New Algorithms for Improved Transcendental Functions on IA-64

Shane Story
Intel Corporation
Hillsboro, OR 97124
shane.story@intel.com

Ping Tak Peter Tang
Intel Corporation
Santa Clara, CA 95052

Abstract

The IA-64 architecture provides new opportunities and challenges for implementing an improved set of transcendental functions. Using several novel polynomial-based table-driven techniques, we are able to provide new algorithms for the transcendental functions. Major improvements include an accuracy level of about 0.6 ulps (units in the last place) and forward trigonometric functions that have a period of 2π. The accuracy enhancements are achieved at improved speed, yet without an increase in the table size. In this paper, we highlight the key IA-64 architectural features that influenced our designs and explain the main ideas used in our new algorithms.

1 Introduction

The IA-32 architecture [5] provides hardware instructions to compute a small but important set of transcendental functions – F2XM1, FYL2X, FYL2XP1, FSIN, FCOS, FSINCOS, FPTAN, and FPATAN. The IA-64 architecture, based on the EPIC (Explicitly Parallel Instruction Computing) technology, incorporates a combination of speculation, predication, and explicit parallelism into the basic instruction set and programming control structure. IA-64 was designed to fully support IA-32 applications, so we needed to design and implement new algorithms to support the higher-level IA-32 transcendental function calls. Our primary focus is on these hardware supported instructions, although the methods described can be easily extended to mathematical functions done traditionally in software. This new architecture provides us with an opportunity to further improve the quality of the hardware supported transcendental functions and those done in software.

Our rationale for these improvements is thus. While correctly rounded implementations are ideal, they are unattainable at present within practical speed and resource limits. We aim to minimize the worst-case error under round-to-nearest mode to as close to 0.5 ulps as we see possible, maximize the speed, and limit resource requirements. Therefore, a worst-case error below 0.6 ulps is an improvement when compared to that of 1 ulp in the Pentium™ generation (see [9]). Since these functions do not always yield a correctly rounded result, invoking them twice under two directed rounding modes does not always produce an interval that includes the true mathematical value. In pursuit of more accurate results, we also choose to use a precise argument reduction scheme within the trigonometric functions.

The IA-64 features that are most relevant to the algorithmic design are (1) FMA - fused multiply-add based +, –, ×, and ÷, the last operation with the aid of an approximation to the reciprocal followed by a Newton-Raphson iteration, (2) an internal 82-bit format (1-bit sign, 17-bit exponent, and a 64-bit significand with no hidden bit). For a discussion of other important IA-64 architectural features, see [2].

When approximating transcendental functions, it is sometimes advantageous to have access to the extra bits of precision that result from the basic operations. The IA-64 hardware does not provide a mechanism to access any extra bits of precision. This constraint poses a major challenge for accurate function implementations if fast high-precision simulations are needed (see [8]).

We begin in Section 2 with a discussion of some modifications to a conventional table-driven approach. We apply these modifications to algorithms in the exponential family. Argument reduction for the trigonometric functions is described in Section 3.1. Sections 3.2 and 3.3 outline the algorithms used for the sine and

cosine functions. The algorithms for tangent and cotangent that use both a table driven approach and an iterative division scheme are outlined in Sections 3.4 and 3.5. Section 4 is devoted to the inverse tangent. Section 5 compares the accuracy of these new functions with their counterparts on the Pentium™ processor. In Section 6 we offer our concluding remarks.

We assume throughout this paper that the input arguments discussed are IA-32 80-bit floating-point numbers of the form

$$(-1)^s\, 2^e\, 1.z_1z_2z_3z_4\ldots z_{63},$$

where s = 0 or 1, $-16382 \le e \le 16383$, and the z's = 0 or 1. We refer to values of this type as working-precision numbers.

2 The Exponential Family

The IA-64 architecture has three instructions of the exponential type: F2XM1, FYL2X, and FYL2XP1 which approximate $2^x - 1$, $y\log_2(x)$, and $y\log_2(1 + x)$, respectively. We discuss only the algorithms used for computing $2^x - 1$ and $y\log_2(1 + x)$, as the latter subsumes the function $y\log_2(x)$. The approximation to $\log_2(1 + x)$ is a crucial part of the computation and will be a primary focus.

