

Discovery of large Poisson polynomials using a new arbitrary precision software package

David H. Bailey

<http://www.davidhbailey.com>

Lawrence Berkeley National Lab (retired)

University of California, Davis, Dept. of Computer Science

November 11, 2015

Some applications of high-precision arithmetic

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. Electromagnetic scattering theory (32–100 digits).
7. The Taylor algorithm for ODEs (100–600 digits).
8. Ising integrals from mathematical physics (100–1000 digits).
9. Problems in experimental mathematics (100–100,000 digits).

These and other examples are described in:

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

The PSLQ integer relation algorithm

Let $X = (x_k)$ be an $(m + 1)$ -long real or complex vector. An integer relation algorithm such as PSLQ finds a nontrivial integer vector $A = (a_k)$ such that

$$a_0x_0 + a_1x_1 + \cdots + a_mx_m = 0.$$

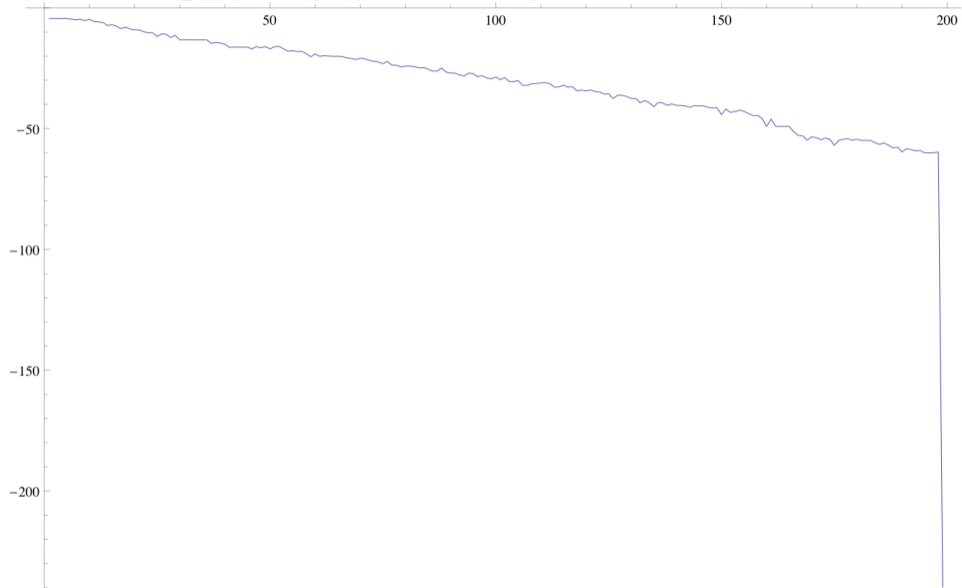
- ▶ The multipair PSLQ algorithm is a more efficient and moderately parallelizable variant of PSLQ, the most widely used integer relation algorithm (although some use a variant of LLL).
- ▶ Integer relation detection (by any algorithm) requires very high precision: at least $(m + 1) \cdot \max_k \log_{10} |a_k|$ digits, both in the input data and the algorithm.
- ▶ Multipair PSLQ is extremely efficient with precision — it can usually detect a relation when the precision is only a few percent higher than this minimum level.

- ▶ D. H. Bailey and D. J. Broadhurst, “Parallel integer relation detection: Techniques and applications,” *Mathematics of Computation*, vol. 70, no. 236 (Oct 2000), 1719–1736.

Multipair PSLQ, continued

- ▶ Iterations of the multipair PSLQ algorithm develop a sequence of invertible integer matrices A_n , their inverses B_n and real matrices H_n (in lower quadrature form), so that the reduced vector $Y = B_n \cdot X$ has steadily smaller entries.
- ▶ A relation is detected when the size of smallest entry of the Y vector drops below the epsilon specified for detection (typically roughly 10^{-p} , where p is the precision level in digits).
- ▶ The size of this drop can be viewed as a “confidence level” that the relation is not a numerical artifact: a drop of 20+ orders of magnitude almost always indicates a real relation.
- ▶ See next viewgraph for an illustration of the steadily decreasing value of $\min |y_k|$, leading up to iteration 199, when the relation was discovered.

