Highly Parallel, High-Precision Numerical Integration

David H. Bailey[1] and Jonathan M. Borwein[2]

Draft: 2005-04-17

Abstract

This paper describes a scheme for rapidly computing numerical values of definite integrals to very high accuracy, ranging from ordinary machine precision to hundreds or thousands of digits, even for functions with singularities or infinite derivatives at endpoints. Such a scheme is of interest not only in computational physics and computational chemistry, but also in experimental mathematics, where high-precision numerical values of definite integrals can be used to numerically discover new identities. This paper discusses techniques for a parallel implementation of this scheme, then presents performance results for 1-D and 2-D test suites. Results are also given for a certain problem from mathematical physics, which features a difficult singularity, confirming a conjecture to 20,000 digit accuracy. The performance rate for this latter calculation on 1024 CPUs is 690 Gflop/s. We believe that this and one other 20,000-digit integral evaluation that we report are the highest-precision non-trivial numerical integrations performed to date.

1. Introduction

Numerical integration (often termed "numerical quadrature") has numerous applications in applied mathematics, particularly in fields such as mathematical physics and computational chemistry. Recently such techniques have found application in the emerging discipline of experimental mathematics, namely the application of high-performance computing to research questions in mathematics. In particular, high-precision numerical values of certain definite integrals, when combined with integer relation detection algorithms, can be used to discover previously unknown analytic evaluations (i.e., closed-form formulas) for certain integrals, and to provide strong numerical confirmation that such computer-discovered identities are valid.

An "integer relation detection" scheme is a numerical algorithm which, given an n-long vector (x_i) of high-precision floating-point values, can recover the integer coefficients (a_i) such that $a_1x_1 + a_2x_2 + \cdots + a_nx_n = 0$ (to within available precision), or else determine

[1]Lawrence Berkeley National Laboratory, Berkeley, CA 94720 dhbailey@lbl.gov. This work was supported in part 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.

[2]Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 2W5, Canada. jborwein@cs.dal.ca. This work supported in part by NSERC and the Canada Research Chair Programme.

that there are no such integers less than a certain size. The precision required in these computations is typically 100–500 digits, although occasionally thousands of digits are required. In one instance, 50,000-digit arithmetic was needed to find the underlying relation [5]. The best-known integer relation algorithm is the "PSLQ" algorithm [4].

Here are some examples of this approach. In 2002, the present authors and Greg Fee of Canada were inspired by a recent problem in the *American Mathematical Monthly* [1]. They found, by using an early version of the integration scheme described in this paper, together with a PSLQ program, that if $C(a)$ is defined by

$$C(a) \;=\; \int_0^1 \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)},$$

then

$$\begin{aligned} C(0) &= \pi\log 2/8 + G/2 \\ C(1) &= \pi/4 - \pi\sqrt{2}/2 + 3\sqrt{2}\arctan(\sqrt{2})/2 \\ C(\sqrt{2}) &= 5\pi^2/96. \end{aligned}$$

Here $G = \sum_{k\geq 0}(-1)^k/(2k+1)^2 = 0.91596559417\ldots$ is Catalan's constant. These specific experimental results then led to the following general result, which now has been rigorously established, and several others:

$$\int_0^\infty \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)} \;=\; \frac{\pi}{2\sqrt{a^2-1}}\left[2\arctan(\sqrt{a^2-1}) - \arctan(\sqrt{a^4-1})\right].$$

As a second example, the present authors empirically determined that

$$\begin{aligned} \frac{2}{\sqrt{3}}\int_0^1 \frac{\log^6(x)\arctan[x\sqrt{3}/(x-2)]}{x+1}\,dx \;=\; & \frac{1}{81648}\,[-229635L_3(8) \\ & +29852550L_3(7)\log 3 - 1632960L_3(6)\pi^2 + 27760320L_3(5)\zeta(3) \\ & -275184L_3(4)\pi^4 + 36288000L_3(3)\zeta(5) - 30008L_3(2)\pi^6 \\ & -57030120L_3(1)\zeta(7)]\,, \end{aligned}$$

where $L_3(s) = \sum_{n\geq 1}[1/(3n-2)^s - 1/(3n-1)^s]$ and where $\zeta(s) = \sum_{n\geq 1} 1/n^s$ is the Riemann zeta function. Based on these experimental results, general results of this type have been conjectured but few have yet been rigorously established.

