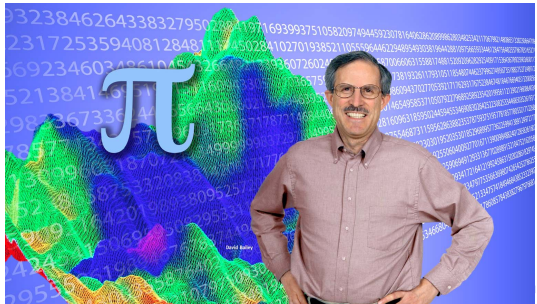
Petascale Computing and High-Precision Arithmetic: Applications and Challenges

David H. Bailey
http://www.davidhbailey.com
Lawrence Berkeley National Lab (retired) and University of California, Davis
Co-author: Jonathan M. Borwein (University of Newcastle, Australia)



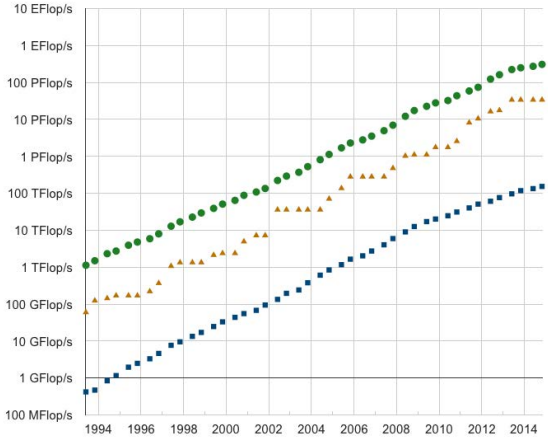
Petascale supercomputers: Measured in quadrillions (10^{15})

The memory capacity of a supercomputer is measured in "Pbytes" or "petabytes" (one Pbyte = one quadrillion bytes).

The performance of a supercomputer is measured in "Pflop/s" or "petaflops" (one Pflop/s = one quadrillion 64-bit floating-point operations per second).

- The volume of Lake Zurich is 3.9×10^{12} litres, or roughly $1/256$ quadrillion litres.
- The volume of Lake Tahoe, California is 1.51×10^{14} litres, or roughly $1/7$ quadrillion litres.
- The distance to Alpha Centauri is 4.15×10^{16} metres, or 41.5 quadrillion metres.
- The time elapsed since the big bang is 7.24×10^{15} minutes, or 7.24 quadrillion minutes.

Increasing performance of the top 500 supercomputers (1994 – present)



Orange: #1
Blue: #500
Green: Sum of #1 thru #500

- "Performance development," Top500.org, available at `http://top500.org`.

Systems at the Lawrence Berkeley National Laboratory's NERSC facility

Current system ("Edison"):

- 5576 Intel "Ivy Bridge" nodes.
- Memory per node: 64 Gbyte.
- Performance per node: 460.8 Gflop/s.
- Total main memory: 357 Tbyte.
- Total performance: 2.57 Pflop/s.
- Total disk storage: 7.56 Pbyte.

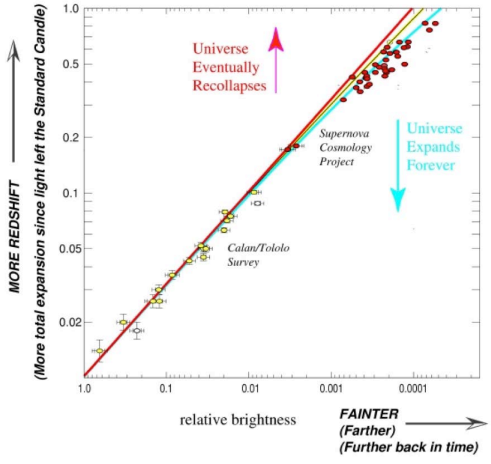
To be installed in 2016 ("Cori"):

- 9300 Intel "Knight's Landing" nodes.
- Main memory per node: 64 Gbyte.
- Performance per node: 3 Tflop/s.
- Total main memory: 595 Tbyte.
- Total peak performance: 28 Pflop/s.
- Total disk capacity: 28 Pbyte.

The accelerating universe

In 1998, two teams of astronomers (one led by Saul Perlmutter of LBNL, and the other led by Brian P. Schmidt of Australian National University), came to the paradoxical conclusion that the expansion of the universe is accelerating, not slowing down.

Both teams based their results on careful measurements of distant supernovas, which in turn were found by sifting through reams of digital telescope data. The U.S. team in particular relied heavily on large computer systems at the NERSC facility, coupled with a worldwide network of collaborating astronomers.

