High-Precision Floating-Point Arithmetic in Scientific Computation

David H. Bailey

28 January 2005

Abstract

At the present time, IEEE 64-bit floating-point arithmetic is sufficiently accurate for most scientific applications. However, for a rapidly growing body of important scientific computing applications, a higher level of numeric precision is required: some of these applications require roughly twice this level; others require four times; while still others require hundreds or more digits to obtain numerically meaningful results. Such calculations have been facilitated by new high-precision software packages that include high-level language translation modules to minimize the conversion effort. These activities have yielded a number of interesting new scientific results in fields as diverse as quantum theory, climate modeling and experimental mathematics, a few of which are described in this article. Such developments suggest that in the future, the numeric precision used for a scientific computation may be as important to the program design as are the algorithms and data structures.

Keywords: high-precision arithmetic, experimental mathematics, climate modeling, quantum theory, computational chemistry, computational physics

Acknowledgement:

This work was supported by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC03-76SF00098.

1. Introduction

Virtually all present-day computer systems, from personal computers 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, and for some applications, such as routine processing of experimental data, even the 32-bit standard often provides sufficient accuracy.

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

Some difficulties of this type may be remedied by changing the underlying algorithms or changing the order in which certain operations are performed. But in other cases, such fixes are not effective, and significantly higher numeric precision is required from the start. For some of these problems, twice the normal level, or roughly 32-digit accuracy, is sufficient. In other cases, four times the normal level is required, or roughly 64-digit accuracy. In still other cases, including applications from physics and mathematics, a much higher precision level is required, ranging from hundreds to thousands of digits.

It is important to keep in mind here that very few of the scientists and engineers who are currently involved in technical computing have rigorous backgrounds in numerical analysis. What's more, this scarcity of numerical expertise is likely to worsen, not improve, in the future, in part because of the dearth of students and young researchers who are interested in numerical mathematics [22]. Thus, while some may argue that numerically sensitive calculations can be remedied by using different algorithms or coding techniques, in practice such changes are highly prone to error and very expensive in human cost. In other words, it is usually easier, cheaper and more reliable to employ high-precision arithmetic to overcome these difficulties, even if other remedies are theoretically feasible.

This article will focus on high-precision floating-point rather than integer arithmetic. It should be noted, however, that high-precision (i.e., multi-word) integer arithmetic is a very interesting arena all by itself, with numerous applications in both pure and applied mathematics. For example, when one logs into a secure website to purchase a book or computer accessory, at some point the browser software is performing 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. An excellent reference here is the recently published book by Crandall and Pomerance [17]. *Mathematica* implementations of the algorithms described in the Crandall-Pomerance book can be obtained from `http://www.perfsci.com/software.htm`.

2. High-Precision Software

Software packages that perform high-precision floating-point arithmetic have been available since the early days of computing. For example, a package written by Richard Brent has been in use since the 1970s [14]. However, many of these packages require one 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, high-precision software packages have been produced that include high-level language interfaces, making such conversions relatively painless. These packages typically utilize custom datatypes 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*. These products incorporate arbitrary-precision arithmetic in a very natural way, extending to a wide range of the advanced functions that arise in pure and applied mathematics. These packages are very useful in some contexts, but they are generally not as fast as specialized high-precision arithmetic packages, and they do not provide a means to convert existing scientific programs, written say in Fortran-90 or C++, to utilize their high-precision arithmetic facilities.

Some examples of high-precision arithmetic software packages that are freely available on the Internet are the following, listed in alphabetical order. The ARPREC, DDFUN, QD and MPFUN90 packages are available from the author's website: `http://crd.lbl.gov/~dhbailey/mpdist` or from `http://www.expmath.info`.

- ARPREC. This package 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 are available for C++ and Fortran-90, supporting real, integer and complex datatypes. This software was written by the author, Xiaoye S. Li and Brandon Thompson.

- DDFUN. This packages provide double-double (approx. 31 digits) arithmetic, in an all-Fortran-90 environment. A high-level language interface supports real, integer and complex datatypes. This package is much faster than using arbitrary precision or quad-double software in applications where 31 digits is sufficient, and often faster than using the `real*16` datatype available in some Fortran systems.

