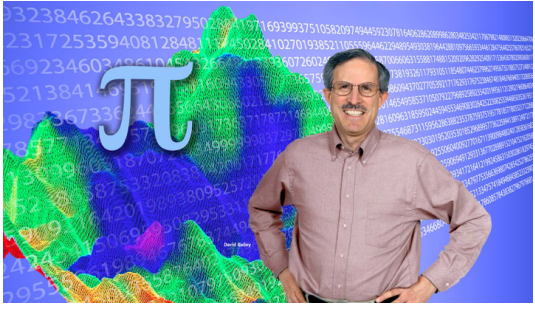
Variable precision computing: Applications and challenges

David H. Bailey
Lawrence Berkeley Natlional Laboratory (retired)
University of California, Davis, Department of Computer Science
This talk is available at:
http://www.davidhbailey.com/dhbtalks/dhb-icerm-2020.pdf



Variable precision computing

- Mixed 16-bit/32-bit: Artificial intelligence, machine learning, graphics.
- Mixed 32-bit/64-bit: A broad range of current scientific applications.
- Mixed 64-bit/128-bit: Large applications with numerically sensitive portions.
- Mixed 64-bit/multiple: Applications in computational mathematics and physics.

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	EEE 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)	double-quad	1	15	224	1	68
(none)	multiple	varies	varies	varies	varies	varies

Advantages of variable precision

Compared with using a fixed level of precision for the entire application, employing a variable precision framework features:

- Faster processing.
- Better cache utilization.
- Lower run-time memory usage.
- Lower offline data storage.
- Lower energy costs.
- Improved reproducibility and replicability.

Numerical reproducibility

Excerpt for 2012 ICERM workshop on reproducibility in mathematical and scientific computing:

Numerical reproducibility has emerged as a particularly important issue, since the scale of computations has greatly increased in recent years, particularly with computations performed on many thousands of processors and involving similarly large datasets. Large computations often greatly magnify the level of numeric error, so that numerical difficulties that were once of little import now are large enough to alter the course of the computation or to draw into question the overall validity of the 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," manuscript, 2 Feb 2013, `https://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.

In an attempt to speed up the calculation, researchers found that merely changing the underlying math library (which should only affect at most the last bit) 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 a large code library such as this be maintained, producing numerically reliable and reproducible results?

How to solve accuracy problems and ensure reproducibility

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 an expert numerical analyst to convert large portions of the code to use interval arithmetic (greatly increasing run time and code complexity).
4. Employ significantly higher precision than is really necessary (greatly increasing run time).
5. Employ variable-precision arithmetic, assisted with some "smart" tools to help determine where extra precision is needed and where it is not.

Item 5 is the only practical solution for real-world scientific 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 very likely over 1000; probably 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 about 2% of U.C. Berkeley graduates who will do technical computing in their careers have had rigorous, expert-level training in numerical analysis.

Mixed half-single applications: Machine learning

NVIDIA researchers found that a combination of IEEE 32-bit and bfloat16 arithmetic achieved nearly the same training loss reduction curve on a English language learning model as with 100% IEEE 32-bit:

Maintain a master copy of training system weights in IEEE 32-bit, and select a scaling factor S. Then for each iteration:

- Convert the array of 32-bit weights to a bfloat16 array.
- Perform forward propagation with bfloat16 weights and activations.
- Multiply the resulting loss with a scaling factor S.
- Perform backward propagation with bfloat16 weights, activations and gradients.
- Multiply the weight gradient by $1/S$.
- Update the 32-bit weights using the bfloat16 data.

- "Deep learning SDK documentation," NVIDIA, 2019, `https://docs.nvidia.com/deeplearning/sdk/index.html`.

DeepMind's AlphaGo Zero teaches itself to play Go

- In October 2017, DeepMind researchers programmed a machine learning system with the rules of Go, then had the program play itself — 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 was 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`.

Challenges for mixed half-single computing

- Reproducibility: Only eight mantissa bits severely limits any measure of accuracy and reliability. At the least, 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 relatively few specialized packages and well-supported high-level language interfaces for 16-bit computing.
- Hardware compatibility: The proliferation of hardware formats (IEEE half, bfloat16, arm16) complicates the development of software.

