HIGH-PRECISION FLOATING-POINT ARITHMETIC IN SCIENTIFIC COMPUTATION

IEEE 64-bit floating-point arithmetic is sufficient for most scientific applications, but a rapidly growing body of scientific computing applications requires a higher level of numeric precision. New software packages have yielded interesting scientific results that suggest numeric precision in scientific computations could be as important to program design as algorithms and data structures.

Virtually all present-day computer systems, from PCs to the largest supercomputers, implement the IEEE 64-bit floating-point arithmetic standard, which provides 53 mantissa bits, or approximately 16-decimal-digit accuracy. For most scientific applications, this is more than sufficient—in fact, for some applications (such as routine processing of experimental data), the 32-bit standard often provides sufficient accuracy.

However, for a rapidly expanding body of applications—ranging from interesting new mathematical computations to large-scale physical simulations performed on highly parallel supercomputers—64-bit IEEE arithmetic is no longer sufficient. In these applications, portions of the code typically involve numerically sensitive calculations that produce results of questionable accuracy using conventional arithmetic. These inaccurate results could in turn induce other errors, such as taking the wrong path in a conditional branch.

Few of the scientists and engineers currently involved in technical computing have rigorous backgrounds in numerical analysis. What's more, this scarcity of numerical expertise is likely to worsen in the future rather than improve, in part because of the dearth of students and young researchers interested in numerical mathematics. Thus, while some might argue that numerically sensitive calculations can be remedied by using different algorithms or coding techniques, in practice such changes are highly error-prone and very expensive. It's usually easier, cheaper, and more reliable to use high-precision arithmetic to overcome these difficulties, even if other remedies are theoretically feasible.

This article focuses on high-precision floating-point (rather than integer) arithmetic, although high-precision (that is, multiword) integer arithmetic is an interesting arena in its own right, with numerous applications in both pure and applied mathematics. When you use a secure Web site to purchase a book or computer accessory, for example, at some point the browser software performs high-precision integer computations to communicate credit-card numbers and other secure information. Closely related to this technology is ongoing research in large integer factorization algorithms, in which several remarkable advances have recently been made.[1]

1521-9615/05/$20.00 © 2005 IEEE
Copublished by the IEEE CS and the AIP

DAVID H. BAILEY
Lawrence Berkeley National Laboratory

High-Precision Software

Software packages that perform high-precision floating-point arithmetic have been available since the early days of computing—a package Richard Brent wrote, for example, has been in use since the 1970s.[2] However, many of these packages require the user to rewrite a scientific application with individual subroutine calls for each arithmetic operation. The difficulty of making such changes, and the difficulty of debugging the resulting code, has deterred all but a few scientists from using such software. In the past few years, though, high-precision software packages have increasingly included high-level language interfaces that make such conversions relatively painless. These packages typically use custom data types and operator overload features, which are now available in languages such as C++ and Fortran 90, to facilitate conversion.

Even more advanced high-precision computation facilities are available in the commercial products Mathematica and Maple, which incorporate arbitrary-precision arithmetic in a very natural way. Although these packages are extremely useful in some contexts, they're generally not as fast as specialized high-precision arithmetic packages, and they don't provide a means to convert existing scientific programs written in, say, Fortran 90 or C++, to use their high-precision arithmetic facilities.

The following list gives some examples of freely available high-precision arithmetic software packages (ARPREC, DDFUN, QD, and MPFUN90 are available on my Web site: http://crd.lbl.gov/~dhbailey/mpdist or from www.experimentalmath.info):