Usually we require only simple polynomial approximations for regions near the functions' roots – for each of these functions, x = 0 is the only root. In regions bounded away from the roots, we employ well-known table-driven methodologies (see [11], [12], and [13]). With the help of function values tabulated beforehand at suitably chosen *breakpoints*, function evaluations are transformed into calculations at *reduced arguments* that are relatively small in magnitude. Consequently, the calculations at the reduced arguments are usually faster and easier to perform.

2.1 Simple Polynomial Approximations

One difficulty here is that we are dealing with the base 2 and not the base e family of functions. Near the root of $f(x) = 2^x - 1$ and $\log_2(1 + x)$, the Taylor's series representations are of the form

$$f(x) = \rho_1 x + \rho_2 x^2 + \rho_3 x^3 + \ldots,$$

where the leading coefficient is either ln(2) or 1/ln(2), neither of which is an exact machine number. Thus, direct polynomial approximations formulated with machine numbers for the ρ_j's will have a relative accuracy limited to roughly 2^{-64} as x tends to zero. Utilizing polynomials of the form

$$p_0 x + p_1 x + p_2 x^2 + \ldots + p_N x^N$$

surmounts this difficulty wherein ln(2) and 1/ln(2) are approximated accurately by the sum of the machine numbers p_0 and p_1. We obtain these special polynomials using a Remez algorithm [1].

To complete the approximation to $f(x)$, we evaluate the polynomial,

$$q \approx p_1 x + p_2 x^2 + \ldots + p_N x^N$$

using a variant of Horner's rule and perform a last step, $p_0 x + q$, with a single fused multiply-add. This sequence of operations simulates the effect of having stored highly accurate values for ln(2) and 1/ln(2).

2.2 Table Driven Algorithms

Typical table-driven approaches [11], [12], [13] for $f(x) = 2^x - 1$ or $\log_2(1 + x)$ usually require us to tabulate beforehand, $f(B)$, where B is the leading part of x or 1 + x, respectively. The core approximations are then tailored to the calculation of $f(r)$ for some r small in magnitude. Specifically, for $f(x) = 2^x - 1$

$$f(x) = f(B) + (f(B) + 1)f(x - B),$$

and for $f(x) = \log_2(1 + x)$,

$$f(x) = f(B - 1) + f((1 + x - B)/B).$$

However, these table-driven methods will not allow us to meet our goals entirely. Our objective is to achieve an accuracy level of about 0.6 ulps as efficiently as possible, which means that x – B and 1 + x – B need to be on the order of 2^{-10}. This would in turn require extremely large tables, tables too large to be accommodated easily on our target platform.

We overcome this obstacle in two ways. First, the two functions in question have properties that permit us to construct a table with 2^K entries from several smaller tables. These smaller tables

are of the sizes 2^L, 2^M, and 2^N, where $K = L + M + N$. Second, we formulate a strategy for each approximation that facilitates an exact calculation of its dominant part.

Consider, for example, $f(x) = 2^x - 1$ where $1/16 \le |x| < 1$. In its unnormalized form, x is actually

$$x = (-1)^s\, 0.b_1\, b_2\, b_3 \ldots b_{67}.$$

Instead of using a table with 2^{13} entries that correspond to breakpoints of the form

$$B = (-1)^s\, 0.b_1\, b_2\, b_3\, b_4 \ldots b_{12}\, 1$$

so that $|x - B| \le 2^{-13}$, we define instead B_1, B_2, and B_3 such that

$$B_1 = (-1)^s\, 0.b_1b_2b_3b_4,$$
$$B_2 = (-1)^s\, 0.\, 0\, 0\, 0\, 0\, b_5b_6b_7b_8, \text{ and}$$
$$B_3 = (-1)^s\, 0.\, 0\, 0\, 0\, 0\, 0\, 0\, 0\, 0\, b_9b_{10}b_{11}b_{12}1.$$

Clearly, $|x - (B_1 + B_2 + B_3)| \le 2^{-13}$. In principle, this example requires that we need only tabulate

$$f(B_j) + 1,\ j = 1, 2, 3, \text{ for}$$

$$f(B_1+B_2+B_3) = (f(B_1)+1)(f(B_2)+1)(f(B_3)+1) - 1.$$

Thus the table size is reduced to 3×2^5 from 2^{13}. We require an extra factor of 2 because table entries are needed for both the positive and negative breakpoints.