Mixed single-double application: LU decomposition

- The usual scheme is to factor a pivoted coefficient matrix $PA = LU$, where L is lower triangular and U is upper triangular, then solve $Lu = Pb$ and $Ux = y$, all using double precision.
- In a mixed single-double scheme, the factorization and solutions are done in single precision; then the residual and full solutions are done in double.
- Note that the factorization of A, which is the only step that is $O(n^3)$, is done entirely in single.
- A similar scheme can be used for sparse systems.
- A broad range of iterative refinement methods can be performed in this way.
- Baboulin et al. (below) achieved speedups of 1.86 on a direct sparse system on a suite of test problems.

- M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, P. Luszczek and S. Tomov, "Accelerating scientific computations with mixed precision algorithms," *Computational Physics Communications*, vol. 180 (2009), 2256–2253.

Mixed double-quad application: Computational biology

- Certain models of biochemical reaction networks, e.g., "metabolic expression models," involve solving large multiscale linear programming problems.
- Since double precision implementations of these models often produce numerically unreliable results, researchers have resorted to exact rational arithmetic methods, which are enormously expensive — run times are typically weeks or months.
- Ding Ma and Michael Sanders of Stanford have achieved excellent results using double codes enhanced with quad precision for numerically sensitive operations.
- Their code was a straightforward modification of a double precision Fortran code, using the `REAL(16)` datatype, which is supported in the GNU gfortran compiler.

- D. Ma and M. Saunders, "Solving multiscale linear programs using the simplex method in quadruple precision," in M. Al-Baali, L. Grandinetti and A. Purnama, ed., *Recent Developments in Numerical Analysis and Optimization*, Springer, NY, 2017, `http://web.stanford.edu/group/SOL/reports/quadLP3.pdf`.

Innocuous example where IEEE 64-bit precision is inadequate

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

Common approach: Use a least-squares scheme, solving the linear system:

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

An implementation using IEEE 64-bit arithmetic fails to find the correct polynomial, even if one rounds results to nearest integer.

But an implementation using 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 64-bit arithmetic using Lagrange interpolation or the Demmel-Koev algorithm. But few outside of expert numerical analysts are aware of these schemes.

Besides, even these two 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

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

This is easily found by an IEEE quad or double-double implementation of any of the three algorithms (least-squares, Lagrange or Demmel-Koev).

Mixed double/multiprecision applications

1. Solutions of ordinary differential equations in planetary orbit models (32 digits).
2. Supernova simulations (32–67 digits).
3. Scattering amplitudes of fundamental particles (32 digits).
4. Solutions of certain discrete dynamical systems (32 digits).
5. Coulomb *n*-body atomic system simulations (32–120 digits).
6. Solving ordinary differential equations using the Taylor algorithm (100–300 digits).
7. Recognizing integrals from the Ising theory of mathematical physics in terms of simple mathematical identities (200–1000 digits).
8. Finding minimal polynomials connected to the Poisson equation of mathematical physics (1000–64,000 digits).

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

Mixed double/multiprecision: Computational math and physics

Frequently-used "experimental math" methodology:

1. Compute a limit, infinite series sum, definite integral or other mathematical quantity to high precision, typically 100–10,000 digits.
2. Use an integer relation algorithm such as "PSLQ" to recognize the numerical value in terms of a mathematical expression. In other words, if the computed quantity is α, and if one suspects that $\alpha = x_0$ is given by a formula involving known constants $(x_1, x_2, \cdots, x_{n-1})$, then use PSLQ to find integers $(a_0, a_1, a_2, \cdots, a_{n-1})$ such that $a_0 x_0 + a_1 x_1 + \cdots + a_{n-1} x_{n-1} = 0$.
3. When a result is found experimentally, seek a formal mathematical proof of the discovered relation.
4. Efficient implementations typically require three levels of precision, such as double/250-digit/10,000-digit, varying up and down many times within the run, performing as much computation as possible using only double precision.

Experimental mathematics and physics results

- New formulas and identities for various mathematical constants, including π, $\log(2)$ and others, some with remarkable properties (250–1000 digits).
- 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 (1000–64,000 digits).

- D. H. Bailey and J. M. Borwein, "High-precision arithmetic in mathematical physics," *Mathematics*, vol. 3 (2015), 337–367.

Mixed double/multiprecision application: Finding Poisson polynomials

In 2012 Richard Crandall of Apple Computers found that each pixel in the image produced by a sharpening algorithm is given by the 2-D Poisson potential function of mathematical physics:

$$\phi_2(x,y) = \frac{1}{\pi^2} \sum_{m,n \text{ odd}} \frac{\cos(m\pi x)\cos(n\pi y)}{m^2+n^2}$$

Crandall and others then found that for rational x and y,

$$\phi_2(x,y) = \frac{1}{\pi} \log \alpha$$

where α is algebraic, i.e., the root of a some integer polynomial of degree m.