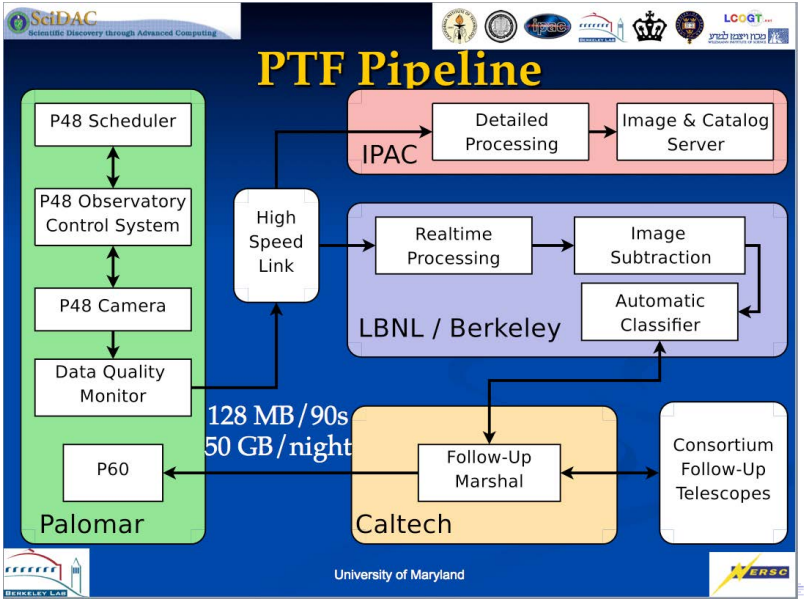


Discovery of supernova in the Pinwheel Galaxy

In 2011, Peter Nugent of LBNL, working within a worldwide consortium of fellow astronomers, discovered a Type Ia supernova in the Pinwheel Galaxy, which is "only" 21 million light-years from earth. It is the closest and brightest supernova of this type seen in the last 30 years.

Nugent and his team utilized the Palomar Transient Facility (PTF), a robot-controlled telescope that produces digital images, plus a worldwide network of data processing and storage facilities.

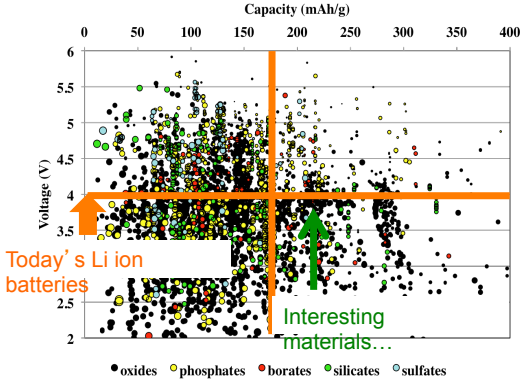
The Palomar Transient Facility (PDF) data pipeline



The Materials Project

Researchers at LBNL and Harvard University started the "Materials Project":

- Reduce lag time between materials science advances and real-world commercialization.
- Invert the conventional materials science paradigm: Ask "What properties do I want?," then "Which materials have them?"
- Method: Perform *ab initio* calculations of tens of thousands of potentially interesting compounds, then save in searchable database.
- Now in operation: `http://www.materialsproject.org`.



Voltage vs. capacity for over 20,000 Li-ion cathode compounds using high-throughput *ab initio* methods.

DANGER AHEAD: Reproducibility in scientific computing

A December 2012 workshop on reproducibility in computing, held at Brown University in Rhode Island, USA, identified these issues:

- The need to carefully document the full context of computational experiments—system environment, input data, code used, computed results, etc.
- The need to save the code and output data in a permanent repository.
- The need for reviewers, research institutions and funding agencies to recognize the importance of computing and computing professionals, and to allocate funding for after-the-grant support and repositories.
- The need to encourage publication of negative results—other researchers can often learn from them.
- The need to ensure responsible reporting of performance.
- The increasing importance of numerical reproducibility, and the need for tools to ensure and enhance numerical reliability.

- 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, `http://www.davidhbailey.com/dhbpapers/icerm-report.pdf`.

Reproducibility in biomedicine

The biomedical field has been stung by numerous cases where pharma products look good based on clinical trials, but later disappoint in real-world usage, or the results cannot be reproduced in separate studies. Examples:

- In 2004, GlaxoSmithKline acknowledged that while some trials of Paxil found it effective for depression in children, other unpublished studies showed no benefit.
- In 2011, Bayer researchers reported that they were able to reproduce the results of only 17 of 67 published studies they examined.
- In 2012, Amgen researchers reported that they were able to reproduce the results of only 6 of 53 published cancer studies.
- In 2014, a review of Tamiflu found that while it made flu symptoms disappear a bit sooner, it did not stop serious complications or keep people out of the hospital.

These experiences have exposed a fundamental flaw in methodology:
Only publicizing the results of successful trials introduces a bias into the results.

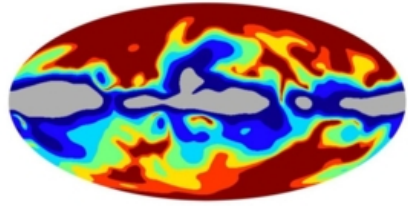
The AllTrials movement would require all results to be public: `http://www.alltrials.net`

Reproducibility in physics

In March 2014, a team of researchers from Harvard University made the dramatic announcement that they had discovered an interesting "twisting" pattern in cosmic microwave background data, measured using their BICEP2 experimental system.

This pattern fit very well with the hypothesized pattern of the most commonly assumed model of "inflation" in the first tiny fraction of a second after the big bang, and thus has been trumpeted as the first experimental evidence of the inflationary cosmology.

But other researchers had difficulty reconstructing the claimed results. Finally, two teams challenged the BICEP2 findings, saying that the results could more readily be explained by dust in the Milky Way.



- Ron Cowen, "Doubt grows about gravitational waves detection," *Scientific American*, 2 Jun 2014.

Numerical reproducibility

The ICERM reproducibility report noted:

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

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

Numerical reproducibility, continued

Particularly vulnerable are:

1. Large-scale, highly parallel simulations, running on systems with hundreds of thousands or millions of processors — numerical sensitivities are greatly magnified.
2. Certain applications with highly ill-conditioned linear systems.
3. Large summations, especially those involving cancellations.
4. Long-time, iterative simulations (such as molecular dynamics or climate models).
5. Computations to resolve small-scale phenomena.
6. Studies in computational physics or experimental mathematics often require huge precision levels.