- *ARPREC* includes routines to perform arithmetic with an arbitrarily high (but pre-set) level of precision, including many algebraic and transcendental functions. High-level language interfaces available for C++ and Fortran 90 support real, integer, and complex data types. Xiaoye S. Li, Brandon Thompson, and I wrote this software.
- *DDFUN* provides "double-double" (approximately 31 digits) arithmetic in an all-Fortran 90 environment. A high-level language interface supports real, integer, and complex data types. This package is much faster than using arbitrary precision or "quad-double" (approximately 62 digits) software in applications where 31 digits are sufficient, and it's often faster than using the `real*16` data type available in some Fortran systems.
- *FMLIB* is a low-level multiple-precision library, and *FMZM90* provides high-level Fortran 90 language interfaces for real, integer, and complex data types. David Smith of Loyola Marymount University wrote this software (http://myweb.lmu.edu/dmsmith/FMLIB.html).
- *GMP* includes an extensive library of routines to support high-precision integer, rational, and floating-point calculations. Produced by a volunteer effort, GMP is distributed under a GNU license by the Free Software Foundation (www.swox.com/gmp).
- *Predicates* was specifically designed to perform the numerically sensitive operations that often arise in computational geometry. Jonathan Shewchuk of the University of California, Berkeley, wrote this software (http://www-2.cs.cmu.edu/~quake/robust.html).
- *QD* includes routines to perform double-double and quad-double arithmetic. High-level language interfaces are available for C++ and Fortran 90, supporting real, integer, and complex data types. The QD package is much faster than using arbitrary-precision software in applications where 31 or 62 digits are sufficient. Xiaoye S. Li, Yozo Hida, and I wrote this software.
- The *MPFR* library is a C library for multiple-precision floating-point computations with exact rounding, and is based on the GMP multiple-precision library (www.mpfr.org).
- *MPFR++* is a high-level C++ interface to MPFR (http://perso.ens-lyon.fr/nathalie.revol/software.html).
- *MPFUN90* is equivalent to ARPREC in its user-level functionality, but it's written entirely in Fortran 90 and provides a Fortran 90 language interface.
- *VPA* provides an arbitrary-precision functionality together with a Fortran language interface. J.L. Schonfelder of the UK's N.A. Software wrote this application (http://pcwww.liv.ac.uk/~jls/vpa20.htm).

Several of these packages include sample application programs—for example, the ARPREC and MPFUN90 packages include programs that implement the PSLQ algorithm for integer-relation detection. The ARPREC, MPFUN90, and QD packages also include programs to evaluate definite integrals to high precision. In fact, it's now possible to evaluate many integrals that arise in mathematics or physics to very high accuracy, even in cases in which the integrand functions have singularities such as vertical derivatives or infinite values at endpoints. When these high-precision integration schemes are combined with integer-relation-detection methods, it's often possible to obtain analytic evaluations of definite integrals that otherwise have no known solution.[3,4]

Using high-precision software definitely increases computer runtimes (compared to using conventional machine precision). Computations using double-double precision, for example, typically run five times longer than with ordinary 64-bit arithmetic. This figure rises to 25 times for quad-double arithmetic, and to more than 50 times for 100-digit arithmetic. Beyond 100-digit-precision, the computational cost for precision p increases roughly as p^2 up to roughly 1,000 digits, after which the cost increases as roughly $p \log p$ (presuming we're using fast Fourier transform [FFT]-based multiplication, which is available in ARPREC and MPFUN90). Fortunately, it's often not necessary to employ high-precision arithmetic for the entire calculation; one critical loop is often all that requires this higher accuracy.

Let's examine some of the scientific applications that use high-precision arithmetic.

Climate Modeling

We all know that weather and climate simulations are fundamentally chaotic—if microscopic changes occur in the present state, within a certain period of simulated time, the future state will change completely. As a result, computational scientists involved in climate-modeling applications have long resigned themselves to knowing that their codes quickly diverge from any baseline calculation, even if they only change the number of processors used to run the code. As a result, it's not only difficult for researchers to compare results, but it's often problematic to even determine whether they correctly deployed their code on a given system.

Recently, Helen He and Chris Ding at Lawrence Berkeley National Laboratory investigated this non-reproducibility phenomenon in a widely used climate-modeling code.[5] They observed that almost all the numerical variation occurred in one inner product loop in the atmospheric data assimilation step and in a similar operation in a large conjugate gradient calculation. He and Ding found that a straightforward solution was to employ double-double arithmetic (using the DDFUN package) for these loops. This single change dramatically reduced the entire application's numerical variability, permitting computer runs to be compared for much longer runtimes than before.

In retrospect, it's not clear that handling these sums in this manner is the best solution—other approaches could preserve meaningful numerical accuracy and yield reproducible results. Such phenomena deserve further study and are currently being investigated, but in the meantime, He and Ding's solution is a straightforward and effective way to deal with this problem.