A third result is the following, which was found by one of the present authors (Borwein) and British physicist David Broadhurst [8]:

$$\begin{aligned} &\frac{24}{7\sqrt{7}}\int_{\pi/3}^{\pi/2} \log\left|\frac{\tan t+\sqrt{7}}{\tan t-\sqrt{7}}\right| dt \;\stackrel{?}{=}\; L_{-7}(2) \;= \\ &\sum_{n=0}^{\infty}\left[\frac{1}{(7n+1)^2} + \frac{1}{(7n+2)^2} - \frac{1}{(7n+3)^2} + \frac{1}{(7n+4)^2} - \frac{1}{(7n+5)^2} - \frac{1}{(7n+6)^2}\right]. \end{aligned}$$

This integral arose out of some studies in quantum field theory, in analysis of the volume of ideal tetrahedra in hyperbolic space. It is the simplest of 998 empirically determined cases where the volume of a hyperbolic knot complement is expressible in terms of an L-series and an apparently unexpected integral or sum [8]. The question mark is used here because although this identity has been numerically verified to *very* high precision (see Section 10), as of this date no proof is yet known.

The above examples are ordinary one-dimensional integrals. Two-dimensional integrals are also of interest. Along this line, recently the present authors determined that

$$\frac{2}{3}\int_0^1\int_0^1 \sqrt{x^2+y^2}\,dx\,dy + \frac{1}{3}\int_0^1\int_0^1 \sqrt{1+(u-v)^2}\,du\,dv \;=\; \frac{1}{9}\sqrt{2} + \frac{5}{9}\log(\sqrt{2}+1) + \frac{2}{9}.$$

See [3] for some additional examples.

2. The Tanh-Sinh Quadrature Algorithm

In a previous paper [7], one of the present authors and two co-authors investigated several numerical integration schemes, suitable for high-precision usage, and exhibited results for computer runs with 400- and 1000-digit precision. The authors concluded the "tanh-sinh" quadrature scheme holds the best promise for very high-precision usage. The tanh-sinh scheme features: (1) an initialization procedure that is fundamentally faster than the other schemes, (2) the ability to obtain fully accurate results even for many integrand functions with infinite derivatives or blow-up singularities at endpoints, and (3) very fast run times.

The tanh-sinh scheme is based on the observation, rooted in the Euler-Maclaurin summation formula, that for certain bell-shaped integrands, a simple block-function approximation to the integral is much more accurate than one would normally expect [2, pg. 180]. This principle is exploited in the tanh-sinh scheme by transforming an integral of a given function $f(x)$ on a finite interval such as $[-1,1]$ to an integral on $(-\infty,\infty)$, by using the change of variable $x = g(t)$, where $g(t) = \tanh(\pi/2 \cdot \sinh t)$. The function $g(t)$ has the property that $g(x) \to 1$ as $x \to \infty$ and $g(x) \to -1$ as $x \to -\infty$, and also that $g'(x)$ and all higher derivatives rapidly approach zero for large positive and negative arguments. Thus one can write, for $h > 0$,

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

where $x_j = g(hj)$ and $w_j = g'(hj)$ and $E(h)$ is an error term. In many cases, even where $f(x)$ has an infinite derivative or an integrable singularity at one or both endpoints, the resulting integrand $f(g(t))g'(t)$ is a smooth bell-shaped function for which the Euler-Maclaurin argument applies. In these cases, the error $E(h)$ decreases very rapidly with h (faster than any power of h).

Thus one can approximate the integral of $f(t)$ on $[-1,1]$ by the sum $\sum_{j=-N}^{N} w_j f(x_j)$, where the abscissas are given by $x_j = g(hj)$ and the weights are given by $w_j = g'(hj)$. In our implementation, the parameter h is set to 2^{-k}, where k is the "level" of the quadrature calculation, and N is chosen large enough that terms beyond N are smaller

than the "epsilon" of the arithmetic precision being used. Successively larger levels reduce h in half, double the number of abscissa-weight pairs (and thus double the number of function evaluations required in a quadrature calculation), but also approximately double the number of correct digits in the result, in many cases. The abscissa-weight pairs of one level are the even-indexed pairs for the next level. Full details are given in [7]. The basic tanh-sinh integration scheme was first introduced by Takahasi and Mori [13].

3. High-Precision Arithmetic

The Arbitrary Precision (ARPREC) computation library was used to perform the required high-precision arithmetic computations described in this paper [6]. This software library is written in C++, but it includes both C++ and Fortran-90 translation modules, so that existing C++ and Fortran-90 application programs can utilize this library by making only very minor changes to the source code. In most cases, it is only necessary to change type statements and input/output statements of the variables that one wishes to be treated as arbitrary precision, and all other operations are automatically performed by the library. One fortunate feature of high-precision numerical quadrature, as described in this paper, is that individual high-precision arithmetic operations and transcendental function evaluations can be performed locally on a single processor. Thus it is not necessary to invoke parallel processing within the ARPREC library itself, at least for the problems considered below.