- FMLIB and FMZM90. FMLIB is a low-level multiple-precision library, and FMZM90 provides high-level Fortran-90 language interfaces for real, integer and complex datatypes. This software was written by David M. Smith of Loyola Marymount University, and is available at `http://myweb.lmu.edu/dmsmith/FMLIB.html`.

- GMP. This package includes an extensive library of routines to support high-precision integer, rational and floating-point calculations. GMP has been produced by a volunteer effort and is distributed under the GNU license by the Free Software Foundation, and is available at `http://www.swox.com/gmp`.

- Predicates. This is a high-precision package specifically designed for performing the numerically sensitive operations that often arise in computational geometry. This software was written by Jonathan Shewchuk of U.C. Berkeley, and is available at `http://www-2.cs.cmu.edu/~quake/robust.html`.

- QD. This package includes routines to perform "double-double" (approx. 31 digits) and "quad-double" (approx. 62 digits) arithmetic. High-level language interfaces are available for C++ and Fortran-90, supporting real, integer and complex datatypes. The QD package is much faster than using arbitrary precision software in applications where 31 or 62 digits is sufficient. This software was written by the author, Xiaoye S. Li and Yozo Hida.

- MPFR. 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. Additional information is available at `http://www.mpfr.org`.

- MPFR++. This is a high-level C++ interface to MPFR. Additional information is available at `http://perso.ens-lyon.fr/nathalie.revol/software.html`.

- MPFUN90. This is equivalent to ARPREC in user-level functionality, but is written entirely in Fortran-90 and provides a Fortran-90 language interface.

- VPA. This package provides an arbitrary precision functionality together with a Fortran language interface. The software was written by J.L. Schonfelder of N.A. Software of the U.K., and is available at `http://pcwww.liv.ac.uk/ jls/vpa20.htm`.

Several of these packages includes sample application programs. For example, the ARPREC and MPFUN90 packages include programs that implement the "PSLQ" algorithm for integer relation detection (see Sections 10 and 11). The ARPREC, MPFUN90 and QD packages include programs to evaluate definite integrals to high precision. Along this line, it is now possible to evaluate many integrals that arise in mathematics or physics to very high accuracy, even in cases where 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 is often possible to obtain analytic evaluations of definite integrals that otherwise have no known solution [9, 10].

Using high-precision software definitely increases computer run times, compared with using conventional machine precision. For example, computations using the double-double precision typically run five times longer than with ordinary 64-bit arithmetic. This figure rises to 25 times for the 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 1000 digits, after which the cost increases as roughly $p \log p$ (presuming one is using FFT-based multiplication, which is available for instance in ARPREC and MPFUN90). Fortunately, it is often not necessary to employ high-precision

arithmetic for the entire calculation. Often just one critical loop is all that requires this higher accuracy.

We will now describe some of the scientific applications that utilize high-precision arithmetic.

3. Climate Modeling

It is well-known that weather or climate simulations are fundamentally chaotic—if microscopic changes are made to the present state, within a certain period of simulated time the future state is completely different. Indeed, ensembles of these calculations are required to obtain statistical confidence in global climate trends produced from such calculations. As a result, computational scientists involved in climate modeling applications have long resigned themselves 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 is not only difficult for researchers to compare results, but it is often problematic even to determine whether they have correctly deployed their code on a given system.

Recently Helen He and Chris Ding, two researchers at LBNL, investigated this non-reproducibility phenomenon in a widely-used climate modeling code. They found that almost all of 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 above) for these loops. This single change dramatically reduced the numerical variability of the entire application, permitting computer runs to be compared for much longer run times than before. Details of their work can be read in [21].

In retrospect, it is not clear that handling these sums in this manner is the best solution—there may be other ways to preserve meaningful numerical accuracy, while at the same time yielding reproducible results. Such phenomena deserve further study and refinement and are being investigated. But in the meantime, He and Ding's solution is a straightforward and effective means of dealing with this problem.

4. Supernova Simulations

Recently Edward Baron, Peter Hauschildt, and Peter Nugent used the QD package, which provides double-double (128-bit or 31-digit) and quad-double (256-bit or 62-digit) datatypes, to solve for the non-local thermodynamic equilibrium populations of iron and other atoms in the atmospheres of supernovae and other astrophysical objects [11, 20]. Iron for example may exist as Fe II in the outer parts of the atmosphere, but in the inner parts Fe IV or Fe V could be dominant. Introducing artificial cutoffs leads to numerical glitches, so it is necessary to solve for all of these populations simultaneously. Since 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 very large.