Supernova Simulations

Edward Baron, Peter Hauschildt, and Peter Nugent recently used the QD package, which provides double-double (128-bit or 31-digit) and quad-double (256-bit or 62-digit) data types, to solve for the nonlocal thermodynamic equilibrium populations of iron and other atoms in the atmospheres of supernovae and astrophysical objects.[6] Iron, for example, could exist as Fe II in the outer parts of the atmosphere, but in the inner parts, Fe IV or Fe V might dominate. Introducing artificial cutoffs leads to numerical glitches, so it is necessary to solve for all these populations simultaneously. Because the relative population of any state from the dominant stage is proportional to the exponential of the ionization energy, the dynamic range of these numerical values can be quite large.

To handle this potentially very large dynamic range, yet also perform the computation in reasonable time, Baron, Hauschildt, and Nugent employed a dynamic scheme to determine whether to use 64-bit, 128-bit, or 256-bit arithmetic both in constructing the matrix elements and in solving the linear system. The runtime of the code using 256-bit arithmetic took up a large fraction of the total runtime, so the researchers are planning an even faster version of the 256-bit routines to reduce these runtimes.

Coulomb *N*-Body Atomic System Simulations

Researchers perform numerous computations with high-precision arithmetic to study atomic-level Coulomb systems. Alexei Frolov of Queen's University in Ontario, Canada, for example, used high-precision software to solve the generalized eigenvalue problem $(\hat{H} - E\hat{S})C = 0$, where the matrices $\hat{H}$ and $\hat{S}$ are large (typically 5,000 × 5,000 in size) and very nearly degenerate.[7,8] Until recently, progress in this arena was severely hampered by the numerical difficulties these matrices induce.

Frolov did his calculations using the MPFUN package, with a numeric precision level exceeding 100 digits. He noted to me that in this way, "we can consider and solve the bound state few-body problems which have been beyond our imagination even four years ago." He also used MPFUN to compute the matrix elements of the Hamiltonian matrix $\hat{H}$ and the overlap matrix $\hat{S}$ in four- and five-body atomic problems. Future plans include generalizing this method for use in photodetachment and scattering problems.

Studies of the Fine Structure Constant

In the past few years, researchers have made significant progress in using high-precision arithmetic to obtain highly accurate solutions to the Schrödinger equation for the lithium atom. In particular, they've calculated the nonrelativistic ground-state energy to an accuracy of a few parts in a trillion, a factor of 1,500 improvement over the best previous results. With these highly accurate wave functions, Zong-Chao Yan and others have tested the relativistic and quantum electrodynamics effects at the 50 parts per million (ppm) level as well as the 1 ppm level.[9] Yan has also calculated several properties of lithium and lithium-like ions, including the oscillator strengths for certain resonant transitions, isotope shifts in some states, dispersion coefficients, and Casimir-Polder effects between two lithium atoms.

Theoretical calculations of the fine structure splittings in helium atoms have now advanced to the planning of highly accurate experiments. When certain additional computations finish, we could get a unique atomic physics value of the fine structure constant to an accuracy of 16 parts per billion.[10]

Electromagnetic Scattering Theory

A key operation in computational studies of electromagnetic scattering is to find the branch points of the spheroidal wave function's asymptotic expansion, typically by using Newton-Raphson iterations. Unfortunately, the accuracy of the results from conventional 64-bit arithmetic for these Newton iterations significantly limits the range of the wave functions that we can study.

Ben Barrowes has used the MPFUN package to remedy this situation.[11] His calculation required the conversion of a large body of existing code, which was facilitated by the Fortran 90 language translation modules in the MPFUN package. However, in the course of his work, Barrowes found that arbitrary-precision versions of the Fortran 90 array operations were required. He then worked with the author to implement these functions, which are now available as an optional part of the MPFUN90 package.

Vortex Sheet Roll-Up Simulations

Researchers have exploited high-precision arithmetic for some time in their attempts to resolve complex phenomena associated with fluid flows. In 1993, for example, Russel Caflisch used 32-, 64-, and even 128-digit arithmetic in studies of singularity formation under three-dimensional incompressible Euler flows.[12]

A long-standing problem of fluid dynamics is whether a fluid undergoing vortex sheet roll-up assumes a true power law spiral. This problem's resulting calculation, which is fairly long-running even with standard precision, becomes prohibitively expensive on a single-processor system with high-precision arithmetic. However, Robert Krasny, Richard Pelz, and I were able to implement the calculation on a highly parallel system using 64 processors.[13] The calculation ultimately used 10,000 CPU-hours, and the result showed that when we reduce a critical parameter below 0.02, the solution is no longer a smooth spiral, but instead develops "blisters" that aren't merely artifacts of insufficient precision (see Figure 1). These blisters aren't well understood, and thus warrant further investigation.