4. 1-D Parallel Implementation

Among its virtues, the tanh-sinh quadrature scheme is very well suited for implementation on a highly parallel computer system, using either a MPI or OpenMP programming model, although there are several details that must be observed to avoid major reductions in parallel performance.

Two approaches were considered for a parallel implementation: (1) computing abscissa-weight pairs in parallel, distributing all resulting pairs to all processors, and then parceling out function evaluations to processors in some evenly distributed manner; and (2) calculating the abscissa-weight pair indexed j only on processor $\rho(j)$ (for some processor assignment function $\rho(j)$), and performing only those function evaluations corresponding to pairs indexed j on processor $\rho(j)$. (Some additional details, such as the error estimation procedure, are not mentioned in this brief sketch of the parallelization strategy.) Either way, function-weight products are summed locally, then these sums are combined onto a single processor and added together using high-precision arithmetic to produce a global sum. Option (1) requires substantial interprocessor communication, and in the tests below is scalable only to about 256 processors. It also requires a large amount of memory on each node. Option (2) requires much less communication and only a modest amount of memory on each node, but suffers from severe load imbalances (and reductions in scalability) unless the processor assignment function $\rho(j)$ is chosen carefully.

One straightforward processor assignment scheme is a simple cyclic scheme, namely $\rho(j) = j \bmod m$, where m is the total number of processors. One difficulty with this scheme derives from the fact that different integration problems require different numbers

		CYC		BC1		BC2		MCBC		RAND	
Proc.	Stride	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
16	1	4375	4375	4368	4384	4369	4386	4374	4376	4267	4505
16	2	0	4375	2184	2192	2056	2322	2187	2188	2133	2247
16	4	0	4375	1092	1096	1028	1290	1093	1094	1037	1135
16	8	0	4375	546	548	514	774	546	547	500	596
16	16	0	4375	273	274	257	516	273	274	246	305
64	1	1093	1094	1088	1104	1088	1105	1093	1095	1022	1169
64	2	0	1094	544	552	512	585	545	548	505	600
64	4	0	1094	272	276	256	325	272	276	242	315
64	8	0	1094	136	138	128	195	136	138	112	176
64	16	0	1094	68	69	64	130	68	69	48	87
256	1	273	274	272	288	272	289	272	274	234	320
256	2	0	274	136	144	128	153	136	137	106	168
256	4	0	274	68	72	64	85	68	69	45	93
256	8	0	274	34	36	32	51	34	35	17	51
256	16	0	274	17	18	16	34	17	18	6	29
1024	1	68	69	64	80	68	85	67	69	42	97
1024	2	0	69	32	40	32	45	32	36	16	54
1024	4	0	69	16	20	16	25	16	20	6	30
1024	8	0	69	8	10	8	15	8	10	0	19
1024	16	0	69	4	5	4	10	4	5	0	12

Table 1: Min/max processor counts for five assignment functions (70,000 indices)

of abscissa-weight pairs to achieve a given accuracy target. Even among the problems described in the next section, some achieve full 2000-digit accuracy with only nine levels of abscissa-weight pairs ($h = 2^{-9}$), while others require 13 levels of abscissa-weight pairs ($h = 2^{-13}$), which means 16 times as many pairs and 16 times as many function evaluations. But if one has pre-computed 13 levels of abscissa-weight pairs, then when only nine levels are used, the abscissa-weight array is accessed with a stride of 16. Such power-of-two strides result in catastrophic load imbalances when a cyclic assignment function is employed—some processors have a large fraction of the pairs and corresponding function evaluations, while other processors literally have none.

It turns out that it is not easy to find an optimal assignment scheme $\rho(j)$ for this application. Several popular "hashing" schemes from the computer science literature were tried but found not to be very effective. Table 1 gives the results of tests of five different assignment schemes. In these tests, 70,000 indices (the approximate number of abscissa-weight pairs actually generated in the quadrature computations described in the Sections 5 and 6) were assigned to processors according to the five schemes. The smallest and largest number of indices assigned to any processor by a given scheme, for various processor numbers and strides, are then shown in the table. The more nearly equal these max and min figures are, the better the assignment scheme. The five assignment schemes are:

1. CYC, a cyclic scheme: $\rho(j) = \mathrm{mod}(j, m)$

2. BC1, a block-cyclic scheme: $\rho(j) = \mathrm{mod}(\lfloor j/16 \rfloor, m)$
3. BC2, a block-cyclic scheme: $\rho(j) = \mathrm{mod}(\lfloor j/17 \rfloor, m)$
4. MCBC, a mixed cyclic, block-cyclic scheme: $\rho(j) = \mathrm{mod}(j + \lfloor j/16 \rfloor, m)$
5. RAND, a pseudo-random scheme: $\rho(j) = \lfloor z_j m \rfloor$, where z_j is a uniform generator on $(0, 1)$.