In order to handle this potentially very large dynamic range, yet at the same time perform the computation in reasonable time, Baron, Hauschildt and Nugent employ a dynamic scheme to determine whether to use 64-bit, 128-bit or 256-bit arithmetic in both

constructing the matrix elements and in solving the linear system. The run time of the code that uses 256-bit arithmetic has emerged as a large fraction of the total run time, so plans are being drawn up for an even faster version of the 256-bit routines to reduce these run times.

We should note here that numerical difficulties in dealing with quantum calculations of atomic species have been recognized for many years. Michael Strayer, now a senior program manager at the U.S. Department of Energy, recalls utilizing a variable-length precision facility in the IBM 604 for similar calculations back in the late 1960s [26].

5. Coulomb N-Body Atomic System Simulations

Numerous computations have been performed recently using high-precision arithmetic to study atomic-level Coulomb systems. For example, Alexei Frolov of Queen's University in Ontario, Canada has 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 $5,000 \times 5,000$ in size) and very nearly degenerate. Until recently, progress in this arena was severely hampered by the numerical difficulties induced by these nearly degenerate matrices.

Frolov has done his calculations using the MPFUN package, with a numeric precision level exceeding 100 digits. Frolov notes 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 has 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.

As of this date, Frolov has written a total of 21 papers based on high-precision computations. Two illustrative examples are [7] and [18].

6. Studies of the Fine Structure Constant of Physics

In the past few years, significant progress has been achieved in using high-precision arithmetic to obtain highly accurate solutions to the Schrodinger equation for the lithium atom. In particular, the nonrelativistic ground state energy has been calculated 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 wavefunctions, Zong-Chao Yan and others have been able to test the relativistic and QED effects at the 50 parts per million (ppm) level and also at the one ppm level [27]. Along this line, a number of properties of lithium and lithium-like ions have also been calculated, 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 stage that highly accurate experiments are now planned. When some additional computations are completed, a unique atomic physics value of the fine structure constant may be obtained to an accuracy of 16 parts per billion [28].

7. Electromagnetic Scattering Theory

A key operation in computational studies of electromagnetic scattering is to find the branch points of the asymptotic expansion of the spheroidal wave function. Newton-

Raphson iterations are typically used here. Unfortunately, the accuracy of the results obtained using conventional 64-bit arithmetic for these Newton iterations significantly limits the range of the wave functions that can be studied.

Recently Ben Barrowes used the MPFUN package to greatly extend the range of wavefunctions that can be studied [12]. This 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 this work, Barrowes found that arbitrary-precision versions of the Fortran-90 array operations were required. He then worked with the author to implement these array functions, which are now available as an optional part of the MPFUN90 package.

8. Vortex Sheet Roll-Up Simulations

High-precision arithmetic has been exploited for some time in attempts to resolve complex phenomena associated with fluid flows. For example, in 1993 Russel Caflisch used 32, 64 and even 128-digit arithmetic in studies of singularity formation under three-dimensional incompressible Euler flows [16].

A long-standing problem of fluid dynamics is whether a fluid undergoing vortex sheet roll-up assumes a true power law spiral. Recently this question was investigated using the QD and MPFUN90 packages. As mentioned in Section 2, the computational cost is multiplied by a large factor by employing these packages. For this problem, the resulting calculation, which is fairly long-running even with standard precision, becomes prohibitively expensive on a single processor system with high-precision arithmetic. However, researchers (Robert Krasny, Richard Pelz and the author) were able to implement the calculation on a highly parallel system, using 64 processors. The calculation ultimately used 10,000 CPU-hours. The result of this calculation showed that when a critical parameter is reduced below 0.02, the solution is no longer a smooth spiral, but instead develops "blisters" (see Figure 1) that are not merely artifacts of insufficient precision. At the present time, these "blisters" are not well understood, and further investigation in under way [8].