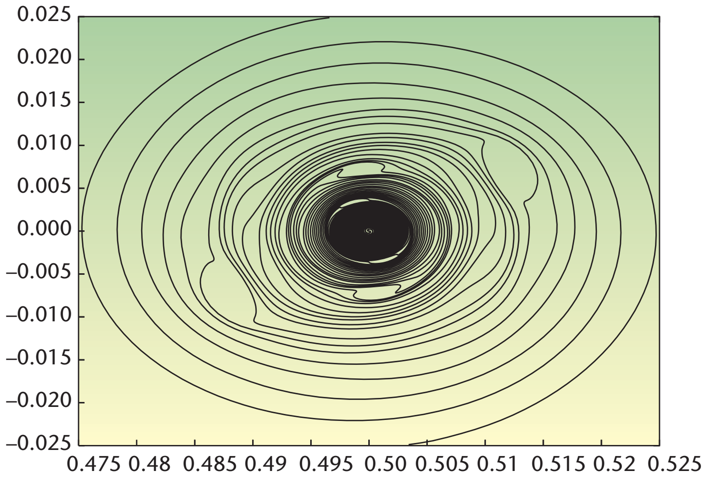

Figure 1. Vortex roll-up computed with quad-double arithmetic. The "blisters" aren't merely artifacts of insufficient precision; their source is not yet understood.

Computational Geometry and Grid Generation

Grid generation, contour mapping, and several other computational geometry applications rely on highly accurate arithmetic. To prove this, William Kahan and Joseph Darcy showed that small numerical errors in the computation of the point nearest to a given point on a line intersecting two planes can result in the computed point being so far from either plane as to rule out the solution being correct for a reasonable perturbation of the original problem.[14]

$$\pi\sqrt{3} = \frac{9}{32}\sum_{k=0}^{\infty}\frac{1}{64^k}\left(\frac{16}{6k+1}-\frac{8}{6k+2}-\frac{2}{6k+4}-\frac{1}{6k+5}\right)$$

$$\pi^2 = \frac{1}{8}\sum_{k=0}^{\infty}\frac{1}{64^k}\left[\frac{144}{(6k+1)^2}-\frac{216}{(6k+2)^2}-\frac{72}{(6k+3)^2}-\frac{54}{(6k+4)^2}+\frac{9}{(6k+5)^2}\right]$$

$$\begin{aligned}\pi^2 = \frac{2}{27}\sum_{k=0}^{\infty}\frac{1}{729^k}\Bigg[&\frac{243}{(12k+1)^2}-\frac{405}{(12k+2)^2}-\frac{81}{(12k+4)^2}-\frac{27}{(12k+5)^2}\\ &-\frac{72}{(12k+6)^2}-\frac{9}{(12k+7)^2}-\frac{9}{(12k+8)^2}-\frac{5}{(12k+10)^2}+\frac{1}{(12k+11)^2}\Bigg]\end{aligned}$$

$$\log 3 = \frac{1}{729}\sum_{k=0}^{\infty}\frac{1}{729^k}\left(\frac{729}{6k+1}+\frac{81}{6k+2}+\frac{81}{6k+3}+\frac{9}{6k+4}+\frac{9}{6k+5}+\frac{1}{6k+6}\right)$$

$$\begin{aligned}\pi\log 2 = \frac{1}{256}\sum_{k=0}^{\infty}\frac{1}{4096^k}\Bigg[&\frac{4096}{(24k+1)^2}-\frac{8192}{(24k+2)^2}-\frac{26112}{(24k+3)^2}+\frac{15360}{(24k+4)^2}\\ &-\frac{1024}{(24k+5)^2}+\frac{9984}{(24k+6)^2}+\frac{11520}{(24k+8)^2}+\frac{2368}{(24k+9)^2}-\frac{512}{(24k+10)^2}\\ &+\frac{768}{(24k+12)^2}-\frac{64}{(24k+13)^2}+\frac{408}{(24k+15)^2}+\frac{720}{(24k+16)^2}\\ &+\frac{16}{(24k+17)^2}+\frac{196}{(24k+18)^2}+\frac{60}{(24k+20)^2}-\frac{37}{(24k+21)^2}\Bigg]\end{aligned}$$