Decrease of $\log_{10}(\min |y_i|)$ in a multipair PSLQ run



Application of multipair PSLQ

One simple but important application of multipair PSLQ is to recognize a computed numerical value as the root of an integer polynomial of degree m .

Example: The following constant is suspected to be an algebraic number:

$$\alpha = 1.232688913061443445331472869611255647068988824547930576057634684778\dots$$

What is its minimal polynomial?

Method: Compute the vector $(1, \alpha, \alpha^2, \dots, \alpha^m)$ for $m = 30$, then input this vector to multipair PSLQ.

Answer (using 250-digit arithmetic):

$$\begin{aligned} 0 = & 6971 - 1440\alpha - 20520\alpha^2 - 98280\alpha^3 - 102060\alpha^4 - 1458\alpha^5 + 80\alpha^6 - 43920\alpha^7 \\ & + 538380\alpha^8 - 336420\alpha^9 + 1215\alpha^{10} - 80\alpha^{12} - 56160\alpha^{13} - 135540\alpha^{14} - 540\alpha^{15} \\ & + 40\alpha^{18} - 7380\alpha^{19} + 135\alpha^{20} - 10\alpha^{24} - 18\alpha^{25} + \alpha^{30} \end{aligned}$$

High-precision numerical integration

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

$$\int_{-1}^1 f(x) dx = \int_{-\infty}^{\infty} f(g(t))g'(t) dt \approx h \sum_{j=-N}^N w_j f(x_j),$$

where $x_j = g(jh)$ and $w_j = g'(jh)$, independent of the function $f(x)$, for some interval h . The summation continues until the terms are consistently smaller than epsilon.

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). Otherwise we use Gaussian quadrature.

- ▶ D. H. Bailey, X. S. Li and K. Jeyabalan, "A comparison of three high-precision quadrature schemes," *Experimental Mathematics*, vol. 14 (2005), 317–329.

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 D_n :

$$C_n := \frac{4}{n!} \int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{du_1}{u_1} \cdots \frac{du_n}{u_n}$$

$$D_n := \frac{4}{n!} \int_0^\infty \cdots \int_0^\infty \frac{\prod_{i<j} \left(\frac{u_i - u_j}{u_i + u_j}\right)^2}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{du_1}{u_1} \cdots \frac{du_n}{u_n}$$

$$E_n = 2 \int_0^1 \cdots \int_0^1 \left(\prod_{1 \leq j < k \leq n} \frac{u_k - u_j}{u_k + u_j} \right)^2 dt_2 dt_3 \cdots dt_n,$$

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

- ▶ 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?

The C_n integrals can be converted to 1-D integrals involving the Bessel function $K_0(t)$:

$$C_n = \frac{2^n}{n!} \int_0^\infty t K_0^n(t) dt.$$

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

$$C_{1024} = 0.6304735033743867961220401927108789043545870787 \dots$$

What is this number? We copied the first 50 digits into the online Inverse Symbolic Calculator (ISC) at <https://isc.carma.newcastle.edu.au>. The result was:

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

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

Other Ising integral evaluations found using multipair PSLQ

$$D_3 = 8 + 4\pi^2/3 - 27L_{-3}(2)$$

$$D_4 = 4\pi^2/9 - 1/6 - 7\zeta(3)/2$$

$$E_2 = 6 - 8\log 2$$

$$E_3 = 10 - 2\pi^2 - 8\log 2 + 32\log^2 2$$

$$E_4 = 22 - 82\zeta(3) - 24\log 2 + 176\log^2 2 - 256(\log^3 2)/3 \\ + 16\pi^2\log 2 - 22\pi^2/3$$

$$E_5 = 42 - 1984\text{Li}_4(1/2) + 189\pi^4/10 - 74\zeta(3) - 1272\zeta(3)\log 2 + 40\pi^2\log^2 2 \\ - 62\pi^2/3 + 40(\pi^2\log 2)/3 + 88\log^4 2 + 464\log^2 2 - 40\log 2,$$

where ζ is the Riemann zeta function and $\text{Li}_n(x)$ is the polylog function.

E_5 remained a “numerical conjecture” for several years, but was proven in March 2014 by Erik Panzer.