Analysis of collisions at the Large Hadron Collider

- 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: 5 millions line of C++ and python code, developed by roughly 2000 physicists and engineers over 15 years.
- Recently, in an attempt to speed up the calculation, researchers found that merely changing the underlying math library resulted in some collisions being missed or misidentified.

Questions:

- How serious are these numerical difficulties?
- How can they be tracked down?
- How can the library be maintained, producing numerically reliable results?

Numerical analysis expertise among U.C. Berkeley graduates

Of the 2010 U.C. Berkeley graduating class, 870 were in disciplines likely to require technical computing:

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

Other fields (not counted) that will likely involve significant computing:

- Biology, geology, medicine, economics, psychology, sociology.

Enrollment in numerical analysis courses:

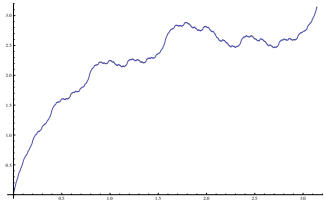
- Math 128A (Introductory numerical analysis required of math majors): 219.
- Math 128B (A more advanced course, required to do serious work): 24.

Conclusion: Fewer than 2% of Berkeley graduates who will do technical computing have had rigorous training in numerical analysis!

Enhancing reproducibility with high-precision arithmetic

Problem: Find the arc length of the irregular function $g(x) = x + \sum_{0 \le k \le 10} 2^{-k} \sin(2^k x)$, over the interval $(0, \pi)$ (using 10^7 abscissa points).

- If this computation is done with ordinary double precision arithmetic, the calculation takes 2.59 seconds and yields the result 7.073157029008510.
- If it is done using all double-double arithmetic (31-digit accuracy), it takes 47.39 seconds seconds and yields the result 7.073157029007832.
- But if only the summation is changed to double-double, the result is identical to the double-double result (to 15 digits), yet the computation only takes 3.47 seconds.



Graph of $g(x) = x + \sum_{0 \le k \le 10} 2^{-k} \sin(2^k x)$, over $(0, \pi)$.

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

Aren't 64 bits enough?

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

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

A 64-bit computation (e.g., using Matlab, Linpack or LAPACK) fails to find the correct polynomial in this instance, even if one rounds results to nearest integer.

However, if Linpack routines are converted to use double-double arithmetic (31-digit accuracy), the above computation quickly produces the correct polynomial:

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

Aren't 64 bits enough? continued

The result on the previous page can be obtained with double precision using Lagrange interpolation or the Demmel-Koev algorithm. But few scientists, outside of expert numerical analysts, are aware of these schemes—most people use home-grown code.

Besides, even these schemes fail for higher-degree problems. For example: $(1, 134217731, 8589938753, 97845255883, 549772595201, 2097396156251, 6264239146561, 15804422886323, 35253091827713, 71611233653971, 135217729000001, 240913322581691, 409688091758593)$ is generated by:

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

In contrast, a straightforward Linpack scheme, implemented with double-double arithmetic, works fine for this and a wide range of similar problems.

Techniques for high-precision computation

- Data is stored as a string of ints or floats, with initial words holding the string length and binary exponent.
- For modest precision levels (< 1000 digits):
  1. Use conventional grade-school schemes for basic arithmetic.
  2. Use conventional Taylor series schemes for transcendentals.
- For extra-high precision levels (> 1000 digits):
  1. Use FFT-based multiplication and Taylor series division.
  2. Use quadratically convergent algorithms for transcendentals.
- Operator overloading (available in Fortran-90, C++ and other languages) can be exploited to construct high-level translation interfaces, which greatly simplify conversion of existing code.

1. D.H. Bailey, Y. Hida, X.S. Li and B. Thompson, "ARPREC: An arbitrary precision computation package," `http://www.davidhbailey.com/dhbpapers/arprec.pdf`.
2. Y. Hida, X.S. Li and D.H. Bailey, "Algorithms for quad-double precision floating point arithmetic," *15th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society, 2001, pg. 155–162.

Free software for high-precision computation

1. ARPREC: Arbitrary precision, with numerous algebraic and transcendental functions. High-level interfaces for C++ and Fortran-90. `http://crd.lbl.gov/~dhbailey/mpdist`.
2. GFORTRAN: Now provides full REAL*16 (IEEE 128-bit, or 34-digit) support.
3. GMP: Produced by a volunteer effort and distributed under the GNU license. `http://gmplib.org`.
4. MPFR: C library for multiple-precision floating-point computations with exact rounding, based on GMP. `http://www.mpfr.org`.
5. MPFR++: High-level C++ interface to MPFR. `http://perso.ens-lyon.fr/nathalie.revol/software.html`.
6. MPFUN2015: Similar to ARPREC, but is written entirely in Fortran-90. A 100% thread-safe design. `http://crd.lbl.gov/~dhbailey/mpdist`.
7. QD: Performs "double-double" (31 digits) and "quad-double" (62 digits) arithmetic. High-level interfaces for C++ and Fortran-90. `http://crd.lbl.gov/~dhbailey/mpdist`.

U.C. Berkeley's CORVETTE project

Developing software tools to find and ameliorate numerical anomalies in large-scale computations:

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

- C. Rubio-Gonzalez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu and D. Hough, "Precimonious: Tuning assistant for floating-point precision," *Proceedings of SC13*, 17 Nov 2013.

