.

Besides, even these schemes fail for larger problems, such as:
$(1, 134217731, 8589938753, 97845255883, 549772595201,$
$2097396156251, 6264239146561, 15804422886323, 35253091827713,$
$71611233653971, 135217729000001, 240913322581691, 409688091758593)$.

These values are generated by the simple polynomial

$$f(x) = 1 + 134217729x^6 + x^{12} \;=\; 1 + (2^{27} + 1)x^6 + x^{12},$$

which is easily found using quad or double-double arithmetic.

Types of applications that may require more than 64-bit precision

1. Large-scale, highly parallel simulations, running on systems with hundreds of thousands or millions of processors — numerical sensitivities are greatly magnified.
2. Certain large linear programming calculations.
3. Certain large applications with highly ill-conditioned linear systems.
4. Large summations, especially those involving $+/-$ terms and cancellations.
5. Long-time, iterative simulations.
6. Studies in computational physics and experimental mathematics — tens, hundreds or thousands of digits may be required.

As applications continue to scale up in size, numerical difficulties will grow.

Some specific applications of more than 64-bit precision

1. Linear programming applications in biology and other fields (34 digits).
2. Planetary orbit calculations (32 digits).
3. Supernova simulations (32–64 digits).
4. Climate modeling (32 digits).
5. Coulomb n-body atomic system simulations (32–120 digits).
6. Schrodinger solutions for lithium and helium atoms (32 digits).
7. Electromagnetic scattering theory (32–100 digits).
8. Scattering amplitudes of fundamental particles (32 digits).
9. Discrete dynamical systems (32 digits).
10. The Taylor algorithm for ODEs (100–600 digits).
11. Problems in experimental mathematics (100–100,000 digits).

- D. H. Bailey, R. Barrio, and J. M. Borwein, "High precision computation: Mathematical physics and dynamics," *Applied Mathematics and Computation*, vol. 218 (2012), pg. 10106–10121.

Linear programming applications in biology

- Stanford researchers are using the Constraint-Based Reconstruction and Analysis (COBRA) approach, a linear programming based scheme, to study biological processes such as metabolism and macromolecular synthesis.
- Researchers found that 32-bit arithmetic is "not useful," and that 64-bit arithmetic "may not ensure adequate accuracy," but IEEE 128-bit ("quad") produces solutions of "exceptional accuracy."

The authors conclude: We believe that quadruple-precision solutions are now practical for multiscale [linear programming] applications ..., and that they justify increased confidence as systems biologists build ever-larger models.

- Ding Ma and Michael A. Saunders, "Solving multiscale linear programs using the simplex method in quadruple precision," in M. Al-Baali et al. (eds.), *Numerical Analysis and Optimization*, Springer Proceedings in Mathematics and Statistics, vol. 134, DOI 10.1007/978-3-319-17689-5_9.
- D. Ma, L. Yang, R. M. T. Fleming, I. Thiele, B. O. Palsson and M. A. Saunders, "Reliable and efficient solution of genome-scale models of metabolism and macromolecular expression," *Nature Scientific Reports*, vol. 7, 40863 (2017), `https://www.nature.com/articles/srep40863`.

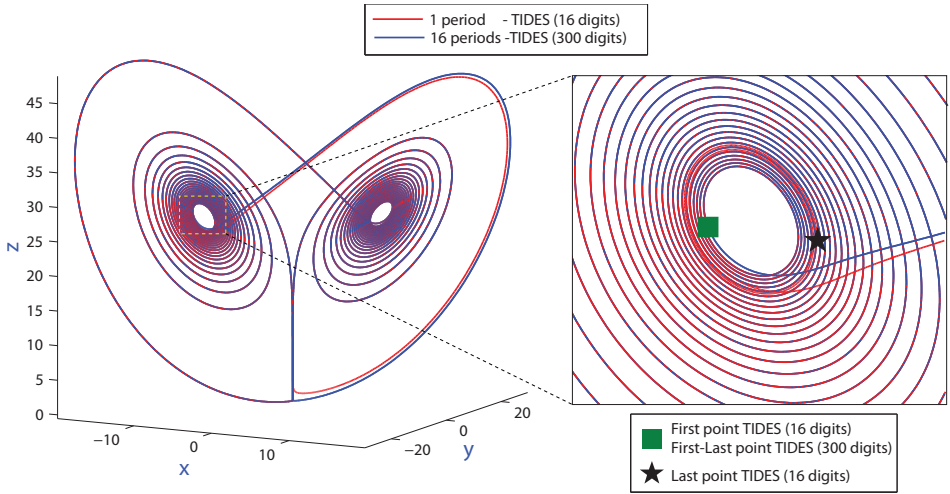
Coulomb n-body atomic system simulations