Free software for high-precision floating-point computation

High-precision is built into commercial mathematical software packages such as *Maple* and *Mathematica*, but many researchers use one of the following:

1. ARPREC: Includes 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: Supports high-precision integer, rational and floating-point calculations. <http://gmplib.org>.
4. MPFR: C library for arbitrary precision with exact rounding, based on GMP. <http://www.mpfr.org>.
5. MPFR C++: A high-level C++ interface to MPFR with a thread-safe option. <http://www.holoborodko.com/pavel/mpfr>.
6. Pari/GP: A computer algebra system that includes high-precision, with numerous transcendental functions. <http://pari.math.u-bordeaux.fr>.
7. QD: Performs “double-double” (31 digits) and “quad-double” (62 digits) arithmetic, with transcendental functions. High-level interfaces for C++ and Fortran-90. <http://crd.lbl.gov/~dhbailey/mpdist>.

Thread safety: A major challenge for high-precision software

- ▶ Because of greatly magnified run times (often 1000X or more) of high-precision computations, highly parallel implementations are in order. However, most high-precision packages are *not* thread-safe.
- ▶ Many packages employ global read/write variables, e.g., for transcendental function evaluation, which are deadly for thread safety.
- ▶ The working precision level, a global variable that is changed frequently within the package itself and often by users also, is particularly troublesome.

One thread-safe package: The MPFR C++ package, which calls the lower-level MPFR package, is thread-safe (provided MPFR is compiled with the thread-safe build option). MPFR is a very well-designed, features correct rounding and is the fastest low-level package currently available.

But there is no thread-safe high-level Fortran package.

MPFUN2015: A new thread-safe arbitrary precision package

DHB has written a new package, available in two versions:

- ▶ MPFUN-Fort: Completely self-contained, all-Fortran version. Compilation is a simple one-line command, which completes in a few seconds.
- ▶ MPFUN-MPFR: Calls the MPFR package for lower-level operations. Installation is significantly more complicated (since GMP and MPFR must first be installed), but performance is roughly 3X faster.

The two versions are “plug-compatible” in the sense that applications written for one version also work with the other.

A C++ version is in the works.

The software is available at <http://www.davidhbailey.com/dhbsoftware>.

Full details of design, algorithms, installation and usage are given in

- ▶ D. H. Bailey, “MPFUN2015: A thread-safe arbitrary precision computation package,” manuscript, 1 Oct 2015. <http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf>.

Features of MPFUN-Fort and MPFUN-MPFR

- ▶ 100% thread safe (provided MPFR is compiled with the thread-safe build option).
- ▶ Both employ advanced algorithms, including FFT-based arithmetic.
- ▶ Both versions include a full-featured, high-level Fortran interface (implemented using custom datatypes and operator overloading).
- ▶ Both provide full support for both multiprecision real (MPR) and multiprecision complex (MPC) datatypes, as well as a wide variety of mixed-mode operations (e.g., double precision times multiprecision real, multiprecision complex to an integer power, etc.).
- ▶ All common transcendentals and several special functions are included.
- ▶ Precision is scalable to over one million digits.
- ▶ Both provide an effective solution to the double precision accuracy problem that plagues many high-precision packages.

Solution to the thread safety problem

- ▶ The user sets a default (maximum) working precision level in a compile-time data statement.
- ▶ A “working precision” **carried in data structure itself** is associated with each multiprecision (MP) variable and array element.
- ▶ The result of any operation or function involving MP variables or array elements inherits the largest working precision level of the input operands.
- ▶ The built-in function `mpreal` allows one to convert a MP variable or array element to one with a different working precision level.
- ▶ The built-in function `mpwprec` returns the working precision level currently assigned to any MP variable or array element.