Some applications where high precision is essential

1. Planetary orbit calculations (32 digits).
2. Supernova simulations (32–64 digits).
3. Climate modeling (32 digits).
4. Optimization problems in biology and other fields (32 digits).
5. Coulomb n-body atomic system simulations (32–120 digits).
6. Schrodinger solutions for lithium and helium atoms (32 digits).
7. Electromagnetic scattering theory (32–100 digits).
8. Scattering amplitudes of fundamental particles (32 digits).
9. Discrete dynamical systems (32 digits).
10. The Taylor algorithm for ODEs (100–600 digits).
11. Ising integrals from mathematical physics (100–1000 digits).
12. Problems in experimental mathematics (100–50,000 digits).

Long-term planetary orbit calculations

Researchers have recognized for centuries that planetary orbits exhibit chaotic behavior:

> *"The orbit of any one planet depends on the combined motions of all the planets, not to mention the actions of all these on each other. To consider simultaneously all these causes of motion and to define these motions by exact laws allowing of convenient calculation exceeds, unless I am mistaken, the forces of the entire human intellect."* *[Isaac Newton, Principia, 1687]*

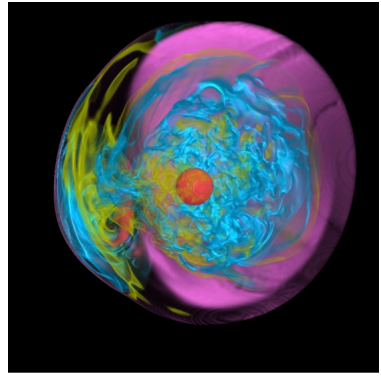
Long-term simulations of planetary orbits using double precision do fairly well for long periods, but then fail at certain key junctures.

Researchers have found that double-double or quad-double arithmetic is required to avoid severe inaccuracies, even if other techniques are employed to reduce numerical error.

- G. Lake, T. Quinn and D. C. Richardson, "From Sir Isaac to the Sloan survey: Calculating the structure and chaos due to gravity in the universe," *Proc. of the 8th ACM-SIAM Symp. on Discrete Algorithms*, 1997, pg. 1–10.

Supernova simulations

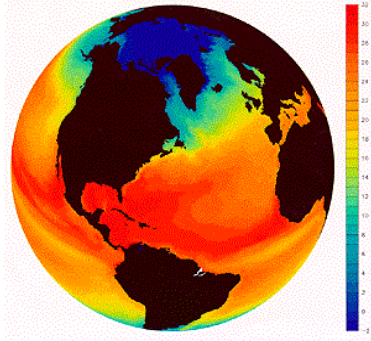
- Researchers at LBNL have used quad-double arithmetic to solve for non-local thermodynamic equilibrium populations of iron and other atoms in the atmospheres of supernovas.
- Iron may exist in several species, so it is necessary to solve for all species simultaneously.
- Since the relative population of any state from the dominant state is proportional to the exponential of the ionization energy, the dynamic range of these values can be very large.
- The quad-double portion now dominates the entire computation.



- 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.

Climate modeling: High-precision for reproducibility

- Climate and weather simulations are fundamentally chaotic: if microscopic changes are made to the current state, soon the future state is quite different.
- In practice, computational results are altered even if minor changes are made to the code or the system.
- This numerical variation is a major nuisance for code maintenance.
- He and Ding found that by using double-double arithmetic in two key inner loops, most of this numerical variation disappeared.



- Y. He and C. 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.

Optimization solutions in biochemistry

- Michael Saunders (a well-known optimization researcher) and his graduate students have encountered significant numerical difficulties when trying to produce numerically reliable results using the Modular In-core Nonlinear Optimization System (MINOS) software.
- In many applications, final results are only produced to a few significant digits, due to numerical error.
- For one biochemistry problem in particular, Harvard researchers have asserted that an exact rational arithmetic scheme was the only means to produce reliable results.
- When the MINOS software was converted to use gfortran's new REAL*16 (quad precision) facility, almost all of these troublesome applications (including the biochemistry problem) now produce strong, numerically reliable results.

- D. Ma and M. A. Saunders, "Solving multiscale linear programs using the simplex method in quadruple precision," to appear in M. Al-Baali, L. Grandinetti and A. Purnama, ed., *Recent Developments in Numerical Analysis and Optimization*, Springer, 2014.

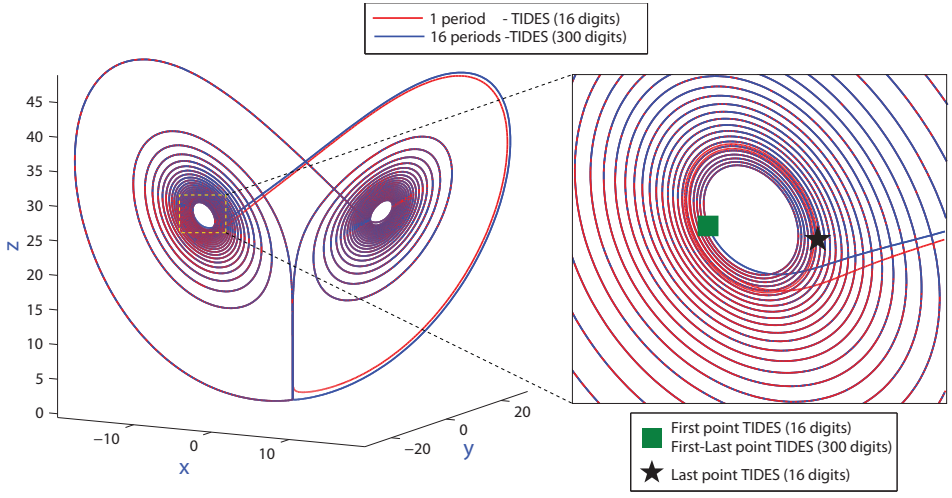
Coulomb n-body atomic system simulations