What is the connection between x and y and the degree of α?

- D. H. Bailey, J. M. Borwein, R. E. Crandall and J. Zucker, "Lattice sums arising from the Poisson equation," *Journal of Physics A: Mathematical and Theoretical*, vol. 46 (2013), 115201.
- 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, 349–363.

Finding Poisson polynomials, cont.

By computing high-precision numerical values of $\phi_2(x, y)$ for various specific rational x and y, using three precision levels (e.g., double/250-digit/10,000-digit), and applying a variant of the PSLQ algorithm, we found the following results among others:

s	Minimal polynomial of α corresponding to $x = y = 1/s$
5	$1 + 52\alpha - 26\alpha^2 - 12\alpha^3 + \alpha^4$
6	$1 - 28\alpha + 6\alpha^2 - 28\alpha^3 + \alpha^4$
7	$-1 - 196\alpha + 1302\alpha^2 - 14756\alpha^3 + 15673\alpha^4 + 42168\alpha^5 - 111916\alpha^6 + 82264\alpha^7$ $-35231\alpha^8 + 19852\alpha^9 - 2954\alpha^{10} - 308\alpha^{11} + 7\alpha^{12}$
8	$1 - 88\alpha + 92\alpha^2 - 872\alpha^3 + 1990\alpha^4 - 872\alpha^5 + 92\alpha^6 - 88\alpha^7 + \alpha^8$
9	$-1 - 534\alpha + 10923\alpha^2 - 342864\alpha^3 + 2304684\alpha^4 - 7820712\alpha^5 + 13729068\alpha^6$ $-22321584\alpha^7 + 39775986\alpha^8 - 44431044\alpha^9 + 19899882\alpha^{10} + 3546576\alpha^{11}$ $-8458020\alpha^{12} + 4009176\alpha^{13} - 273348\alpha^{14} + 121392\alpha^{15}$ $-11385\alpha^{16} - 342\alpha^{17} + 3\alpha^{18}$
10	$1 - 216\alpha + 860\alpha^2 - 744\alpha^3 + 454\alpha^4 - 744\alpha^5 + 860\alpha^6 - 216\alpha^7 + \alpha^8$

Finding Poisson polynomials, cont.

Based on these preliminary results, Jason Kimberley conjectured that the degree $m(s)$ of the minimal polynomial associated with the case $x = y = 1/s$ is given by this rule:

> *Set $m(2) = 1/2$. Otherwise for primes p congruent to 1 mod 4, set $m(p) = \mathrm{int}^2(p/2)$, where int denotes greatest integer, and for primes p congruent to 3 mod 4, set $m(p) = \mathrm{int}\,(p/2)(\mathrm{int}\,(p/2) + 1)$. Then for any other positive integer s whose prime factorization is $s = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$,*

$$m(s) = 4^{r-1} \prod_{i=1}^{r} p_i^{2(e_i-1)}\, m(p_i).$$

Additional three-level precision computations, (e.g., double/2000-digit/64,000-digit), confirmed Kimberley's formula for all s up to 40, and for even s up to 52.

By examining the computed results, connections were found to a sequence of polynomials defined in a 2010 paper by Savin and Quarfoot. These observations ultimately led to a proof of Kimberley's formula and several other facts.

Software and language issues: Conversions between types

Existing programming languages have significant issues with varying precision levels, particularly in conversions between types. For example, consider:

```
program testprec
real(8) d1, d2, d3
d1 = sqrt (3. + 1./7.)
d2 = sqrt (3.d0 + 1./7.)
d3 = sqrt (3. + 1.d0/7.)
write (6, '(f20.15)') d1, d2, d3
stop
end
```

This code, when compiled with the gfortran compiler with default options, produces:

```
1.772810459136963
1.772810522656991
1.772810520855837
```

Only the last is a correct, full-precision value!

One could set a compiler flag to treat all single constants as double (default in some languages), but this defeats the goal of using varying precision levels.

Software and language issues: Conversions between types, cont.