9. Computational Geometry and Grid Generation

Grid generation, contour mapping and several other computational geometry applications crucially rely on highly accurate arithmetic. This is because small numerical errors can lead to geometrically inconsistent results. Such difficulties are inherent in the mathematics of the formulas commonly used in such calculations and cannot be easily remedied. For example, William Kahan and Joseph Darcy have shown that small numerical errors in the computation of the point nearest to a given point on a line of intersection of 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 [23].

Two commonly used computational geometry operations are the orientation test and the incircle 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, an incircle test determines whether a point lies inside, outside, or on a

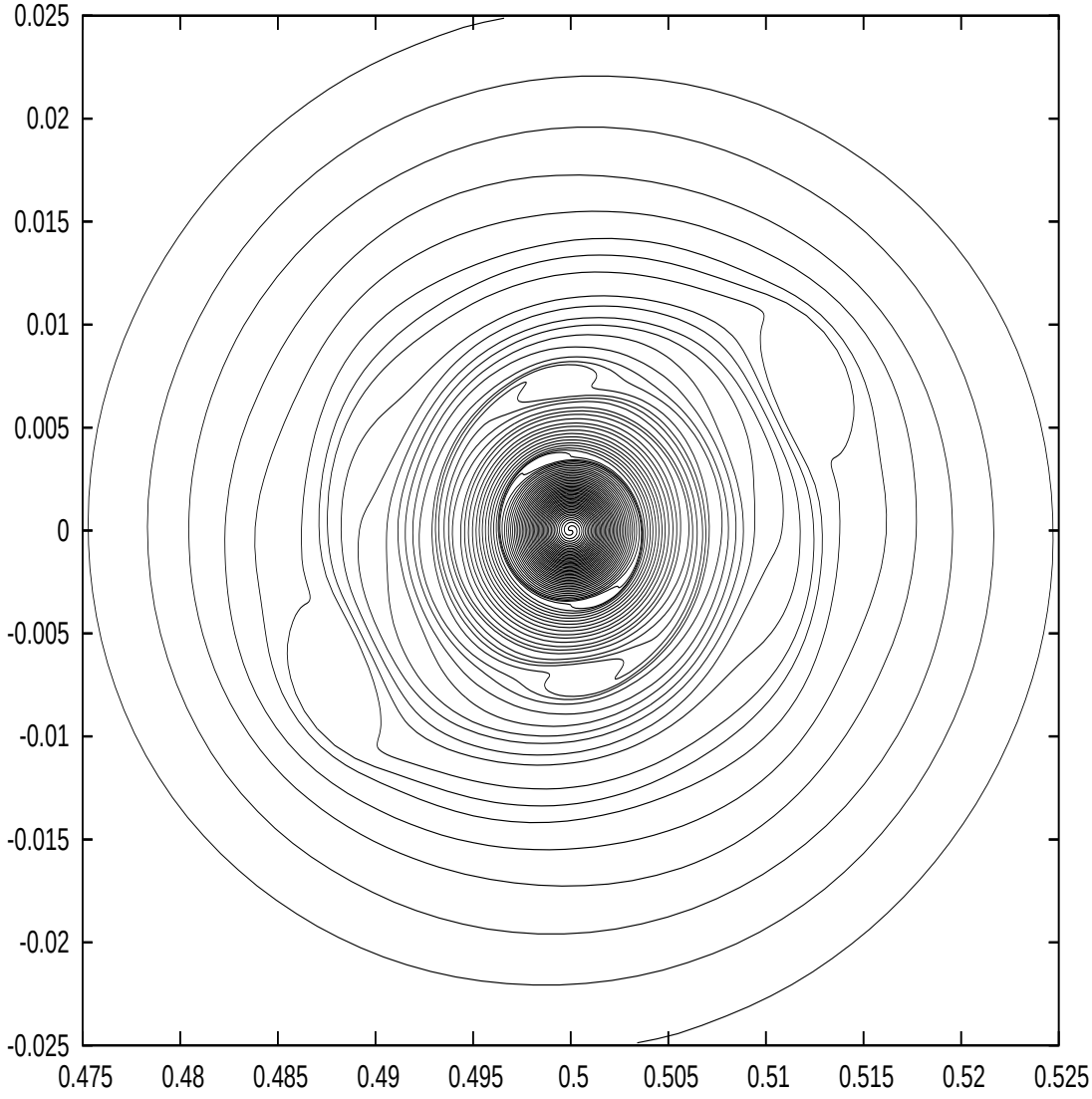

Figure 1: Vortex roll-up computed using quad-double arithmetic. The "blisters" are not merely artifacts of insufficient precision.

circle defined by other points. Each of these tests is typically performed by evaluating the sign of a determinant that is expressed in terms of the coordinates of the points. If these coordinates are expressed as single or double precision floating-point numbers, roundoff error may lead to an incorrect result when the true determinant is near zero. In turn, this misinformation can lead an application to fail or produce incorrect results, as noted above.