Converting existing codes to use MPFUN-Fort or MPFUN-MPFR

- ▶ Place this line in every subprogram that contains a MP variable or array:
`use mpmodule`
- ▶ To designate a variable or array as multiprecision real (MPR) or complex (MPC):
`type (mp_real) a, b(10), c(10,10); type (mp_complex) d, e, f(20)`
- ▶ All basic and mixed-mode operations are supported, e.g.:
`d = a + b(i) * sqrt(3.d0 - c(i,j))`

Functions provided:

- ▶ Standard Fortran functions with MPR arguments:
`abs, acos, aint, anint, asin, atan, atan2, cos, cosh, dble, exp, log, max, min, sign, sin, sqrt, tan, tanh`
- ▶ Stanford Fortran functions with MPC arguments:
`abs, aimag, conjg, cos, dcplx, exp, log, sin, sqrt`
- ▶ Special functions:
`bessj, gamma, incgamma, mpberne, zeta, zetaem`

A solution to the double precision accuracy loss problem

- ▶ If a user code includes a statement such as $a = b + d / 3.d0$, where a and b are multiprecision real (MPR) and d is double precision (DP), then by established language conventions the division is only performed to DP accuracy, and this low accuracy propagates to the result a .
- ▶ Similar difficulties afflict multiprecision programs written in other languages, including C and C++.
- ▶ Both MPFUN-Fort and MPFUN-MPFR check every DP constant, variable and expression value in MP statements at **execution time** to see if it has more than 40 significant bits. If so, it is flagged as an error. This feature catches 99.99% of the precision loss problems.
- ▶ Special routines handle legitimate DP data with more than 40 bits.
- ▶ To convert a floating constant to full precision, enclose it in apostrophes, e.g.,
 $a = '1.234'$

Libraries and applications included with MPFUN-Fort and MPFUN-MPFR

Description	Lines of code
MPFUN-Fort library	33,640
MPFUN-MPFR library*	11,540
One-level standard PSLQ code	752
One-level multipair PSLQ code	905
Two-level multipair PSLQ code	1694
Three-level multipair PSLQ code	2129
Tanh-sinh quadrature code, with 14 integrals	871
Parallel tanh-sinh code, with 14 integrals	949
Poisson polynomial code with 3-level multipair PSLQ	2217
Total	54,697

*To install MPFUN-MPFR, the GMP library (approximately 83,000 lines of C) and the MPFR library (approximately 93,000 lines of C) must first be installed.

The Poisson potential function of mathematical physics

- ▶ Gravitation — the gravitational potential for a given mass distribution.
 - ▶ Electrostatics — the electric potential for a given charge distribution.
 - ▶ Studies of cyclotomic polylogarithms and harmonic sums.
 - ▶ Studies of crystal symmetry.
 - ▶ Studies of the Madelung constant of mathematical physics.
 - ▶ Studies of image sharpening.
-
- ▶ J. Borwein, L. Glasser, R. McPhedran, J. Wan and J. Zucker, *Lattice Sums: Then and Now, Encyclopedia of Mathematics and its Applications*, 150, Cambridge University Press, 2013.
 - ▶ S. Kanemitsu, Y. Tanigawa, H. Tsukada and M. Yoshimoto, “Crystal symmetry viewed as zeta symmetry,” p. 91–130 in T. Aoki, S. Kanemitsu, M. Nakahara and Y. Ohno, eds., *Zeta Functions, Topology and Quantum Physics*, Springer, New York, 2005 (reprinted 2010).

The Poisson potential function problem

In a 2013 study, we numerically discovered, and then proved the intriguing fact that for rational (x, y) , the 2-D Poisson potential function $\phi_2(x, y)$ 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 a some integer polynomial of degree m .

By computing high-precision numerical values of $\phi_2(x, y)$ for various specific rational x and y , and applying a multipair PSLQ program, we were able to produce the explicit minimal polynomials for α in several specific cases.

This study was hampered by the huge complexity and computational costs involved. Some runs required up to 12,000-digit precision, while other runs failed.

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

Kimberley's formula for the degree of the polynomial

Based on our preliminary results, Jason Kimberley of the University of Newcastle, Australia observed that the degree $m(s)$ of the minimal polynomial associated with the case $x = y = 1/s$ appears to be given by the following:

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

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

Does Kimberley's formula hold for larger s ? Why? What is the true mathematical connection between the pair of rationals (x, y) and the algebraic number α ?

New three-level multipair PSLQ program

Employs three levels of numeric precision:

- ▶ Ordinary double precision.
- ▶ Medium precision, typically 100–2000 digits.
- ▶ Full precision, typically many thousands of digits.