It can be seen from the results in Table 1 that of the five schemes mentioned, the one named "mixed cyclic, block-cyclic" (MCBC) is the best. It provides a virtually perfect load balance across a large range of processors (up to 1024) and strides (up to stride 16). This is the scheme that was used in the computations described below.

5. 1-D Test Problems

The following 14 integrals are taken from the suite used in the earlier paper [7]. They are typical of the integrals that have been encountered in experimental math research, except that in each of these cases an analytic result is known, as shown below, facilitating the checking of results:

- 1–4: Continuous functions on finite intervals.
- 5–6: Continuous functions on finite intervals, but with an infinite derivative at an endpoint.
- 7–10: Functions on finite intervals with an integrable singularity at an endpoint.
- 11–13: Functions on an infinite interval.
- 14: An oscillatory function on an infinite interval.

$$
\begin{aligned}
1 \;:&\quad \int_0^1 t\log(1+t)\,dt \;=\; 1/4 \\
2 \;:&\quad \int_0^1 t^2 \arctan t\,dt \;=\; (\pi - 2 + 2\log 2)/12 \\
3 \;:&\quad \int_0^{\pi/2} e^t \cos t\,dt \;=\; (e^{\pi/2} - 1)/2 \\
4 \;:&\quad \int_0^1 \frac{\arctan(\sqrt{2+t^2})}{(1+t^2)\sqrt{2+t^2}}\,dt \;=\; 5\pi^2/96 \\
5 \;:&\quad \int_0^1 \sqrt{t}\log t\,dt \;=\; -4/9 \\
6 \;:&\quad \int_0^1 \sqrt{1-t^2}\,dt \;=\; \pi/4 \\
7 \;:&\quad \int_0^1 \frac{\sqrt{t}}{\sqrt{1-t^2}}dt \;=\; 2\sqrt{\pi}\Gamma(3/4)/\Gamma(1/4) \;=\; \beta(1/2, 3/4)/2
\end{aligned}
$$

$$
\begin{aligned}
8 &: \int_0^1 \log t^2 \, dt = 2 \\
9 &: \int_0^{\pi/2} \log(\cos t) \, dt = -\pi \log(2)/2 \\
10 &: \int_0^{\pi/2} \sqrt{\tan t} \, dt = \pi\sqrt{2}/2 \\
11 &: \int_0^{\infty} \frac{1}{1+t^2} \, dt = \pi/2 \\
12 &: \int_0^{\infty} \frac{e^{-t}}{\sqrt{t}} \, dt = \sqrt{\pi} \\
13 &: \int_0^{\infty} e^{-t^2/2} \, dt = \sqrt{\pi/2} \\
14 &: \int_0^{\infty} e^{-t} \cos t \, dt = 1/2
\end{aligned}
$$

6. 1-D Performance Results

The results of the 1-D parallel quadrature tests are given in Table 2. The first line gives the run time, in seconds, for the initialization process (calculating all abscissa-weight pairs). The second column gives the levels of abscissa-weight pairs (see Sections 2 and 4) required to achieve the target accuracy (10^{-2000}) for the individual problems. The corresponding number of abscissa-weight pairs is roughly 8.7×2^k, where k is the number of levels shown in the table. The target accuracy was achieved in each problem, except in Problem 14 where the accuracy was 10^{-1972}. When 14 levels are used, or if a slightly smaller value of h is used, the error target is achieved here also.

These runs were made on "System X," an Apple G5-based parallel supercomputer at Virginia Technical University, using the IBM xlf90 Fortran-90 compiler (for the parallel quadrature application program) and the IBM xlC C++ compiler (for the ARPREC library). The parallel program performs all functions of the single processor code, including the calculation of an estimated error in the result [7]. The one-processor timings shown in Table 2 are for an efficient single-processor version of this program, with no parallel constructs. Thus the speedup ratios shown in the table are true comparisons to single-processor performance. Note that these are not scaled speedup figures—each run, including the 1-CPU run, is for the same full-sized problem. Note also that these timings exhibit super-linear speedup up to 256 CPUs, but drop back a bit for 1024 CPUs.