- Alexei Frolov of Queen's University in Canada has used high-precision arithmetic to solve a generalized eigenvalue problem that arises in Coulomb n-body interactions.
- Matrices are typically 5000×5000 and are very nearly singular.
- Computations typically involve massive cancellation, and high-precision arithmetic must be employed to obtain numerically reproducible results.
- Frolov has also computed elements of the Hamiltonian matrix and the overlap matrix in four- and five-body systems.
- These computations typically require 120-digit arithmetic.

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

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

Taylor's method for ODEs with high-precision arithmetic



Numerical integration of the L25-R25 unstable periodic orbit for the Lorenz model during 16 time periods using TIDES with 300 digits, versus 1 time period using DP.

Experimental mathematics: Discovering new mathematical results

Methodology:

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

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

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

The PSLQ integer relation algorithm

Let (x_n) be a given vector of real numbers. An integer relation algorithm either finds integers (a_n) such that

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \quad = \quad 0$$

(to within the "epsilon" of the arithmetic being used), or else finds bounds within which no relation can exist.

The "PSLQ" algorithm of mathematician-sculptor Helaman Ferguson is the most widely used integer relation algorithm.

Integer relation detection requires very high precision (at least $n \times d$ digits, where d is the size in digits of the largest a_k), both in the input data and in the operation of the algorithm.

1. H. R. P. Ferguson, D. H. Bailey and S. Arno, "Analysis of PSLQ, an integer relation finding algorithm," *Mathematics of Computation*, vol. 68, no. 225 (Jan 1999), pg. 351–369.
2. D. H. Bailey and D. J. Broadhurst, "Parallel integer relation detection: Techniques and applications," *Mathematics of Computation*, vol. 70, no. 236 (Oct 2000), pg. 1719–1736.

The first major PSLQ discovery: The BBP formula for π

In 1996, a PSLQ program discovered this new formula for π:

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

This formula permits one to compute binary (or hexadecimal) digits of π beginning at an arbitrary starting position, using a very simple scheme that requires only standard 64-bit or 128-bit arithmetic.

In 2004, Borwein, Galway and Borwein proved that no base-n formulas of this type exist for π, except when $n = 2^m$.

BBP-type formulas (discovered with PSLQ) are now known for numerous other mathematical constants.

1. D. H. Bailey, P. B. Borwein and S. Plouffe, "On the rapid computation of various polylogarithmic constants," *Mathematics of Computation*, vol. 66, no. 218 (Apr 1997), pg. 903–913.
2. J. M. Borwein, W. F. Galway and D. Borwein, "Finding and excluding b-ary Machin-type BBP formulae," *Canadian Journal of Mathematics*, vol. 56 (2004), pg. 1339–1342.

High-precision numerical integration: The tanh-sinh algorithm

Given $f(x)$ defined on $(-1, 1)$, define $g(t) = \tanh(\pi/2 \sinh t)$. Then

$$\int_{-1}^{1} f(x)\,dx \approx h \sum_{j=-N}^{N} w_j f(x_j),$$

where $x_j = g(h_j)$ and $w_j = g'(h_j)$.

Features:

- Reducing h by half typically doubles the number of correct digits.
- Works well even for functions with singularities at the endpoints.
- Generation of abscissas and weights is *much* faster than Gaussian quadrature.

We have found that tanh-sinh is the best general-purpose integration scheme for functions with vertical derivatives or singularities at endpoints, or for any function at very high precision (> 1000 digits).

1. D. H. Bailey, X. S. Li and K. Jeyabalan, "A Comparison of three high-precision Quadrature Schemes," *Experimental Mathematics*, vol. 14 (2005), no. 3, pg. 317–329.
2. H. Takahasi and M. Mori, "Double exponential formulas for numerical integration," *Publications of RIMS*, Kyoto University, vol. 9 (1974), pg. 721-741.

Ising integrals from mathematical physics

We applied our methods to study three classes of integrals: C_n are connected to quantum field theory, D_n arise in the Ising theory of mathematical physics, while the E_n integrands are derived from Dn:

$$C_n := \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n}$$

$$D_n := \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{\prod_{i<j}\left(\frac{u_i-u_j}{u_i+u_j}\right)^2}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n}$$

$$E_n = 2\int_0^1 \cdots \int_0^1 \left(\prod_{1\le j<k\le n} \frac{u_k - u_j}{u_k + u_j}\right)^2 \mathrm{d}t_2\, dt_3 \cdots \mathrm{d}t_n$$

where in the last line $u_k = t_1 t_2 \cdots t_k$.

D. H. Bailey, J. M. Borwein and R. E. Crandall, "Integrals of the Ising class," *Journal of Physics A: Mathematical and General*, vol. 39 (2006), pg. 12271–12302.

Limiting value of C_n: What is this number?