To remedy such problems, Jonathan Shewchuk has produced a software package that performs "adaptive" floating-point arithmetic, which dynamically increases numeric precision until an unambiguous result is obtained. This software and some related information on this type of application is publicly available—see "Predicates" in the software list given in Section 2.

10. Computational Number Theory

One active field of current mathematical research, which has applications in fields such as cryptography, is computational number theory. 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 $2^{24036583} - 1$ (which has seven million decimal digits) number was recently proven prime via the Lucas–Lehmer test. This test requires very large-integer arithmetic, which in turn utilizes fast Fourier transforms (FFT) to perform multiplications, since these multiplications are merely large linear convolutions ([17]). As it turns out, the size of these convolutions that can be done reliably is limited by the numerical accuracy of IEEE arithmetic, unless one stores only a few words in each word of memory, thus wasting memory. Thus fast double-double arithmetic, for instance, has a significant advantage in these computations.

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

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

which among other things encodes profound information about the distribution of prime numbers. For example, the best method currently known for evaluating the classical function $\pi(x)$, the number of primes not exceeding x, involves calculations of $\zeta(3/2 + it)$. Along this line, Will Galway has observed that calculating zeta using IEEE 64-bit arithmetic only permits one to accurately compute $\pi(x)$ for x up to 10^{13} (see [17]). Thus, there is considerable interest in utilizing higher-precision floating-point arithmetic, which should permit researchers to press up toward the current frontier of knowledge namely $\pi(10^{22})$.

11. 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 [13][1]. 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 [3]. Integer relation computations require very high precision in the input vector to obtain numerically meaningful results. Computations with several hundred digits are typical, although in one extreme instance, 50,000-digit arithmetic was required to obtain the desired result [4].

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

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

This formula has the remarkable property that it permits one to calculate binary or hexadecimal digits beginning at the n-th digit, without needing to calculate any of the first $n-1$ digits, using a simple scheme that requires very little memory and no multiple-precision arithmetic software [2][13, pg. 135-143]. The BBP formula for π has recently found a practical application—it is now employed in the g95 Fortran compiler as part of transcendental function evaluation software.

Since 1995, numerous other formulas of this type have been found, using the PSLQ-based computational approach, and then subsequently proven [13, pg. 147–149]. A sampling of these discoveries is shown in Figure 2, including a recent discovery regarding the integrals

$$J_n \;=\; \int_{n\pi/60}^{(n+1)\pi/60} \log \left| \frac{\tan t + \sqrt{7}}{\tan t - \sqrt{7}} \right| dt$$

The $\stackrel{?}{=}$ 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.

Further, it has been found that 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 [5][13, pg. 163–174]. These developments have attracted considerable attention, including feature articles in *Scientific American* and *Science News* [19, 24]. This same line of investigation has further 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

$$\alpha_{2,3} \;=\; \sum_{n=1}^{\infty} \frac{1}{3^n 2^{3^n}},$$

which is 2-normal: every string of m binary digits appears, in the limit, with frequency 2^{-m} [6][13, pg. 174–178]. An efficient pseudorandom number generator, based on the binary digits of this constant, can be obtained from LBNL high-precision software site: `http://crd.lbl.gov/~dhbailey/mpdist`.