6. Two-Dimensional Quadrature

The tanh-sinh scheme described above can be generalized to two or more dimensions. In particular, a 2-D iterated integral can be approximated as follows:

$$
\begin{aligned}
\int_{-1}^{1} \int_{-1}^{1} f(x,y) \, dx \, dy &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(g(s), g(t)) g'(s) g'(t) \, ds \, dt \\
&= h \sum_{k=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} w_j w_k f(x_j, x_k) + E(h),
\end{aligned}
$$

Problem Number	Levels Required	Processors					
		1	4	16	64	256	1024
Init		4329.04	1085.34	271.87	68.88	17.73	5.02
1	10	480.07	101.63	25.55	6.45	1.65	0.53
2	10	1403.63	294.32	74.04	18.83	4.99	1.54
3	10	1421.99	317.01	79.69	20.42	5.24	1.83
4	10	1553.24	328.73	82.13	20.84	5.52	1.63
5	9	236.68	51.62	12.90	3.30	0.93	0.30
6	10	26.31	5.62	1.42	0.36	0.11	0.05
7	10	52.65	11.46	2.87	0.72	0.20	0.10
8	9	234.06	50.98	12.85	3.26	0.90	0.27
9	10	1552.38	333.24	83.60	21.34	5.44	1.84
10	10	1138.78	245.45	61.39	15.73	3.99	1.44
11	11	25.30	5.17	1.30	0.33	0.09	0.04
12	12	655.03	161.99	40.71	10.20	2.65	0.80
13	13	871.99	216.50	54.13	13.65	3.52	0.97
14	13	8291.43	1826.02	457.02	114.84	29.48	7.87
Total		22272.58	5035.08	1261.47	319.15	82.44	24.23
Speedup		1.00	4.42	17.66	69.79	270.17	919.22

Table 2: Parallel run times (in seconds) and speedup ratios for 1-D problems

where $g(t) = \tanh(\pi/2 \cdot \sinh(t))$ as in the 1-D case, and where x_j and w_j are the 1-D abscissas and weights. This same approach can easily be extended to numerically evaluate more general integrals of the form

$$\int_a^b \int_{c(y)}^{d(y)} f(x,y)\,dx\,dy.$$

As before, the Euler-Maclaurin formula asserts that for a certain class of functions $f(x,y)$, including many with infinite derivatives and blow-up singularities at the boundaries of the rectangle, the error $E(h)$ in the above approximation goes to zero faster than any power of h. As a result, 2-D tanh-sinh quadrature, like the 1-D version, often achieves quadratic convergence, wherein each additional level of abscissa-weight pairs yields twice as many correct digits in the result.

However, 2-D quadrature inherently is much more expensive than 1-D quadrature, because the number of function evaluations in a 2-D array, assuming the same overall spacing, is many times larger than in a 1-D problem. Millions of function evaluations may be required to obtain, say, 100-digit accuracy in the result. Also, it has been found that 2-D tanh-sinh scheme is more sensitive to anomalies such infinite derivatives or blow-up singularities at boundaries. In such cases, each additional level typically yields only about 1.4 times as many correct digits, whereas in 1-D quadrature, problems with similar anomalies typically exhibit quadratic convergence (each additional level approximately

doubles the number of correct digits). What's more, in 2-D quadrature, each additional level quadruples the computational cost instead of merely doubling the cost, since four times as many function evaluations are required.

7. 2-D Parallel Implementation

The parallel implementation of the 2-D scheme again relies crucially on a carefully chosen scheme for allocating processors to the abscissa array for function evaluations. The program assigns a batch of 16 consecutively numbered processors to each column, and then assigns the function evaluations in this column among these 16 processors. In this way, the program exploits available parallelism in both dimensions. The particular assignment scheme used by the program is as follows: the 16 processors p that satisfy $\lfloor p/16 \rfloor = \mod(j+j/16, n/16)$ are assigned to column j of the 2-d array of abscissas. Then within column j, location (i, j) is assigned to the processor p that satisfies $\mod(p, 16) = \mod(i + i/16, 16)$. Note that both rows and columns employ a mixed cyclic, block-cyclic scheme, which provides an even load balance for function evaluations, yet avoids difficulties with power-of-two strides.