Key observation: The C_n integrals can be converted to one-dimensional integrals involving the modified Bessel function $K_0(t)$:

$$C_n \;=\; \frac{2^n}{n!}\int_0^\infty t K_0^n(t)\,\mathrm{d}t$$

High-precision numerical values, computed using this formula and tanh-sinh quadrature, approach a limit. For example:

$$C_{1024} \;=\; 0.630473503374386796122040192710878904354587078 7\ldots$$

What is this number? We copied the first 50 digits into the online Inverse Symbolic Calculator (ISC) at `http://carma-lx1.newcastle.edu.au:8087`. The result was:

$$\lim_{n\to\infty} C_n \;=\; 2e^{-2\gamma}.$$

where γ denotes Euler's constant. This is now proven.

Other Ising integral evaluations found using PSLQ

$$
\begin{aligned}
D_2 &= 1/3 \\
D_3 &= 8 + 4\pi^2/3 - 27\,\mathrm{L}_{-3}(2) \\
D_4 &= 4\pi^2/9 - 1/6 - 7\zeta(3)/2 \\
E_2 &= 6 - 8\log 2 \\
E_3 &= 10 - 2\pi^2 - 8\log 2 + 32\log^2 2 \\
E_4 &= 22 - 82\zeta(3) - 24\log 2 + 176\log^2 2 - 256(\log^3 2)/3 \\
&\quad +16\pi^2\log 2 - 22\pi^2/3 \\
E_5 &= 42 - 1984\,\mathrm{Li}_4(1/2) + 189\pi^4/10 - 74\zeta(3) - 1272\zeta(3)\log 2 \\
&\quad +40\pi^2\log^2 2 - 62\pi^2/3 + 40(\pi^2\log 2)/3 + 88\log^4 2 \\
&\quad +464\log^2 2 - 40\log 2
\end{aligned}
$$

where ζ is the Riemann zeta function and $Li_n(x)$ is the polylog function. E_5 remained a "numerical conjecture" for several years, but was proven in March 2014 by Erik Panzer.

Box integrals

The following integrals appear in numerous applications:

$$B_n(s) := \int_0^1 \cdots \int_0^1 \left(r_1^2 + \cdots + r_n^2\right)^{s/2} \, \mathrm{d}R$$

$$\Delta_n(s) := \int_0^1 \cdots \int_0^1 \left((r_1 - q_1)^2 + \cdots + (r_n - q_n)^2\right)^{s/2} \, \mathrm{d}R\mathrm{d}Q$$

- $B_n(1)$ is average distance of a random point from the origin.
- $\Delta_n(1)$ is average distance between two random points.
- $B_n(-n+2)$ is average electrostatic potential in an n-cube whose origin has a unit charge.
- $\Delta_n(-n+2)$ is average electrostatic energy between two points in a uniform n-cube of charged "jellium."
- Recently integrals of this type have arisen in neuroscience, e.g. the average distance between synapses in a mouse brain.

D. H. Bailey, J. M. Borwein and R. E. Crandall, "Box integrals," *Journal of Computational and Applied Mathematics*, vol. 206 (2007), pg. 196–208.

Sample evaluations of box integrals

n	s	$B_n(s)$
any	even $s \geq 0$	rational, e.g., : $B_2(2) = 2/3$
1	$s \neq -1$	$\frac{1}{s+1}$
2	-4	$-\frac{1}{4} - \frac{\pi}{8}$
2	-3	$-\sqrt{2}$
2	-1	$2\log(1+\sqrt{2})$
2	1	$\frac{1}{3}\sqrt{2} + \frac{1}{3}\log(1+\sqrt{2})$
2	3	$\frac{7}{5}\sqrt{2} + \frac{3}{20}\log(1+\sqrt{2})$
2	$s \neq -2$	$\frac{2}{2+s}{}_2F_1\left(\frac{1}{2}, -\frac{s}{2}; \frac{3}{2}; -1\right)$
3	-5	$-\frac{1}{6}\sqrt{3} - \frac{1}{12}\pi$
3	-4	$-\frac{3}{2}\sqrt{2}\arctan\frac{1}{\sqrt{2}}$
3	-2	$-3G + \frac{3}{2}\pi\log(1+\sqrt{2}) + 3\ \mathrm{Ti}_2(3-2\sqrt{2})$
3	-1	$-\frac{1}{4}\pi + \frac{3}{2}\log\left(2+\sqrt{3}\right)$
3	1	$\frac{1}{4}\sqrt{3} - \frac{1}{24}\pi + \frac{1}{2}\log\left(2+\sqrt{3}\right)$
3	3	$\frac{2}{5}\sqrt{3} - \frac{1}{60}\pi - \frac{7}{20}\log\left(2+\sqrt{3}\right)$

Here F is hypergeometric function; G is Catalan; Ti is Lewin's inverse-tan function.

Algebraic numbers in Poisson potential functions

Lattice sums arising from the Poisson equation have been studied widely in mathematical physics and image processing. We numerically discovered, and then proved, that for rational (x, y), the 2-D Poisson potential function satisfies

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

where α is *algebraic*, i.e., the root of an integer polynomial

$$0 \;=\; a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_n\alpha^n$$

The minimal polynomials for these α were found by PSLQ calculations, with the $(n+1)$-long vector $(1, \alpha, \alpha^2, \cdots, \alpha^n)$ as input, where $\alpha = \exp(\pi\phi_2(x,y))$. PSLQ returns the vector of integer coefficients $(a_0, a_1, a_2, \cdots, a_n)$ as output.