When an entry of the double precision reduced vector is smaller than 10^{-14} , or when an entry of one of the integer-valued double precision arrays exceeds $2^{53} \approx 9.007 \cdot 10^{15}$, the medium precision arrays are updated by matrix multiplication.

Similarly, when an entry of the medium precision reduced vector is smaller than the medium precision “epsilon,” the full-precision arrays are updated.

Substantial care must be taken to manage this three-level hierarchy, and to correctly handle numerous atypical scenarios.

A fast algorithm to compute the Poisson potential function $\phi_2(x, y)$

$$\phi_2(x, y) = \frac{1}{2\pi} \log \left| \frac{\theta_2(z, q)\theta_4(z, q)}{\theta_1(z, q)\theta_3(z, q)} \right|,$$

where $q = e^{-\pi}$ and $z = \frac{\pi}{2}(y + ix)$. Compute the four theta functions using the following rapidly convergent formulas involving complex variables:

$$\theta_1(z, q) = 2 \sum_{k=1}^{\infty} (-1)^{k-1} q^{(2k-1)^2/4} \sin((2k-1)z),$$

$$\theta_2(z, q) = 2 \sum_{k=1}^{\infty} q^{(2k-1)^2/4} \cos((2k-1)z),$$

$$\theta_3(z, q) = 1 + 2 \sum_{k=1}^{\infty} q^{k^2} \cos(2kz),$$

$$\theta_4(z, q) = 1 + 2 \sum_{k=1}^{\infty} (-1)^k q^{k^2} \cos(2kz).$$

Empirical formulas to estimate precision, memory, run time, etc.

$m(s)$ is the estimated polynomial degree; D is the detection level;

P_1 is the medium precision level; P_2 is the full precision level;

N is the number of multipair PSLQ iterations;

T is the total one-core run time in seconds; M is the memory in Mbytes.

$$m(s) = 4^{r-1} \prod_{i=1}^r p_i^{2(e_i-1)} m(p_i) \quad \text{i.e., the Kimberley formula}$$

$$\log_{10} D = -0.462 m^2(s), \quad P_1 = 5 m(s), \quad P_2 = 5 m(s) + 0.462 m^2(s),$$

$$\log_{10} N = 105.278/m^2(s) - 33.1073/m(s) + 4.01963 + 0.00696322 m(s),$$

$$\log_{10} T = 374.411/m^2(s) - 79.2388/m(s) + 1.93981 + 0.0176301 m(s),$$

$$M = 1.6 \cdot 10^{-7} \cdot m^2(s)(5m(s) + 0.462 m^2(s)).$$

High-level computational algorithm

1. Given rationals $x = p/s$ and $y = q/s$, select a conjectured minimal polynomial degree $m(s)$ and other parameters for the run, using the formulas above.
2. Calculate $\phi_2(x, y)$ to P_2 -digit precision using the formulas from two viewgraphs above. When done, calculate $\alpha = \exp(8\pi\phi_2(x, y))$ and generate the $(m + 1)$ -long vector $X = (1, \alpha, \alpha^2, \dots, \alpha^m)$, to P_2 -digit precision.
3. Apply the three-level multipair PSLQ algorithm to X . For larger problems, employ a parallel version of the three-level multipair PSLQ code, using the OpenMP construct `DO PARALLEL` to perform certain time-intensive loops in parallel.
4. If no numerically significant relation is found, try again with a larger degree m or higher precision P_2 . If a relation is found, employ the polynomial factorization facilities in *Mathematica* and *Maple* to ensure that the polynomial is irreducible.

Application program and libraries for the Poisson calculations

Description	Language	Lines of code
Poisson polynomial program*	Fortran	2,000
MPFUN-MPFR package	Fortran	12,000
MPFR package	C	93,000
GMP package	C	83,000
Total		190,000

*This includes the computation of $\phi_2(x, y)$ and the 3-level multipair PSLQ program.

Samples of minimal polynomials found by multipair PSLQ

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

These computations are very expensive. The case $x = y = 1/37$ required 51,000-digit precision and produced a polynomial of degree 324, with up to 145-digit coefficients; the computer run consumed 7.7 million core-seconds.