8. 2-D Test Problems

The serial and parallel implementations of this 2-D tanh-sinh quadrature scheme have been tested on a suite of eight test problems. Because of the much higher computational cost of 2-D quadrature, a more modest goal of 100-digit accuracy was established in these problems, using 120-digit working precision. As before, this set includes some rather difficult examples, including one problem with a non-differentiable point at a boundary (Problem 1), two problems with a blow-up singularity at a boundary (Problems 4 and 6), two problems where the inner integral boundary is not merely an interval but instead bounded by two functions (Problems 7 and 8), and one problem on an infinite interval (Problem 5).

$$1 \;:\; \int_0^1 \int_0^1 \sqrt{s^2+t^2}\, ds\, dt \;=\; \sqrt{2}/3 - \log(2)/6 + \log(2+\sqrt{2})/3$$

$$2 \;:\; \int_0^1 \int_0^1 \sqrt{1+(s-t)^2}\, ds\, dt \;=\; -\sqrt{2}/3 - \log(\sqrt{2}-1)/2 + \log(\sqrt{2}+1)/2 + 2/3$$

$$3 \;:\; \int_{-1}^1 \int_{-1}^1 (1+s^2+t^2)^{-1/2}\, ds\, dt \;=\; 4\log(2+\sqrt{3}) - 2\pi/3$$

$$4 \;:\; \int_0^\pi \int_0^\pi \log[2-\cos s - \cos t]\, ds\, dt \;=\; 4\pi G - \pi^2 \log 2$$

$$5 \;:\; \int_0^\infty \int_0^\infty \sqrt{s^2+st+t^2}\, e^{-s-t}\, ds\, dt \;=\; 1 + 3/4 \cdot \log 3$$

$$6 \;:\; \int_0^1 \int_0^1 (s+t)^{-1}[(1-s)(1-t)]^{-1/2}\, ds\, dt \;=\; 4G$$

$$7 \;:\; \int_0^1 \int_0^t (1+s^2+t^2)^{-1/2}\, ds\, dt \;=\; -\pi/12 - 1/2 \cdot \log 2 + \log(1+\sqrt{3})$$

$$8 \;:\; \int_0^\pi \int_0^t (\cos s \sin t) e^{-s-t}\, ds\, dt \;=\; 1/4 \cdot (1+e^{-\pi})$$

Problem Number	Levels Required	Processors				
		1	16	64	256	1024
1	9	1246.26	96.42	24.66	7.05	3.33
2	6	19.03	1.52	0.46	0.27	0.73
3	7	82.79	6.56	1.91	0.64	1.17
4	9	15310.44	1194.52	305.11	81.88	24.40
5	9	2209.86	170.84	44.38	12.23	4.62
6	9	1552.87	120.86	30.80	8.67	3.37
7	6	21.79	1.72	0.54	0.28	0.73
8	6	113.04	8.90	2.87	1.08	1.51
Total		20556.08	1601.34	410.73	112.10	39.86
Speedup		1.00	12.84	50.05	183.37	515.71

Table 3: Parallel run times (in seconds) and speedup ratios for 2-D problems

9. 2-D Performance Results

Performance results for the 2-D quadrature program are shown in Table 2. In each problem over 100-digit accuracy, except for Problems 4 and 6, where the errors were 10^{-86} and 10^{-80}, respectively. No results are shown in this table for four processors, since the parallel program assumes a minimum of 16 processors.

It is clear from these results that, unlike the 1-D case, there is a large difference in run times between well-behaved integrands and those with singularities at a corner or boundary. Those without such anomalies can be evaluated to over 100-digit accuracy with just six levels, requiring only a few minutes run time. For those problems that do exhibit such anomalies, nine levels are needed, requiring many more function evaluations. Indeed, for Problems 4 and 6, even nine levels of abscissa-weight pairs evidently were not sufficient—it appears that one additional level would be required in each case to achieve over 100 digit accuracy, which would multiply the run times by a factor of four.

The parallel speedups for 2-D quadrature are not nearly as high as for 1-D quadrature, in part because features such as estimated error calculation are significantly more complicated than in the 1-D case, and also because it is more difficult to allocate tasks evenly in 2-D. Also, much of this reduction in scalability occurs in the shorter-running problems, whose modest computational work cannot be as efficiently distributed among 1024 processors.