1. 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), pg. 115201, `http://www.davidhbailey.com/dhbpapers/PoissonLattice.pdf`.
2. D. H. Bailey and J. M. Borwein, "Compressed lattice sums arising from the Poisson equation: Dedicated to Professor Hari Sirvastava," *Boundary Value Problems*, vol. 75 (2013), DOI: 10.1186/1687-2770-2013-75, `http://www.boundaryvalueproblems.com/content/2013/1/75`.

Samples of minimal polynomials found by PSLQ

k	Minimal polynomial for $\exp(8\pi\phi_2(1/k, 1/k))$
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	$1 - 216\alpha + 860\alpha^2 - 744\alpha^3 + 454\alpha^4 - 744\alpha^5 + 860\alpha^6 - 216\alpha^7 + \alpha^8$

The minimal polynomial for $\exp(8\pi\phi_2(1/32, 1/32))$ has degree 128, with individual coefficients ranging from 1 to over 10^{56}. This PSLQ computation required 10,000-digit precision. See next page.

Other polynomials required up to 50,000-digit precision.

Degree-128 minimal polynomial found for $k = 32$

$$-1 + 21888\alpha + 5893184\alpha^{2} + 15077928064\alpha^{3} - 3696628330464\alpha^{4} - 287791501240448\alpha^{5} - 30287462976198976\alpha^{6}$$
$$+4426867843186404992\alpha^{7} - 554156920878198587888\alpha^{8} + 10731545733669133574528\alpha^{9}$$
$$+120048731928709050250048\alpha^{10} + 4376999211577765512726656\alpha^{11} - 279045693458194222125366432\alpha^{12}$$
$$+18747586287780118903854334848\alpha^{13} - 643310226865188446831485766208\alpha^{14}$$
$$+12047117225922787728443496655488\alpha^{15} - 117230595100328033884939566091384\alpha^{16}$$
$$+667772184328316952814362214365568\alpha^{17} - 4130661734713288144037409932696512\alpha^{18}$$
$$+72313626239383964765274946226530432\alpha^{19} - 1891420571205861612091802761809141088\alpha^{20}$$
$$+38770881730553471470590641060872686464\alpha^{21} - 577943965397394779947709633563006963008\alpha^{22}$$
$$+6279796382074485140847650604801614559872\alpha^{23} - 50438907678331243798448849245156136801232\alpha^{24}$$
$$+305806320133365055812520453224169520739712\alpha^{25} - 1441007171934715336769224848138270812591296\alpha^{26}$$
$$+5554617356232728647085822946642640269497472\alpha^{27} - 20280024430170705107000630261773759070647328\alpha^{28}$$
$$+99541720739995105011861264308551867164583808\alpha^{29} - 754081464712315412970559119390477134883548736\alpha^{30}$$
$$+6271958646895434365874802435136411922022336128\alpha^{31} - 45931349314815625339442690290912948480194150172\alpha^{32}$$
$$+280907040806572157908285324812126135484630889344\alpha^{33} - 1427273782916972532576299009596755423149111059136\alpha^{34}$$
$$+6055180299673737231932804443230077408291723908736\alpha^{35} - 21609910939164553316101994301952988793013291135584\alpha^{36}$$
$$+65433275736596914909292838375737685959952141180288\alpha^{37} - 169928170513492897108417040254326115991438719391296\alpha^{38}$$
$$+385709310577705218843549196766620216295554031550592\alpha^{39} - 801233230832691550861608914233661767474963249815792\alpha^{40}$$
$$+1706210557291030772074402183123327251333271061516160\alpha^{41} - 4421210594351357102505784181831242174063263551938496\alpha^{42}$$
$$+14444199585866329915643888187597383540233619718619776\alpha^{43} - 50968478530199956388487913417905125665738409426112032\alpha^{44}$$
$$+169891313454945514927724813351516976839425267825908096\alpha^{45} - 506612996672385619931633440499093959534203673546181440\alpha^{46}$$
$$+1330573388204326565144545192834096788469932897185696896\alpha^{47} - 3069501638444045841407951432645059776135089489403138888\alpha^{48}$$
$$+6226636397646752257692349351542872634032398917736673152\alpha^{49} - 11133383491631126059761752734485434504397040890449485504\alpha^{50}$$
$$+17601823309919260471943648355479182983209248554083752576\alpha^{51} - 24723027443995082126054012492323603544226813344022687712\alpha^{52}$$
$$+31141043717679289808081270766611355726695735914995681664\alpha^{53} - 35982430389670551550204799905599476866868765647852189248\alpha^{54}$$
$$+40292583920117898286863491450657424717015372825433076864\alpha^{55} - 48512188214363976290470868896252008979896310883132967248\alpha^{56}$$
$$+69275112214095149977288310632868535966705567728055958400\alpha^{57} - 114516830148561378617778209682642099604147034577152904128\alpha^{58}$$
$$+195760470467323759899736578743283333538805684128806803072\alpha^{59} - 317349593507106729834513764473487031789280056911012860320\alpha^{60}$$
$$+468944248086031450001465269696090117959962662732817675648\alpha^{61} - 622467103741378906100611838210632752408312516281305008960\alpha^{62}$$
$$+738516443137003178837650661261546833168555909499151978624\alpha^{63} - 781916756680856373187881889706233393197646662361906135622\alpha^{64}$$
$$+738516443137003178837650661261546833168555909499151978624\alpha^{65} - 622467103741378906100611838210632752408312516281305008960\alpha^{66}$$
$$+468944248086031450001465269696090117959962662732817675648\alpha^{67} - 317349593507106729834513764473487031789280056911012860320\alpha^{68}$$
$$+195760470467323759899736578743283333538805684128806803072\alpha^{69} - 114516830148561378617778209682642099604147034577152904128\alpha^{70}$$
$$+69275112214095149977288310632868535966705567728055958400\alpha^{71} - 48512188214363976290470868896252008979896310883132967248\alpha^{72}$$
$$+40292583920117898286863491450657424717015372825433076864\alpha^{73} - 35982430389670551550204799905599476866868765647852189248\alpha^{74}$$
$$+31141043717679289808081270766611355726695735914995681664\alpha^{75} - 24723027443995082126054012492323603544226813344022687712\alpha^{76}$$
$$+17601823309919260471943648355479182983209248554083752576\alpha^{77} - 11133383491631126059761752734485434504397040890449485504\alpha^{78}$$
$$+6226636397646752257692349351542872634032398917736673152\alpha^{79} - 3069501638444045841407951432645059776135089489403138888\alpha^{80}$$
$$+1330573388204326565144545192834096788469932897185696896\alpha^{81} - 506612996672385619931633440499093959534203673546181440\alpha^{82}$$
$$+169891313454945514927724813351516976839425267825908096\alpha^{83} - 50968478530199956388487913417905125665738409426112032\alpha^{84}$$
$$+14444199585866329915643888187597383540233619718619776\alpha^{85} - 4421210594351357102505784181831242174063263551938496\alpha^{86}$$
$$+1706210557291030772074402183123327251333271061516160\alpha^{87} - 801233230832691550861608914233661767474963249815792\alpha^{88}$$
$$+385709310577705218843549196766620216295554031550592\alpha^{89} - 169928170513492897108417040254326115991438719391296\alpha^{90}$$
$$+65433275736596914909292838375737685959952141180288\alpha^{91} - 21609910939164553316101994301952988793013291135584\alpha^{92}$$
$$+6055180299673737231932804443230077408291723908736\alpha^{93} - 1427273782916972532576299009596755423149111059136\alpha^{94}$$
$$+280907040806572157908285324812126135484630889344\alpha^{95} - 45931349314815625339442690290912948480194150172\alpha^{96}$$
$$+6271958646895434365874802435136411922022336128\alpha^{97} - 754081464712315412970559119390477134883548736\alpha^{98}$$
$$+99541720739995105011861264308551867164583808\alpha^{99} - 20280024430170705107000630261773759070647328\alpha^{100}$$
$$+5554617356232728647085822946642640269497472\alpha^{101} - 1441007171934715336769224848138270812591296\alpha^{102}$$
$$+305806320133365055812520453224169520739712\alpha^{103} - 50438907678331243798448849245156136801232\alpha^{104}$$
$$+6279796382074485140847650604801614559872\alpha^{105} - 577943965397394779947709633563006963008\alpha^{106}$$
$$+38770881730553471470590641060872686464\alpha^{107} - 1891420571205861612091802761809141088\alpha^{108}$$
$$+72313626239383964765274946226530432\alpha^{109} - 4130661734713288144037409932696512\alpha^{110}$$
$$+667772184328316952814362214365568\alpha^{111} - 117230595100328033884939566091384\alpha^{112}$$
$$+12047117225922787728443496655488\alpha^{113} - 643310226865188446831485766208\alpha^{114}$$
$$+18747586287780118903854334848\alpha^{115} - 279045693458194222125366432\alpha^{116}$$
$$+4376999211577765512726656\alpha^{117} + 120048731928709050250048\alpha^{118} + 10731545733669133574528\alpha^{119}$$
$$-554156920878198587888\alpha^{120} + 4426867843186404992\alpha^{121} - 30287462976198976\alpha^{122}$$
$$-287791501240448\alpha^{123} - 3696628330464\alpha^{124} + 15077928064\alpha^{125} + 5893184\alpha^{126} + 21888\alpha^{127} - \alpha^{128}$$