$$\begin{aligned}0 \overset{?}{=} &-2J_2-2J_3-2J_4+2J_{10}+2J_{11}+3J_{12}+3J_{13}+J_{14}-J_{15}-J_{16}\\ &-J_{17}-J_{18}-J_{19}+J_{20}+J_{21}-J_{22}-J_{23}+2J_{25}\end{aligned}$$

Figure 2. Some new math identities found by high-precision computations.

Two commonly used computational geometry operations are the orientation test and the in-circle test. The orientation test attempts to unambiguously determine whether a point lies to the left of, to the right of, or on a line or plane defined by other points. In a similar way, the in-circle test determines whether a point lies inside, outside, or on a circle defined by other points. Each of these tests is typically performed by evaluating the sign of a determinant expressed in terms of the points' coordinates. If these coordinates are expressed as single- or double-precision floating-point numbers, round-off error might lead to an incorrect result when the true determinant nears zero. In turn, this misinformation can lead an application to fail or produce incorrect results, as noted earlier.

To remedy such problems, Jonathan Shewchuk produced a software package called Predicates (http://www-2.cs.cmu.edu/~quake/robust.html) that performs "adaptive" floating-point arithmetic, which dynamically increases numeric precision until an unambiguous result is obtained.

Computational Number Theory

Computational number theory is an area of current mathematical research that has applications in fields such as cryptography. These computations frequently involve high-precision arithmetic in general and high-precision floating-point arithmetic in particular. For example, the current largest known explicit prime number—namely, $2^{24036583} - 1$ (with 7 million decimal digits)—was recently proven prime via the Lucas-Lehmer test. This test requires very large-integer arithmetic, which in turn uses FFTs to perform multiplications; these multiplications are merely large linear convolutions.[1] The numerical accuracy of IEEE arithmetic limits the size of these convolutions that can be done reliably—unless we store just a few bits in each word of memory (which means we waste memory). Thus, fast double-double arithmetic has a significant advantage in these computations.

Many questions in mathematical number theory revolve around the celebrated Riemann zeta function

$$\varsigma(s) = \sum_{k=1}^{\infty}\frac{1}{n^s}\ , \qquad (1)$$

which, among other things, encodes profound information about the distribution of prime numbers. The best method currently known for evaluating the classical function $\pi(x)$, the number of primes not exceeding x, involves calculations of $\zeta(3/2 + it)$, but Will Galway has observed that calculating zeta using IEEE 64-bit arithmetic only lets us accurately compute $\pi(x)$ for x up to 10^{13}.[1] Thus, there is considerable interest in using higher-precision floating-point arithmetic, which will permit researchers to press up toward the current frontier of knowledge—namely, $\pi(10^{22})$.

Experimental Mathematics

High-precision computations have proven to be an essential tool for the emerging discipline of experimental mathematics—namely, the usage of modern computing technology as an active agent of exploration in mathematical research.[15,16] One of the key techniques used here is the PSLQ integer-relation-detection algorithm, which searches for linear relationships satisfied by a set of numerical values.[17] Integer-relation computations require very high precision in the input vector to obtain numerically meaningful results. Computations with several hundred digits are typical, although one extreme instance required 50,000-digit arithmetic to obtain the desired result.[18]

The best-known application of PSLQ in experimental mathematics is the 1995 discovery, by computer, of what is now known as the Bailey-Borwein-Plouffe (BBP) formula for π:

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

Remarkably, this formula lets us calculate binary or hexadecimal digits beginning at the nth digit, without needing to calculate any of the first $n - 1$ digits, by using a simple scheme that requires very little memory and no multiple-precision arithmetic software.[16,19] The BBP formula is currently used in the g95 Fortran compiler as part of transcendental function evaluation software.