12. 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 [15]. In particular, he found, using

$$
\begin{aligned}
\pi\sqrt{3} &= \frac{9}{32}\sum_{k=0}^{\infty}\frac{1}{64^k}\left(\frac{16}{6k+1}-\frac{8}{6k+2}-\frac{2}{6k+4}-\frac{1}{6k+5}\right)\\
\pi^2 &= \frac{1}{8}\sum_{k=0}^{\infty}\frac{1}{64^k}\left[\frac{144}{(6k+1)^2}-\frac{216}{(6k+2)^2}-\frac{72}{(6k+3)^2}-\frac{54}{(6k+4)^2}+\frac{9}{(6k+5)^2}\right]\\
\pi^2 &= \frac{2}{27}\sum_{k=0}^{\infty}\frac{1}{729^k}\left[\frac{243}{(12k+1)^2}-\frac{405}{(12k+2)^2}-\frac{81}{(12k+4)^2}-\frac{27}{(12k+5)^2}\right.\\
&\quad \left.-\frac{72}{(12k+6)^2}-\frac{9}{(12k+7)^2}-\frac{9}{(12k+8)^2}-\frac{5}{(12k+10)^2}+\frac{1}{(12k+11)^2}\right]\\
\log 3 &= \frac{1}{729}\sum_{k=0}^{\infty}\frac{1}{729^k}\left(\frac{729}{6k+1}+\frac{81}{6k+2}+\frac{81}{6k+3}+\frac{9}{6k+4}+\frac{9}{6k+5}+\frac{1}{6k+6}\right)\\
\pi\log 2 &= \frac{1}{256}\sum_{k=0}^{\infty}\frac{1}{4096^k}\left[\frac{4096}{(24k+1)^2}-\frac{8192}{(24k+2)^2}-\frac{26112}{(24k+3)^2}+\frac{15360}{(24k+4)^2}\right.\\
&\quad -\frac{1024}{(24k+5)^2}+\frac{9984}{(24k+6)^2}+\frac{11520}{(24k+8)^2}+\frac{2368}{(24k+9)^2}-\frac{512}{(24k+10)^2}\\
&\quad +\frac{768}{(24k+12)^2}-\frac{64}{(24k+13)^2}+\frac{408}{(24k+15)^2}+\frac{720}{(24k+16)^2}\\
&\quad \left.+\frac{16}{(24k+17)^2}+\frac{196}{(24k+18)^2}+\frac{60}{(24k+20)^2}-\frac{37}{(24k+21)^2}\right]\\
0 &\stackrel{?}{=} -2J_2-2J_3-2J_4+2J_{10}+2J_{11}+3J_{12}+3J_{13}+J_{14}-J_{15}-J_{16}\\
&\quad -J_{17}-J_{18}-J_{19}+J_{20}+J_{21}-J_{22}-J_{23}+2J_{25}
\end{aligned}
$$

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

PSLQ computations, that in each of ten cases with unit or zero mass, the finite part the scalar 3-loop tetrahedral vacuum Feynman diagram reduces to 4-letter "words" that represent iterated integrals in an alphabet of 7 "letters," namely the one-forms $\Omega := dx/x$ and $\omega_k := dx/(\lambda^{-k} - x)$, where $\lambda := (1 + \sqrt{-3})/2$ is the primitive sixth root of unity, and k runs from 0 to 5. In this context, a 4-letter "word" is a 4-dimensional iterated integral, such as

$$U \;:=\; \zeta(\Omega^2\omega_3\omega_0) = \int_0^1 \frac{dx_1}{x_1} \int_0^{x_1} \frac{dx_2}{x_2} \int_0^{x_2} \frac{dx_3}{(-1-x_3)} \int_0^{x_3} \frac{dx_4}{(1-x_4)} = \sum_{j>k>0} \frac{(-1)^{j+k}}{j^3 k}$$

There are 7^4 four-letter words. Only two of these are primitive terms occurring in the 3-loop Feynman diagrams: U, above, and

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

The remaining terms in the diagrams reduce to products of constants found in Feynman diagrams with fewer loops. These ten cases as shown in Figure 3. In these diagrams, dots indicate particles with nonzero rest mass. The formulas that have been found, using PSLQ, for the corresponding constants are given in Figure 4. 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 a crucial as) a conventional laboratory experiment.