MPFUN2015: A NEW thread-safe arbitrary precision package

- 100% thread safe. NO global read/write global variables.
- NO initialization required (unless over 20,000 digits required).
- NO system-dependent features or settings — the code only assumes 64-bit IEEE format (it does *not* depend on rounding mode).
- NO reliance on sophisticated language features that might not be supported.
- Installation is a snap — one compile line.
- Precision is scalable to over billion digits; largest size is over $10^{64,000,000}$.
- A full package of basic arithmetic functions. Higher-level routines for binary-decimal conversion, I/O and transcendentals.
- A full-featured high-level language interface for Fortran-90 is provided, so that most users need only make minor changes to existing double precision code.
- Extensive error checking and a solution to the double precision accuracy problem.
- Currently only Fortran-90 version; C++ version is being developed in Australia.
- Available at: `http://www.davidhbailey.com`.

For further reading:

1. D. H. Bailey and J. M. Borwein, "High-precision arithmetic in mathematical physics," manuscript, 16 Jan 2015. http://www.davidhbailey.com/dhbpapers/hp-math-physics.pdf.
2. D. H. Bailey, R. Barrio and J. M. Borwein, "High precision computation: Mathematical physics and dynamics," *Applied Mathematics and Computation*, vol. 218 (2012), pg. 10106–10121. http://www.davidhbailey.com/dhbpapers/hpmpd.pdf.
3. David H. Bailey, "MPFUN2015: A thread-safe arbitrary precision package," manuscript, 13 Mar 2015. http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf.

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