Since 1995, researchers have used the PSLQ-based computational approach to find—and then prove—numerous other formulas of this type.[16] Figure 2 shows a sampling of these discoveries, including a recent discovery regarding the integrals

$$\mathcal{J}_n = \int_{n\pi/60}^{(n+1)\pi/60} \log\left|\frac{\tan t + \sqrt{7}}{\tan t - \sqrt{7}}\right| dt. \qquad (3)$$

The $\overset{?}{=}$ notation in Figure 2 means that this "identity" has been discovered numerically and verified to 2,000-digit accuracy, but no formal proof is yet known.

These computer-discovered formulas have implications for the age-old question of whether (and why) the digits of constants such as π and log 2 appear statistically random.[16,20] They've also attracted considerable attention, including feature articles in *Scientific American* and *Science News*.[21,22] Moreover, this same line of investigation has led to a formal proof of normality (statistical randomness in a specific sense) for an uncountably infinite class of explicit real numbers. The simplest example of this class is the constant

$$\alpha_{2,3} = \sum_{n=1}^{\infty} \frac{1}{3^n 2^{3^n}}, \qquad (4)$$

which is 2-normal: every string of m binary digits appears, in the limit, with frequency 2^{-m}.[16,23] For an efficient pseudorandom number generator based on this constant's binary digits, visit http://crd.lbl.gov/~dhbailey/mpdist.

Identification of Constants in Quantum Field Theory

A few years ago, British physicist David Broadhurst found an application of high-precision PSLQ computations in quantum field theory.[24] In particular, he found that by using PSLQ computations in each of 10 cases with unit or zero mass, the finite part of the scalar three-loop tetrahedral vacuum Feynman diagram reduces to four-letter "words" that represent iterated integrals in an alphabet of seven "letters"—namely, the one-forms $\Omega := d_x/x$ and $\omega_k := d_x/(\lambda^{-k} - x)$, where $\lambda := (\,1\sqrt{-3}\,)/2$ is the primitive sixth root of unity, and k runs from 0 to 5. In this context, a four-letter word is a four-dimensional iterated integral, such as

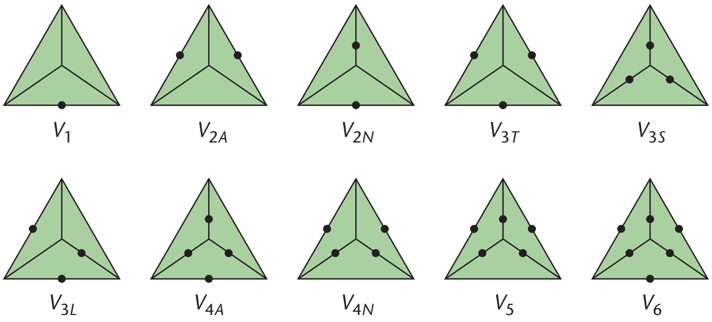

Figure 3. The 10 Feynmann diagrams for the 10 tetrahedral cases.

$$\begin{aligned} U &:= \varsigma\left(\Omega^2\omega_3\omega_0\right) \\ &= \int_0^1 \frac{dx_1}{x_1}\int_0^{x_1}\frac{dx_2}{x_2}\int_0^{x_2}\frac{dx_3}{(-1-x_3)}\int_0^{x_3}\frac{dx_4}{(1-x_4)} \\ &= \sum_{j>k>0}\frac{(-1)^{j+k}}{j^3 k} \end{aligned} \qquad (5)$$

$$V := \mathrm{Re}[\zeta(\Omega^2\omega_3\omega_1)] = \sum_{j>k>0}\frac{(-1)^j\cos(2\pi k/3)}{j^3 k}. \qquad (6)$$

The remaining terms in the diagrams reduce to products of constants found in Feynman diagrams with fewer loops. Figure 3 shows these 10 cases; in the diagrams, dots indicate particles with nonzero rest mass. Figure 4 gives the formulas for the corresponding constants that researchers have found by using PSLQ. Here, the constant $C = \sum_{k>0} \sin(\pi k/3)/k^2$.