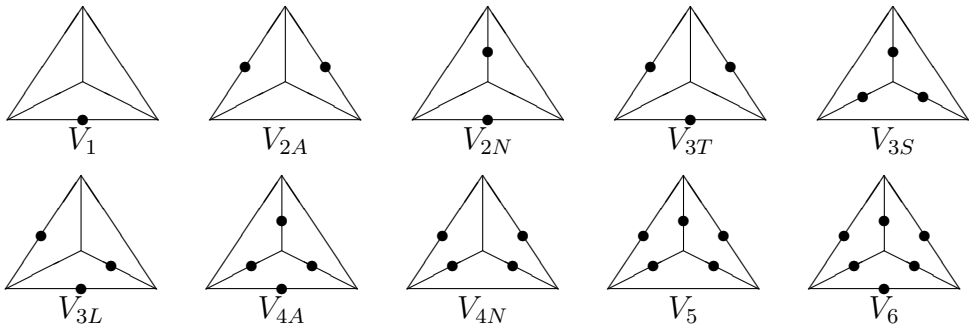

Figure 3: The ten tetrahedral cases

13. Conclusion

We have presented here a brief survey of the rapidly expanding usage of high-precision arithmetic in modern scientific computing. It is worth noting that all of these examples have arisen in the past ten years. Thus we may be witnessing the birth of a new era of scientific computing, in which the numerical precision required for a computation is as important to the program design as are the algorithms and data structures.

$$
\begin{array}{lcl}
V_1 & = & 6\zeta(3) + 3\zeta(4) \\
V_{2A} & = & 6\zeta(3) - 5\zeta(4) \\
V_{2N} & = & 6\zeta(3) - \frac{13}{2}\zeta(4) - 8U \\
V_{3T} & = & 6\zeta(3) - 9\zeta(4) \\
V_{3S} & = & 6\zeta(3) - \frac{11}{2}\zeta(4) - 4C^2 \\
V_{3L} & = & 6\zeta(3) - \frac{15}{4}\zeta(4) - 6C^2 \\
V_{4A} & = & 6\zeta(3) - \frac{77}{12}\zeta(4) - 6C^2 \\
V_{4N} & = & 6\zeta(3) - 14\zeta(4) - 16U \\
V_5 & = & 6\zeta(3) - \frac{469}{27}\zeta(4) + \frac{8}{3}C^2 - 16V \\
V_6 & = & 6\zeta(3) - 13\zeta(4) - 8U - 4C^2
\end{array}
$$

Figure 4: Formulas found by PSLQ for the ten cases of Figure 3

In one respect, perhaps it is not a coincidence that interest in high-precision arithmetic has arisen 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. Also, even on single-processor systems, memory and performance have greatly expanded (a gift of Moore's Law in semiconductor technology), enabling much larger computations than were feasible only a few years ago.

It may well be that other computations now being performed on these systems have significant numerical difficulties, but those who use these codes don't yet realize the extent of these difficulties. With the advent of relatively painless software to convert programs to use high-precision arithmetic, it is now possible to periodically run a large scientific computation with double-double or higher-precision arithmetic, in order to determine how accurate its results really are.

In fact, the software tools now available for high-precision arithmetic can be used in every stage of numerical error management: (1) periodic testing of a scientific code to see if it is becoming numerically sensitive; (2) determining how many digits in the results (intermediate or final) are reliable; (3) pinning down a numerical difficulty to a few lines of code; and (4) rectifying the numerical difficulties that are uncovered.

While software described in this paper and elsewhere will help scientists in the short run, in the longer run it is essential that computer vendors recognize the need to provide both hardware and software support for high-precision arithmetic. An IEEE committee has been formed for updating the IEEE-754 floating-point arithmetic standard; 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 welcome and timely developments.

Acknowledgements

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.

References

[1] David H. Bailey, "Integer Relation Detection," *Computing in Science and Engineering*, Jan-Feb., 2000, pg. 24–28.

[2] David H. Bailey, Peter B. Borwein, and Simon Plouffe, "On the Rapid Computation of Various Polylogarithmic Constants," *Mathematics of Computation*, vol. 66, no. 218 (Apr 1997), pg. 903–913.

[3] David H. Bailey and David Broadhurst, "Parallel Integer Relation Detection: Techniques and Applications," *Mathematics of Computation*, vol. 70, no. 236 (2000), pg. 1719–1736.

[4] David H. Bailey and David J. Broadhurst, "A Seventeenth-Order Polylogarithm Ladder," `http://crd.lbl.gov/~dhbailey/dhbpapers/ladder.pdf` (1999).