- Alexei Frolov of Queen's University in Canada has used high-precision arithmetic to solve a generalized eigenvalue problem that arises in Coulomb n-body interactions.
- Matrices are typically 5,000 x 5,000 or more in size, and are very nearly singular.
- Computations typically involve massive cancellation, and high-precision arithmetic must be employed to obtain numerically reproducible results.
- These computations typically require 120-digit arithmetic.

Frolov: "We can consider and solve the bound state few-body problems ... beyond our imagination even four years ago."

- A. M. Frolov and D. H. Bailey, "Highly accurate evaluation of the few-body auxiliary functions and four-body integrals," *Journal of Physics B*, vol. 36, no. 9 (14 May 2003), pg. 1857–1867.

Numerical solutions in chaotic systems



- R. Barrio, M. Rodríguez, A. Abad, and F. Blesa, "Breaking the limits: The Taylor series method," *Applied Mathematics and Computation*, vol. 217 (2011), 7940–7954.

Discovering new results in math and physics with very high precision

Methodology:

1. Compute various mathematical entities (limits, infinite series sums, definite integrals, etc.) to high precision, typically 100–10,000 digits.
2. Use algorithms such as PSLQ to recognize these numerical values in terms of well-known mathematical constants.
3. When results are found experimentally, seek formal mathematical proofs of the discovered relations.

Thousands of results have recently been found using this methodology, both in pure mathematics and in mathematical physics.

> *If mathematics describes an objective world just like physics, there is no reason why inductive methods should not be applied in mathematics just the same as in physics. [Kurt Godel]*

Examples of experimental mathematics in action

- Discovery of a previously unknown formula for Pi (200 digits):

$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right).$$

  This formula permits one to calculate binary digits of π beginning at an arbitrary starting position, without computing any earlier digits. It has deep implications for the randomness of π, and for the usage of π as a pseudorandom generator.

- Evaluation of Euler sums, used in quantum physics (250–1000 digits).
- Evaluation of Ising integrals, used in quantum physics and string theory (250–2000 digits).
- Evaluation of Poisson polynomials, used in mathematical physics and image processing (1000–64000 digits).

- D. H. Bailey and J. M. Borwein, "High-precision arithmetic in mathematical physics," *Mathematics*, vol. 3 (2015), pg. 337–367.
- D. H. Bailey, J. M. Borwein, J. Kimberley and W. Ladd, "Computer discovery and analysis of large Poisson polynomials," *Experimental Mathematics*, 27 Aug 2016, vol. 26, pg. 349–363.

Hardware support for IEEE 128-bit (quad) floating-point arithmetic?

- Due to the growing need for higher-than-64-bit computing, particularly given the rise of very highly parallel systems (thousands or even millions of processors), hardware support for IEEE 128-bit arithmetic seems inevitable.
- However, at the present time, it is hard to make a business case for providing such a feature, given its significant hardware cost.
- Thus, for the foreseeable future, those who do such computations will need to rely on software implementations.
- In the meantime, can anything be done to lower the performance penalty for higher precision (currently 10–50X performance)?