In each case, these formulas were later proven to be correct, but until the experimental results of the computations were found, no one had any solid reason to believe that such relations might exist. Indeed, the adjective "experimental" is quite appropriate here, because the PSLQ computations played a role entirely analogous to (and just as crucial as) a conventional laboratory experiment.

$$
\begin{aligned}
V_1 &= 6\varsigma(3) + 3\varsigma(4)\\
V_{2A} &= 6\varsigma(3) - 5\varsigma(4)\\
V_{2N} &= 6\varsigma(3) - \frac{13}{2}\varsigma(4) - 8U\\
V_{3T} &= 6\varsigma(3) - 9\varsigma(4)\\
V_{3S} &= 6\varsigma(3) - \frac{11}{2}\varsigma(4) - 4C^2\\
V_{3L} &= 6\varsigma(3) - \frac{15}{4}\varsigma(4) - 6C^2\\
V_{4A} &= 6\varsigma(3) - \frac{77}{2}\varsigma(4) - 6C^2\\
V_{4N} &= 6\varsigma(3) - 14\varsigma(4) - 16U\\
V_5 &= 6\varsigma(3) - 13\varsigma(4) - 8U - 4C^2
\end{aligned}
$$

Figure 4. Formulas found by PSLQ for the 10 cases in Figure 3.

We've just reviewed a brief survey of the rapidly expanding usage of high-precision arithmetic in modern scientific computing, but it's worth noting that all these examples have arisen in the past 10 years. We could be witnessing the birth of a new era of scientific computing, in which the numerical precision required for a computation is as important to program design as algorithms and data structures.

In one respect, perhaps it's not a coincidence that interest in high-precision arithmetic has occurred during the same period that a concerted attempt has been made to implement many scientific computations on highly parallel and distributed systems. These systems have made possible much larger-scale runs than before, greatly magnifying numerical difficulties. Even single-processor system memory and performance have greatly expanded (a gift of Moore's law), enabling much larger computations than were feasible only a few years ago.

Other computations being performed on these systems today could have significant numerical difficulties, but those who use these codes might not yet realize the extent of these difficulties. With the advent of relatively painless software to convert programs to use high-precision arithmetic, researchers can periodically run a large scientific computation with double-double or higher-precision arithmetic and determine how accurate its results really are. In fact, the software tools currently available for high-precision arithmetic can be used in every stage of numerical error management:

- periodic testing of a scientific code to see if it's becoming numerically sensitive;
- determining how many digits in the results (intermediate or final) are reliable;
- pinning down a numerical difficulty to a few lines of code; and
- rectifying the numerical difficulties that are uncovered.

Although the software described here and elsewhere will help scientists in the short run, it is essential in the longer run that computer vendors recognize the need to provide both hardware *and* software support for high-precision arithmetic. An IEEE committee is updating the IEEE-754 floating-point arithmetic standard, and 128-bit support is one of the key provisions of their draft standard. This same committee is also considering establishing standards for arbitrary-precision arithmetic. These are all welcome and timely developments.

Acknowledgments

The author acknowledges the contributions of numerous colleagues in the preparation of this article, including Ed Baron, Ben Barrowes, Jonathan Borwein, Richard Crandall, Chris Ding, Alexei Frolov, William Kahan, Peter Nugent, Michael Strayer, and Zong-Chao Yan. Bailey's work was supported by the director of the Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the US Department of Energy, under contract number DE-AC03-76SF00098.

References

1. R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*, Springer-Verlag, 2001.
2. R.P. Brent, "A Fortran Multiple-Precision Arithmetic Package," *ACM Trans. Mathematical Software*, vol. 4, no. 1, 1978, pp. 57–70.
3. D.H. Bailey and X.S. Li, "A Comparison of Three High-Precision Quadrature Schemes," Computational Research Division, Berkeley Lab Computing Sciences, 2004; http://crd.lbl.gov/~dhbailey/dhbpapers/quadrature.pdf.
4. D.H. Bailey and S. Robins, "Highly Parallel, High-Precision Numerical Integration," Computational Research Division, Berkeley Lab Computing Sciences, 2004; http://crd.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf.
5. Y. He and C. Ding, "Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications," *J. Supercomputing*, vol. 18, no. 3, 2001, pp. 259–277.
6. P.H. Hauschildt and E. Baron, "The Numerical Solution of the Expanding Stellar Atmosphere Problem," *J. Computational and Applied Mathematics*, vol. 109, 1999, pp. 41–63.
7. D.H. Bailey and A.M. Frolov, "Universal Variational Expansion for High-Precision Bound-State Calculations in Three-Body Systems: Applications to Weakly-Bound, Adiabatic and Two-Shell Cluster Systems," *J. Physics B*, vol. 35, no. 20, 2002, pp. 4287–4298.
8. A.M. Frolov and D.H. Bailey, "Highly Accurate Evaluation of the

Few-Body Auxiliary Functions and Four-Body Integrals," *J. Physics B*, vol. 36, no. 9, 2003, pp. 1857–1867.