Unfortunately, this technique alone does not allow us to represent $f(B_j) + 1$ exactly in working precision. Obtaining $f(B_1 + B_2 + B_3)$ to high accuracy would require not only that each $f(B_j) + 1$ be stored to extra precision, but also that additional extra-precise products be computed.

So, we complement our first technique with a second one. Instead of using the B_j's as our breakpoints, we use the slightly perturbed points, β_j, defined by

$$\beta_j = B_j - \delta_j = \log_2([f(B_j) + 1]_{21}),$$

where $[\alpha]_{21}$ maps α to 21 significant bits by a well-defined rounding rule.

We define $T_j = f(\beta_j) + 1$ in 21 significant bits with the rounding rule $T_j = 1$ if $B_j = 0$, or $T_j = 2^{Bj}$ rounded up or down to 21 significant bits depending on whether $x > 0$ or $x < 0$, respectively. The δ_j's, which are small in magnitude, are computed to working precision beforehand and stored alongside the T_j's as d_j's. We note that this method is different from the *accurate tables method* proposed in [3]. There, special breakpoints are selected so that the table values can almost be represented in working precision. Our requirement that the table values be almost 21 significant bits is too restrictive for that method.

The reduced argument,

$$\gamma = x - (\beta_1 + \beta_2 + \beta_3),$$

is computed by

$$r = [x - (B_1 + B_2 + B_3)] + (d_1 + d_2 + d_3).$$

Because r is so small, $|r| \le 2^{-13}$, the rounding errors associated with its computation are harmless. Finally, we compute $f(x)$ by

$$f(x) = f(\beta) + (f(\beta) + 1)f(\gamma),$$
$$\approx (T - 1) + Tp(r),$$

where $\beta = \beta_1 + \beta_2 + \beta_3$, $T = T_1T_2T_3$, and $p(r)$ is a short polynomial approximation to $f(r)$. Note that T is represented exactly in 64 significant bits using two multiplies, and the difference, $T - 1$, is error free because of cancellation.

We now consider $g(x,y) = y\log_2(1 + x)$, which provides a foundation to an implementation of the instructions FYL2X and FYL2XP1. To begin, we apply some basic transformations. Because x is a working-precision number, we are able to represent $1 + x$, exactly as $2^N(Z + z)$, whereby Z and z are two working-precision numbers that satisfy $1 \le Z + z < 2$, $|Z| \ge 2^{63}|z|$, and

$$Z = 1.\, w_1\, w_2 \ldots w_{63}.$$

We focus first on $\log_2(Z + z)$. As before, typical table-driven methods define B as a leading part of Z, for example,

$$B = 1.\, w_1\, w_2\, w_3 \ldots w_9\, 1,$$

and use the identity

$$\log_2(Z + z) = \log_2(B) + \log_2((Z + z)/B).$$

Both $\log_2(B)$ and $1/B$ are then retrieved from a table, and the second term is evaluated using a

short polynomial because (Z + z)/B is very close to 1. The table size needed for this scheme would be about 2^{10} – again, too large to meet our design goals.