Timings for the case $x = y = 1/35$

Multiprecision software	PSLQ code	Cores	Run time	Speedup
ARPREC	2-level	1	$1.599 \cdot 10^6$	1.00
MPFUN-MPFR	2-level	1	$5.249 \cdot 10^5$	3.05
MPFUN-MPFR	3-level	1	$1.240 \cdot 10^5$	12.90
		2	$7.585 \cdot 10^4$	21.08
		4	$4.121 \cdot 10^4$	38.80
		8	$2.476 \cdot 10^4$	64.58
		16	$1.021 \cdot 10^4$	156.61

The run times are wall-clock run times (in seconds), measured on a 16-core 2.4 GHz MacPro, in a typically busy environment with similar jobs running on other cores.

Parallel speedups

- ▶ While a 12X parallel speedup, and a 156X overall speedup, are certainly welcome, it does not appear possible to run this code efficiently on hundreds or thousands of processor cores.
- ▶ Achieving high speedups is fairly straightforward in the medium precision and full precision routines, but is not easy for the double precision (DP) routines. Thus, by Amdahl's law, the DP portions limit overall parallel scalability.
- ▶ It would be easy to exhibit large parallel speedups by starting, say, with the one-level multipair PSLQ code, which performs all iterations using full precision, but this would violate the principle that parallel implementations and speedup figures should be based on the most efficient single-threaded algorithm available; otherwise speedup results can be highly misleading.

- ▶ D. H. Bailey, "Misleading performance reporting in the supercomputing field," *Scientific Programming*, vol. 1., no. 2 (Winter 1992), 141–151.

Selected runs (degrees, precision, timings, etc.) for $x = y = 1/s$

s	m	$\log_{10}(D)$	P_1	P_2	N	M	C	T (sec.)	$C \cdot T$ (sec.)
20	32	-463.84	160	700	1967	0.81	1	$2.19 \cdot 10^0$	$2.19 \cdot 10^0$
24	64	-1883.78	320	2200	9297	11.33	1	$7.73 \cdot 10^1$	$7.73 \cdot 10^1$
30	64	-1868.01	350	2300	9064	11.33	1	$1.02 \cdot 10^2$	$1.02 \cdot 10^2$
32	128	-7577.07	650	8200	45893	168.20	1	$5.13 \cdot 10^3$	$5.13 \cdot 10^3$
34	128	-7574.93	650	8200	45914	168.20	1	$5.16 \cdot 10^3$	$5.16 \cdot 10^3$
36	144	-9570.86	750	10300	62282	267.10	1	$9.54 \cdot 10^3$	$9.54 \cdot 10^3$
37	324	-48431.32	1650	51000	931254	6579.66	16	$4.84 \cdot 10^5$	$7.74 \cdot 10^6$
38	180	-14951.64	900	16000	120984	642.98	1	$3.88 \cdot 10^4$	$3.88 \cdot 10^4$
39	288	-38330.14	1450	40000	667153	4124.24	16	$2.68 \cdot 10^5$	$4.29 \cdot 10^6$
40	128	-7580.00	650	8200	45655	168.20	1	$5.02 \cdot 10^3$	$5.02 \cdot 10^3$
42	192	-16993.99	1000	18000	150364	829.41	8	$1.57 \cdot 10^4$	$1.26 \cdot 10^5$
44	240	-26604.14	1200	28000	323762	2003.33	8	$7.43 \cdot 10^4$	$5.94 \cdot 10^5$
45	288	-38315.08	1450	40000	660001	4124.24	16	$2.09 \cdot 10^5$	$3.35 \cdot 10^6$
46	264	-32036.34	1350	34000	476902	2921.57	16	$1.06 \cdot 10^5$	$1.70 \cdot 10^6$
48	256	-30248.55	1350	32000	415316	2586.39	16	$8.98 \cdot 10^4$	$1.44 \cdot 10^6$
50	200	-18421.18	1000	20000	168947	974.44	8	$2.12 \cdot 10^4$	$1.69 \cdot 10^5$

s = denominator; m = degree; D = detection level; P_1 = medium precision; P_2 = full precision; N = number of iterations; M = Mbytes; C = cores; T = wall clock time; $C \cdot T$ = total core-seconds.