9. Z.-C. Yan and G.W.F. Drake, "Bethe Logarithm and QED Shift for Lithium," *Physical Rev. Letters*, vol. 81, 12 Sept. 2003, pp. 774–777.
10. T. Zhang, Z.-C. Yan, and G.W.F. Drake, "QED Corrections of $O(mc^2\alpha^7\ln\alpha)$ to the Fine Structure Splittings of Helium and He-Like Ions," *Physical Rev. Letters*, vol. 77, no. 26, 1994, pp. 1715–1718.
11. B.E. Barrowes et al., "Asymptotic Expansions of the Prolate Angular Spheroidal Wave Function for Complex Size Parameter," to appear in *Studies in Applied Mathematics*, 2005.
12. R.E. Caflisch, "Singularity Formation for Complex Solutions of the 3D Incompressible Euler Equations," *Physica D*, vol. 67, no. 1, 1993, pp. 1–18.
13. D.H. Bailey, R. Krasny, and R. Pelz, "Multiple Precision, Multiple Processor Vortex Sheet Roll-Up Computation," *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, SIAM Press, 1993, pp. 52–56.
14. W. Kahan and J. Darcy, "How Java's Floating-Point Hurts Everyone Everywhere," 1998; www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf.
15. D.H. Bailey, "Integer Relation Detection," *Computing in Science & Eng.*, vol. 2, no. 1, 2000, pp. 24–28.
16. J.M. Borwein and D.H. Bailey, *Mathematics by Experiment: Plausible Reasoning in the 21st Century*, AK Peters, 2003.
17. D.H. Bailey and D. Broadhurst, "Parallel Integer Relation Detection: Techniques and Applications," *Mathematics of Computation*, vol. 70, no. 236, 2000, pp. 1719–1736.
18. D.H. Bailey and D.J. Broadhurst, "A Seventeenth-Order Polylogarithm Ladder," Computational Research Division, Berkeley Lab Computing Sciences, 1999; http://crd.lbl.gov/~dhbailey/dhbpapers/ladder.pdf.
19. D.H. Bailey, P.B. Borwein, and S. Plouffe, "On the Rapid Computation of Various Polylogarithmic Constants," *Mathematics of Computation*, vol. 66, no. 218, 1997, pp. 903–913.
20. D.H. Bailey and R.E. Crandall, "On the Random Character of Fundamental Constant Expansions," *Experimental Mathematics*, vol. 10, no. 2, 2001, pp. 175–190.
21. W. Gibbs, "A Digital Slice of Pi," *Scientific Am.*, vol. 288, no. 5, 2003, pp. 23–24.
22. E. Klarreich, "Math Lab: How Computer Experiments Are Transforming Mathematics," *Science News*, vol. 165, 24 Apr. 2004, pp. 266–268.
23. D.H. Bailey and R.E. Crandall, "Random Generators and Normal Numbers," *Experimental Mathematics*, vol. 11, no. 4, 2004, pp. 527–546.
24. D.J. Broadhurst, "Massive Three-Loop Feynman Diagrams Reducible to SC* Primitives of Algebras of the Sixth Root of Unity," to appear in *European Physical J. C*, 2005; http://xxx.lanl.gov/abs/hep-th/9803091.

David H. Bailey is chief technologist of the Computational Research Department at Lawrence Berkeley National Laboratory. In 1993, he received the IEEE Computer Society's Sidney Fernbach Award for contributions to numerical computational science including innovation algorithms for FFTs, matrix multiply, and multiple-precision arithmetic on vector computer architecture. Bailey received a BS in mathematics from Brigham Young University and a PhD in mathematics from Stanford University. Contact him at dhbailey@llbl.gov.