Similar difficulties, greatly magnified, afflict mixed double-multiprecision applications:

```
program testprec
use mpmodule
type (mp_real) r1, r2, r3
r1 = sqrt (3.d0 + 1.d0/7.d0)
r2 = sqrt (mpreal (3.d0, 10) + 1.d0/7.d0)
r3 = sqrt (3.d0 + mpreal (1.d0, 10)/7.d0)
call mpwrite (6, 80, 60, r1, r2, r3)
stop
end
```

DHB's MPFUN-MPFR package checks for conversion problems like this at runtime, producing this error message:

```
*** MP_CHECKDP: DP value has more than 40 significant bits:
1.772810520855837D+00
and thus very likely represents an unintended loss of accuracy.
```

One solution: A compiler option to outlaw all mixed-mode operations and automatic type conversions. Unpopular? Perhaps. But finding precision problems is even harder.

Thread safety in high precision libraries

Most multiprecision libraries in use today are not thread-safe. Thus applications written to use them do not work in a shared-memory parallel environment.

The solution is to include the following in each multiprecision datum:

- The full length of the storage allocated to this datum.
- The current working precision associated with this datum.
- The length of nonzero data associated with this datum.

Currently the MPFR package (see below) is the only low-level multiprecision library that is thread-safe (provided thread-safe option is set); it is also the fastest.

DHB's MPFUN-MPFR high-level Fortran multiprecision package is built on MPFR.

- L. Fousse, G. Hanrot, V. Lefevre, P. Pelissier and P. Zimmermann, "MPFR: A multiple precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software*, vol. 33 (2007), `https://doi.org/10.1145/1236463.1236468`.

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).

- 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`.

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 an addition to the IEEE-754 standard that would permit guaranteed bit-for-bit replicability, not only on a single system but also on a parallel system — see references below.
- ARM (a U.K. computer processor vendor) has experimented with a high-precision accumulator, using an adaptable vector unit.

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 users to even be aware that they have a problem with numerical sensitivity and/or inaccuracy?

- P. Ahrens, J. Demmel and H. D. Nguyen, "Algorithms for efficient reproducible floating point summation," manuscript available from authors.
- J. Riedy and J. Demmel, "Augmented arithmetic operations proposed for IEEE-754 2018," `http://www.ecs.umass.edu/arith-2018/pdf/arith25_34.pdf`.

New approaches to floating-point arithmetic

- Although the IEEE-754 floating-point standard has served us very well for several decades, some wonder if it time to completely rethink floating-point arithmetic.
- Gustafson and Yonemoto have proposed "Unum," a floating-point standard that is flexible in size and precision — see references below.
- Lindstrom summarizes Unums and some other proposals — see below.

- J. L. Gustafson, *The End of Error: Unum Computing*, Chapman and Hall, New York, 2017.
- J. Gustafson and I. Yonemoto, "Beating floating point at its own game: Posit arithmetic," 2017, `http://www.johngustafson.net/pdfs/BeatingFloatingPoint.pdf`.
- P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20 (2014), 2674–2683, `https://doi.org/10.1109/TVCG.2014.2346458`.

Research questions to be explored

1. How are various scientific disciplines currently using variable precision? What new opportunities are on the horizon?
2. What does the explosive growth in artificial intelligence and machine learning mean for hardware, software and tools to support variable precision? What are the fundamental accuracy requirements of such computations?
3. Are there new mathematical approaches to efficiently and accurately compute various mathematical operations and library functions to both very modest precision (2-3 digit accuracy) and very high precision (hundreds or thousands of digits)?
4. What new formats and representation systems to facilitate variable level precision computing are available? To what extent do these new proposals overcome strengths and weaknesses in existing systems such as IEEE-754 hardware and MPFR software?
5. What new support is needed from computer hardware and software vendors for variable precision computing? Could some modest hardware changes be made to support low, variable precision and high precision, rather than relying on all-software implementations?
6. Almost all of published research in numerical mathematics and numerical analysis to date has focused on either 32-bit or 64-bit results. Are there other techniques that may be superior in a variable precision environment?

Research questions to be explored, cont.

6. Classical error bounds used in numerical analysis, such as those developed by Wilkinson, typically assume worst-case scenarios and thus are not very useful in a limited precision or variable precision environment. Can more realistic error estimates be derived?
7. How can iterative refinement best be performed in a variable precision environment? What new opportunities are there for using iterative refinement with variable precision?
8. What algorithms and software techniques can effectively compress floating-point data, on-the-fly, to save data storage and transfer? Can improved schemes be devised?
9. What software tools are available to automatically or semi-automatically analyze an application code, identify numerically sensitive spots and propose or implement remedies? What other tools are needed?
10. How can we ensure that computations with variable precision levels are reproducible? What does reproducibility even mean when computing, say, with only 16-bit precision?

This talk is available at http://www.davidhbailey.com/dhbtalks/dhb-icerm-2020.pdf.