We apply our special table-driven methodology and technique for assuring accuracy by defining B_1 and β_1 as

$$B_1 = 1.w_1w_2w_3, \quad \text{and} \quad \beta_1 = 1/[1/B_1]_{21}.$$

We tabulate $G_1 = 1/\beta_1 = [1/B_1]_{21}$ and $\log_2(\beta_1)$ as a single-precision constant T_1 with a double-precision correction t_1. The following holds exactly:

$$\log_2(Z + z) = \log_2(\beta_1) + \log_2(G_1(Z + z)).$$

In principle, G_1Z approximates 1 to within 2^{-3}. β_1 is defined carefully by directed rounding so that the seven high-order bits of G_1Z are $1.000y_4y_5y_6$. We repeat this process a second time by defining

$$B_2 = 1.000y_4y_5y_6, \text{ and } \beta_2 = 1/[1/B_2]_{21}.$$

This yields $G_2 = 1/\beta_2$ and a pair, (T_2,t_2). The technique is repeated a final time, with

$$B_3 = 1.000000z_7z_8z_91, \text{ and } \beta_3 = 1/[1/B_3]_{21}.$$

This results in $G_3 = 1/\beta_3$ and a pair, (T_3,t_3). Thus, the exact relationship,

$$\log_2(Z + z) = \log_2(\beta) + \log_2(G(Z+z)),$$

is established, where $\beta = \beta_1\beta_2\beta_3$, and $G = G_1G_2G_3$. G is in our case computed exactly. With the use of two fused multiply-adds, we are able to approximate the exact reduced argument,

$$\gamma = G(Z + z) - 1,$$

with $r = (GZ - 1) + Gz$. We know that when computed, $|r| < (1.2)2^{-10}$.

We let $T = T_1 + T_2 + T_3$, which can be computed exactly, and $t = t_1 + (t_2 + t_3)$, whose rounding error is small. The final approximation to $f(x)$ is

$$\begin{aligned} f(x) &= y\log_2(1 + x), \\ &= y(N + \log_2(\beta) + \log_2(1 + \gamma)), \\ &\approx y((N + T) + (p(r) + t)) \\ &\approx y(N + T) + (y(p(r) + t)), \end{aligned}$$

where p(r) is a 6^{th} degree polynomial approximation to $\log_2(1 + r)$.

3 The Forward Trigonometric Family

The forward trigonometric family consists of the instructions FSIN, FCOS, FSINCOS, and FPTAN, that approximate the sin(x), cos(x), both sin(x) and cos(x), and tan(x), respectively, for arguments x such that $|x| < 2^{63}$. These functions share a common argument reduction scheme that maps x into $[-\pi/4, \pi/4]$ by the relationship,

$$\gamma = x - N(\pi/2); \quad |\gamma| \le \pi/4.$$

Depending on the value of N modulo 4 (or 2 in the case of tangent), simple trigonometry shows that sin(x) and cos(x) are equivalent to either the positive or negative $\sin(\gamma)$ or $\cos(\gamma)$; and the computation of tan(x) equivalent to either the positive or negative $\tan(\gamma)$ or $\cot(\gamma)$. Our challenge lies in computing differences of the form $x - N(\pi/2)$ to high relative accuracy. We present first our new algorithm that computes a pair of working precision numbers, r and a correction c, so that $|r + c| \le \pi/4 + 2^{-15}$ and

$$|(r + c) - [x - N(\pi/2)]| \le 2^{-72} |x - N(\pi/2)|.$$

We follow that with a discussion of the computation of $f(r + c)$ for $f(x)$ equal to sin(x), cos(x), tan(x), and cot(x).

3.1 Trigonometric Argument Reduction

The reduction of an argument when it is of moderate magnitude, $\pi/4 \le |x| < 2^{24}$, is relatively straightforward. We first need to determine the accuracy needed in the approximation to $\pi/2$ because this is directly related to the magnitude of the final reduced argument. Using techniques similar to those described in [10], we find that for $\pi/4 \le |x| < 2^{63}$, $|x - N(\pi/2)| \ge 2^{-70}$ for all integers N. Therefore, if we approximate $\pi/2$ by three working-precision numbers P_1, P_2, and P_3, we have $|\pi/2 - (P_1 + P_2 + P_3)| < 2^{-194}$. It follows that for $|x| < 2^{24}$,

$$|[x - N(\pi/2)] - [x - N(P_1 + P_2 + P_3)]| < 2^{-170}.$$

This error is tiny enough for even the smallest reduced arguments. Moreover, a similar argument shows that as long as the reduced argument's magnitude is at least 2^{-30}, P_1 and P_2 suffice.