Quad, double-double and quad-double software

GFORTRAN:

- Now includes full support for `real(16)`, i.e., IEEE 128-bit "quad" (34 digits).
- Sadly, IEEE Quad is not yet supported in the GCC and G++ compilers.
- Available at `https://gcc.gnu.org/wiki/GFortranBinaries`

Binary128 mode in MPFR:

- Software implementation of IEEE 128-bit "quad" (34 digits).
- Operates within the MPFR arbitrary precision library.
- Low-level library only — no high-level interface yet.
- Available in MPFR development version 4.0 at `http://www.mpfr.org`

QD:

- Double-double (31 digits) and quad-double (62 digits) arithmetic.
- Includes high-level interfaces for C++ and Fortran.
- Available at `http://crd.lbl.gov/~dhbailey/mpdist`

Arbitrary precision packages

- MPFR: Low-level C library. Very high performance, thread-safe, exact rounding. Based on GMP. `http://www.mpfr.org`
- MPFR++: High-level C++ interface to MPFR. `http://www.holoborodko.com/pavel/mpfr/`
- MPFUN2015: High-level Fortran interface to MPFR. Thread-safe. Stand-alone version also available. `http://www.davidhbailey.com/mpdist`
- ARPREC: C++ package, with high-level interface for C++ and Fortran. `http://crd.lbl.gov/~dhbailey/mpdist`
- Python bigfloat: High-level Python interface to MPFR. `https://pythonhosted.org/bigfloat/`
- GPUMP: Low-level CUDA package for GPUs. `https://www.researchgate.net/publication/224175411_GPUMP_A_Multiple-Precision_Integer_Library_for_GPUs`

The attack from the bottom: New applications only require 16 bits

- The applications we have discussed so far all require more than 64-bit arithmetic, most often quad (128-bit) precision.
- But other applications have recently arisen that are calling for less than IEEE 32-bit arithmetic, in return for greater performance and larger data arrays.
- These applications typically run with IEEE half format (or the similar ARM half format), or the new bfloat16 format.

Many of the emerging low-precision applications come from the world of artificial intelligence and machine learning.

DeepMind's AlphaGo Zero teaches itself to play Go

- In October 2017, DeepMind researchers programmed an AI computer with the rules of Go, then had the program play itself — literally teaching itself with no human input.
- After just three days of training, the resulting program "AlphaGo Zero" defeated their earlier program (which had defeated Ke Jie, the highest rated human) 100 games to 0.
- The Elo rating of Ke Jie, the world's highest rated Go player, is 3661. After 40 days of training, AlphaGo Zero's Elo rating was over 5000.
- AlphaGo Zero is as far ahead of the world champion as the world champion is ahead of a good amateur.

- "The latest AI can work things out without being taught," *The Economist*, 21 Oct 2017, `https://www.economist.com/news/science-and-technology/21730391-learning-play-go-only-start-latest-ai-can-work-things-out-without`.

Artificial intelligence and big data in finance

According to a Bloomberg report, the finance world is entering a new era:

> *The fraternity of bond jockeys, derivatives mavens and stock pickers who've long personified the industry are giving way to algorithms, and soon, artificial intelligence.*

> *Banks and investment funds have been tinkering for years, prompting anxiety for employees. Now, firms are rolling out machine-learning software to suggest bets, set prices and craft hedges. The tools will relieve staff of routine tasks and offer an edge to those who stay. But one day, machines may not need much help.*

- S. Kishan, H. Son and M. Rojanasakul, "Robots are coming for these Wall Street jobs," *Bloomberg*, 18 Oct 2017, `https://www.bloomberg.com/graphics/2017-wall-street-robots/`.

Artificial intelligence and big data in finance, continued

- Credit markets: Automating subjective human decisions.
- Foreign exchange: Anticipating variations in demand and prices.
- Equity (stock) markets: Optimizing order execution, timing purchases and assessing risk.
- Credit (bond) markets: Understanding bond deals, legal documents and court rulings.

- Macroeconomics: Analyzing satellite images of Chinese industrial sites, farming regions, Walmart parking lots, etc., to spot trends in the economy. Several hedge funds and quant firms are already using such AI-based facilities to substantial advantage.

The message to the finance world is clear: Adapt to AI and machine learning, or die!