Selected runs (degrees, precision, timings, etc.) for $x = 1/s, y = q/s$

s	q	m	$\log_{10}(D)$	P_1	P_2	N	M	C	T (sec.)	$C \cdot T$ (sec.)
20	3	32	-359.23	160	700	1637	0.81	1	$3.06 \cdot 10^0$	$3.06 \cdot 10^0$
24	5	32	-336.46	320	2200	1613	11.33	1	$2.83 \cdot 10^0$	$2.83 \cdot 10^0$
30	7	64	-1291.21	350	2300	6867	11.33	1	$1.01 \cdot 10^2$	$1.01 \cdot 10^2$
32	3	128	-5578.06	650	8200	33829	168.20	1	$4.16 \cdot 10^3$	$4.16 \cdot 10^3$
34	3	128	-5411.70	650	8200	32842	168.20	1	$2.53 \cdot 10^3$	$2.53 \cdot 10^3$
36	5	144	-6831.83	750	10300	45559	267.10	1	$6.93 \cdot 10^3$	$6.93 \cdot 10^3$
37	2	324	-35412.09	1650	51000	691277	6579.66	16	$3.42 \cdot 10^5$	$5.47 \cdot 10^6$
38	3	180	-11011.83	900	16000	89722	642.98	1	$7.42 \cdot 10^3$	$7.42 \cdot 10^3$
39	2	288	-27943.70	1450	40000	458238	4124.24	16	$1.84 \cdot 10^5$	$2.94 \cdot 10^6$
40	3	128	-5674.61	650	8200	34273	168.20	1	$4.13 \cdot 10^3$	$4.13 \cdot 10^3$
42	5	192	-12183.50	1000	18000	106770	829.41	8	$1.16 \cdot 10^4$	$9.27 \cdot 10^4$
44	3	240	-19581.93	1200	28000	232713	2003.33	8	$6.18 \cdot 10^4$	$4.95 \cdot 10^5$
45	2	288	-27857.00	1450	40000	482959	4124.24	16	$1.47 \cdot 10^5$	$2.35 \cdot 10^6$
46	3	264	-23318.37	1350	34000	346987	2921.57	16	$7.28 \cdot 10^4$	$1.17 \cdot 10^6$
48	5	256	-21480.15	1350	32000	292974	2586.39	16	$5.93 \cdot 10^4$	$9.50 \cdot 10^5$
50	3	200	-13409.44	1000	20000	122468	974.44	8	$1.63 \cdot 10^4$	$1.30 \cdot 10^5$

s = denominator; m = degree; D = detection level; P_1 = medium precision; P_2 = full precision; N = number of iterations; M = Mbytes; C = cores; T = wall clock time; $C \cdot T$ = total core-seconds.

Analysis and conclusions

- ▶ As far as we are aware, these computations, which employed up to 51,000-digit precision, producing polynomials with degrees up to 324 and integer coefficients up to 10^{145} , constitute the largest successful integer relation computations performed to date.
- ▶ Kimberley's formula was affirmed in every case $x = y = 1/s$, for s up to 50 (except for $s = 41, 43, 47, 49$, which are still too costly to test).
- ▶ Research is need to understand why Kimberley's formula holds.
- ▶ Additional research is needed to understand many other combinations, e.g., $x = p/s$ and $y = q/s$, for different values of p, q and s , which will require even more extreme computation.
- ▶ While a 12X parallel speedup is certainly welcome, a scheme to efficiently employ hundreds or thousands of processor cores is needed. A fundamentally new integer relation algorithm may well be required.

Thanks!

- ▶ This talk is available at <http://www.davidhbailey.com/dhbtalks/dhb-mpfun-pphi.pdf>.
- ▶ A technical report with more details, results and analysis is available at <http://www.davidhbailey.com/dhbpapers/poisson-res.pdf>.
- ▶ The MPFUN-Fort and MPFUN-MPFR software packages are available at <http://www.davidhbailey.com/dhbsoftware>.
- ▶ A technical report giving full details of MPFUN-Fort and MPFUN-MPFR is available at <http://www.davidhbailey.com/dhbpapers/mpfun2015.pdf>.