10. Confirmation of a Conjectured Identity to 20,000 Digits

As mentioned in the Introduction, one of the present authors (Borwein) and David Broadhurst found the following conjectured identity:

$$\frac{24}{7\sqrt{7}} \int_{\pi/3}^{\pi/2} \log \left| \frac{\tan t + \sqrt{7}}{\tan t - \sqrt{7}} \right| dt \stackrel{?}{=} L_{-7}(2) = \tag{1}$$
$$\sum_{n=0}^{\infty} \left[\frac{1}{(7n+1)^2} + \frac{1}{(7n+2)^2} - \frac{1}{(7n+3)^2} + \frac{1}{(7n+4)^2} - \frac{1}{(7n+5)^2} - \frac{1}{(7n+6)^2} \right].$$

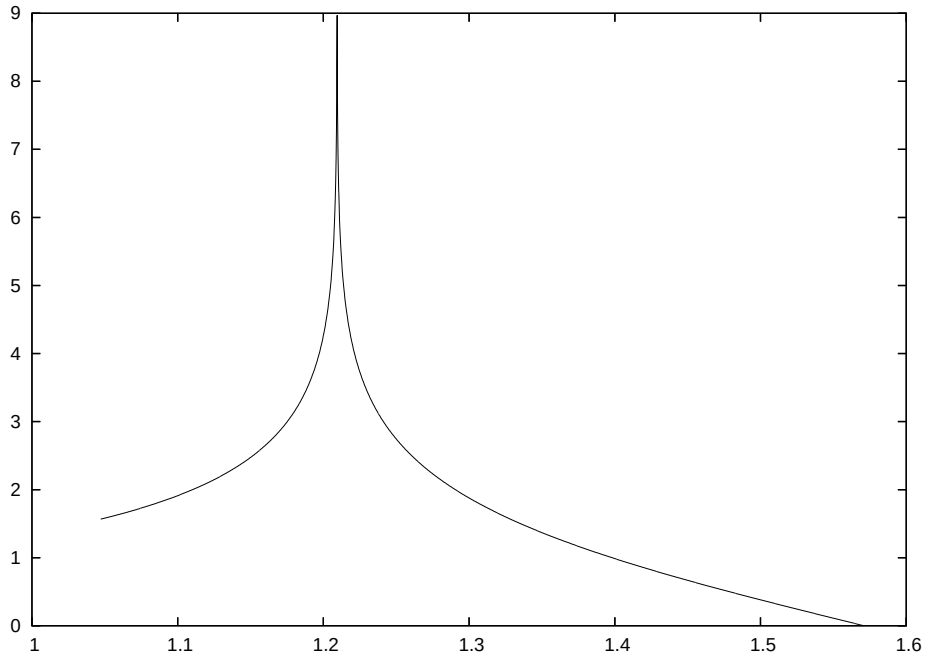

Figure 1: Integrand function with singularity

This integral arose out of some studies in quantum field theory, in analysis of the volume of ideal tetrahedra in hyperbolic space. Note that the integrand function has a nasty singularity at $t = \arctan(\sqrt{7})$ (see Figure 1). The question mark is used because no formal proof is yet known. We note that Richard Crandall [10] has observed that the right-hand expression $L_{-7}(2)$ is also given by the integral

$$L_{-7}(2) \;=\; -\int_0^1 \frac{1+2u+u^2+2u^3+u^4}{1+u+u^2+u^3+u^4+u^5+u^6}\,\log u\,du. \tag{2}$$

Because of the interest expressed by researchers in the above conjecture and some related conjectures [8], we decided to calculate the integral

$$\frac{24}{7\sqrt{7}}\int_{\pi/3}^{\pi/2}\log\left|\frac{\tan t+\sqrt{7}}{\tan t-\sqrt{7}}\right|dt \;=\; 1.15192547054449104710169239732054996\ldots$$

to 20,000-digit accuracy (which approaches the limits of presently feasible computation) and compare with a 20,000-digit evaluation of the six-term infinite series on the right-hand side of (1). This integral was evaluated by splitting it into two integrals, the first from $\pi/3$ to $\arctan(\sqrt{7})$, and the second from $\arctan(\sqrt{7})$ to $\pi/2$, and then applying the 1-D tanh-sinh scheme to each part. This test was successful—the numerical value of the integral on the left-hand side of (1) agrees with the numerical value of the six-term infinite series on the right-hand side of (1) to at least 19,995 digits. The infinite series in was evaluated in approximately five hours on a personal computer using *Mathematica*.