Intel's Nervana neural net software processor uses bfloat16 format

- Intel has announced a new line of processors, known as "Nervana Neural Net L-1000," targeted directly at AI applications.
- This processor supports the bfloat16 numerical format: one sign bit, 8 exponent bits and only 7 mantissa bits plus one hidden bit (roughly 2-3 digits).

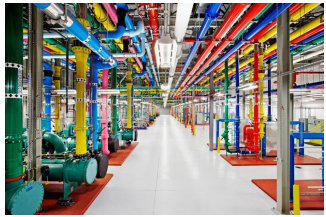
Intel's Naveen Rao, head of AI research, says:

> *Over time, Intel will be extending bfloat16 support across our AI product lines, including Intel Xeon processors and Intel FPGAs. This is part of a cohesive and comprehensive strategy to bring leading AI training capabilities to our silicon portfolio.*

- Naveen Rao, "Behyond the CPU or GPU: Why enterprise-scale artificial intelligence requires a more holistic approach," *Intel Newsroom*, 23 May 2018, `https://newsroom.intel.com/editorials/artificial-intelligence-requires-holistic-approach/`.

Google has also adopted bfloat16 for AI applications

- Google researchers have adopted bfloat16 for their TensorFlow Distributions library, which is a software framework to support AI applications.
- Their library is designed to provide fast, numerically stable methods for generating samples and computing statistics.



- J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman and R. A. Saurous, "TensorFlow Distributions," `https://arxiv.org/pdf/1711.10604.pdf`.

Challenges for low-precision computing

- Reproducibility: Only eight mantissa bits severely limits any measure of accuracy and reliability. At the least, this means that software must provide a simple means to re-run an application using 32-bit or 64-bit precision to verify some level of reliability and reproducibility.
- Software availability: So far there are only a handful of specialized packages available for 16-bit computing. Well-supported high-level language interfaces are conspicuously absent.
- Hardware compatibility: The proliferation of hardware formats (IEEE half, bfloat16) complicates the development of software.

Software support needed for variable precision

1. High-level language support. Manually calling a subroutine library is much too tedious and error-prone. To be usable for real-world applications, precision software must include fully integrated language support, so one can write, for instance,

   ```
   d = a + b * sqrt (c + 3.)
   ```

   where `a`, `b`, `c` and `d` are of possibly different precision levels, and all conversions and computations are automatically performed. Most precision software does not include high-level language support.
2. Thread-safe: Variable precision software must be thread-safe, so it can support shared-memory parallel applications. Most precision software is not thread-safe.
3. Multiple-level support: Multiple precision levels are often required in applications, since even if some sections of code can use limited precision, other sections require more. Most precision software does not provide more than one level.

Software support needed for variable precision, continued

4. A way to specify that mixed-precision computations in an expression are to be done to the maximum level of precision.

   In this line of code (C or Fortran):

   ```
   b = a + 1./3.
   ```

   the division `1./3.` is done with only single or double precision, even if `a` and `b` are declared to be of a higher-precision datatype (a similar issue with half precision).

   Most languages, compilers and precision software packages do not deal with this issue, resulting in hard-to-find errors.

   Partial solution: Provide a full set of conversion routines, so that one can write

   ```
   b = a + real (1.,16) / real (3.,16)
   ```

   where "16" (bytes) is the type specifier for quad precision, for instance.

   But it is also important that precision errors such as "`b = a + 1./3.`" be automatically detected at run time. (This is done in the MPFUN2015 software.)

Summary

- A growing number of applications (mostly in the high-performance scientific arena) require more than the IEEE 64-bit arithmetic.
- A growing number of applications (mostly in the artificial intelligence / machine learning arena) require less than IEEE 32-bit arithmetic.

The future trend is clear: Variable precision calculations, ranging from very low precision (e.g., bfloat16) to very high precision (128-bit or more) are emerging as an important fixture of both business and scientific/engineering applications. Ignore it at your peril!

This talk is available at
http://www.davidhbailey.com/dhbtalks/dhb-camb-2018.pdf