[5] David H. Bailey and Richard E. Crandall, "On the Random Character of Fundamental Constant Expansions," *Experimental Mathematics*, vol. 10, no. 2 (June 2001), pg. 175–190.

[6] David H. Bailey and Richard E. Crandall, "Random Generators and Normal Numbers," *Experimental Mathematics*, vol. 11, no. 4 (2004), pg. 527–546.

[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 Clouster Systems," *Journal of Physics B*, vol. 35, no. 20 (28 Oct 2002), pg. 42870–4298.

[8] David H. Bailey, R. Krasny and R. Pelz, "Multiple Precision, Multiple Processor Vortex Sheet Roll-Up Computation," *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, 1993, SIAM, Philadelphia, pg. 52–56.

[9] David H. Bailey and Xiaoye S. Li, "A Comparison of Three High-Precision Quadrature Schemes," manuscript available at `http://crd.lbl.gov/~dhbailey/dhbpapers/quadrature.pdf` (2004).

[10] David H. Bailey and Sinai Robins, "Highly Parallel, High-Precision Numerical Integration," manuscript available at `http://crd.lbl.gov/~dhbailey/dhbpapers/quadparallel.pdf` (2004).

[11] Edward Baron and Peter Nugent, personal communication, Nov. 2004.

[12] B. E. Barrowes, K. O'Neill, T. M. Grzegorczyk and J. A. Kong, "Asymptotic Expansions of the Prolate Angular Spheroidal Wave Function for Complex Size Parameter," *Studies in Applied Mathematics*, to appear.

[13] Jonathan M. Borwein and David H. Bailey, *Mathematics by Experiment: Plausible Reasoning in the 21st Century*.

[14] Richard P. Brent, "A Fortran Multiple-Precision Arithmetic Package," *ACM Transactions on Mathematical Software*, vol. 4, no. 1 (Mar 1978), pg. 57–70.

[15] David J. Broadhurst, "Massive 3-loop Feynman Diagrams Reducible to SC* Primitives of Algebras of the Sixth Root of Unity", preprint, March 1998, to appear in *European Physical Journal C*. Available from `http://xxx.lanl.gov/abs/hep-th/9803091`.

[16] Russel E. Caflisch, "Singularity Formation for Complex Solutions of the 3D Incompressible Euler Equations," *Physica D*, vol. 67, no. 1 (1993), pg. 1–18.

[17] Richard Crandall and Carl Pomerance, *Prime Numbers: A Computational Perspective*, Springer-Verlag, New York, 2001.

[18] 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.

[19] Wayt Gibbs, "A Digital Slice of Pi," *Scientific American*, vol. 288, no. 5 (May 2003), pg. 23–24.

[20] P. H. Hauschildt and E. Baron, "The Numerical Solution of the Expanding Stellar Atmosphere Problem," *Journal Computational and Applied Mathematics*, vol. 109 (1999), pg. 41–63.

[21] Yun He and Chris Ding, "Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications," *Journal of Supercomputing*, vol. 18, no. 3 (Mar 2001), pg 259–277.

[22] William Kahan, personal communication, Nov. 2004.

[23] William Kahan and Joseph Darcy, "How Java's Floating-Point Hurts Everyone Everywhere," available at `http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf`, 1998.

[24] Erica Klarreich, "Math Lab: How Computer Experiments Are Transforming Mathematics," *Science News*, vol. 165 (Apr. 24, 2004), pg. 266–268.

[25] Jonathan R. Shewchuk, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates," *Discrete and Computational Geometry*, vol. 18 (1997), pg. 305–363.

[26] Michael Strayer, personal communication, Nov. 2004.

[27] Z.-C. Yan and G. W. F. Drake, "Bethe Logarithm and QED Shift for Lithium," *Physics Review Letters*, vol. 81 (12 Sep 2003), pg. 774–777.

[28] T. Zhang, Z.-C. Yan and G. W. F. Drake, "QED Correctins of $O(mc^2\alpha^7 \ln \alpha)$ to the Fine Structure Splittings of Helium and He-Like Ions," *Physics Review Letters*, vol. 77, no. 26 (27 Jun 1994), pg. 1715–1718.