CPUs	Init	Integral #1	Integral #2	Total	Speedup
1	*190013	*1534652	*1026692	*2751357	1.00
16	12266	101647	64720	178633	15.40
64	3022	24771	16586	44379	62.00
256	770	6333	4194	11297	243.55
1024	199	1536	1034	2769	993.63

Table 4: Parallel run times (in seconds) and speedup ratios for the 20,000-digit problem

Efficiently performing any numerical computation to 20,000-digit accuracy requires advanced techniques. The key idea here is to note that high-precision multiplications can be evaluated using a linear convolution scheme, which in turn can be performed using fast Fourier transforms. High-precision divisions can then be done using Newton iterations. Quadratically convergent algorithms (where every iteration roughly doubles the number of correct digits) are known for most transcendental functions. These advanced algorithms are incorporated in the ARPREC multi-precision package and are automatically invoked when a numeric precision of greater than about 2,000 digits is specified [6]. Aside from precision level, the parallel integration was performed exactly as described in Section 4. Thirteen levels (approximately 85,000) abscissa-weight pairs were required to achieve the target accuracy in this problem.

Performance results on the Virginia Tech system are shown in Table 4. Multi-CPU timings include barrier waits and communication operations, as before. The one-CPU timings (noted by asterisks) are the sums of individual process timings in a 64-CPU run, in which only local computation was timed—barrier waits and communication operations were not timed. Based on tests on other problems and problem sizes, these summed timings are accurate estimates of the timings of a true 1-CPU run, which would have taken 32 days in this case. In these runs, as well as the others reported in the paper, timing variations of up to 6% have been noted in tests. The performance rate for the 1024-CPU run is 690 Gflop/s, based on a measurement of floating-point operation count done on the Seaborg system at LBNL, using a hardware performance monitoring tool.

As a separate test of our computer program, we also evaluated Crandall's integral (2), which, as noted above, is equal to $L_{-7}(2)$, or in other words the right-hand side of the conjectured identity (1). On 1024 CPUs, our program was able to evaluate this integral to 20,000 digits (which completely agrees with the results above) in 416 seconds runtime, including initialization. This is 6.6 times faster than the timing (2769 seconds) shown in Table 4, even though (2) also has a singularity and requires 25% more function evaluations than (1) to attain 20,000-digit accuracy. The main reason for the lower run time is the fact that the integrand in (2), which lacks trigonometric functions, is faster to evaluate, although it should also be noted that there is only one integral to evaluate here, whereas two are required for (1).

We believe that these two evaluations are the highest-precision non-trivial numerical integrations performed to date.

References

[1] Zafar Ahmed, "Definitely an Integral," *American Mathematical Monthly*, vol. 109 (2002), no. 7, pg. 670–671.

[2] Kendall E. Atkinson, *Elementary Numerical Analysis*, John Wiley and Sons, 1993.

[3] David H. Bailey, Jonathan M. Borwein, Vishaal Kapoor and Eric Weisstein, "Ten Problems in Experimental Mathematics: A Challenge," manuscript, 2004, available at http://crd.lbl.gov/~dhbailey/dhbpapers/tenproblems.pdf.

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

[5] David H. Bailey and David J. Broadhurst, "A Seventeenth-Order Polylogarithm Ladder," manuscript, 1999, availabe at http://crd.lbl.gov/~dhbailey/dhbpapers/ladder.pdf.

[6] David H. Bailey, Yozo Hida, Xiaoye S. Li and Brandon Thompson, "ARPREC: An Arbitrary Precision Computation Package," technical report LBNL-53651, software and documentation available at http://crdl.bl.gov/~dhbailey/mpdist.

[7] David H. Bailey, Xiaoye S. Li and Karthik Jeyabalan, "A Comparison of Three High-Precision Quadrature Programs," manuscript, available at http://crd.lbl.gov/~dhbailey/dhbpapers/quadrature.pdf.

[8] J. Borwein and D. Broadhurst, "Determination of Rational Dirichlet-zets Invariants of Hyperbolic Manifolds and Feynman Knots and Links," available at http://arxiv.org/hep-th/9811173.

[9] J. Borwein, J. Zucker and J. Boersma, "Evaluation of Character Euler Double Sums," D-DRIVE Preprint #260, 2004, available at http://users.cs.dal.ca/~jborwein/bzb7.pdf.

[10] Richard E. Crandall, private communication, April 2005.

[11] William Gropp, Ewing Lusk, Anthony Skjellum, *Using MPI: A Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, 1996.

[12] Yozo Hida, Xiaoye S. Li and David H. Bailey, "Algorithms for Quad-Double Precision Floating Point Arithmetic," *15th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society, 2001, pg. 155–162.

[13] H. Takahasi and M. Mori, "Double Exponential Formulas for Numerical Integration," Publications of RIMS, Kyoto University, vol. 9 (1974), pg. 721–741.