The crux is then to compute either $x - N(P_1 + P_2)$ or $x - N(P_1 + P_2 + P_3)$ so that r + c provides an accurate approximation, where

$$N = \text{nearest-integer}(x \text{ (two-by-pi)}),$$

and two-by-pi is $2/\pi$ rounded to working-precision. A fused multiply-add generates $U = x - NP_1$ exactly due to cancellation because N approximates the integer part of x/P_1. If $|U| \geq 2^{-30}$, we compute $V = NP_2$ and approximate U + V accurately by r + c. Otherwise, we first compute $N(P_2 + P_3)$ accurately as V + W using a fused multiply-add. We are able to then approximate U + V + W to well beyond working-precision in two pieces, r + c.

For large arguments, $2^{24} \leq |x| < 2^{63}$, we introduce a non-traditional approach. Since all trigonometric functions have a period of 2π,

$$\text{trig}(x) = \text{trig}(x - 2\pi k)$$

for any integer k. Furthermore, some IEEE double-extended numbers are known to closely approximate large integral multiples of 2π. The new procedure requires that we first find the remainder of x with respect to one such close approximation, Γ to the integral multiples of 2π, by computing

$$y = x - M\Gamma, \ |y| \leq \Gamma/2.$$

In our implementation, we use the value

$$\Gamma = 4016C84D32B0CE81B9F1$$

in IA-32 80-bit format which is very close to 4178442π. Note that y is a machine number. Because the resulting y satisfies $|y| < 2^{24}$, we can use the reduction method discussed previously. One need only compensate for the fact that $\Gamma = 2\pi k + \delta$, $|\delta| < 2^{-63}$, is not an exact integer multiple of 2π as follows:

$$y = x - M\Gamma = (x - 2\pi Mk) - M\delta,$$

or in general terms

$$\text{trig}(x) = \text{trig}(x - 2\pi Mk) = \text{trig}(y + M\delta).$$

Hence, we adjust the reduced argument obtained from y by adding $M\delta$. The approximation to δ is computed beforehand and is stored as the two working-precision numbers, d_1 and d_2. Md_1 alone suffices for reduced argument greater than 2^{-14} in magnitude. Otherwise, Md_2 is also used. In summary, we obtain a pair of working precision numbers, r and a correction c, such that

$$|(r + c) - [x - N(\pi/2)]| \leq 2^{-72} \, |x - N(\pi/2)|$$

for some integer N where N mod 4 is known. Note that N may not be the exact nearest integer when the value, $x(2/\pi)$, is close to the middle of two consecutive integers. Nevertheless, we know that $|r + c| \leq \pi/4 + 2^{-15}$.

3.2 Computation of Sin(r+c)

The computation is simple when $|r + c|$ is small. A basic approximation,

$$r + c - r^3/6 + r^5/120,$$

suffices when $|r + c| < 2^{-14}$. In fact, when $|r + c| < 2^{-30}$, the 5th degree term is not required. A more general approximation to sin(r + c) is

$$\begin{aligned} \sin(r + c) &\approx r + c + s_1r^3 + s_2r^5 + ... + s_kr^{2k} \\ &\approx r + s_1r^3 + (r^5 \, p(r^2) + c). \end{aligned}$$

This approximation and its computation is straightforward even up to $|r| < 2^{-3}$. Rounding errors become more significant when $|r| \geq 2^{-3}$ because of the magnitude of s_1r^3. For this reason, we must compute $r + s_1r^3$, the critical part of the polynomial approximation, with great care. We decompose r into two pieces, $r_{hi} + r_{lo}$, so that r_{hi} has the 10 most significant bits of r. Likewise, the coefficient s_1 is broken into two pieces, $s_{hi} + s_{lo}$, whereby s_{hi} is comprised of the 16 most significant bits. This allows us to compute $A = r + s_{hi}(r_{hi})^3$ without error. The value $a = s_1r^3 - s_{hi}(r_{hi})^3$ must still be evaluated in working precision to within a few rounding errors. We do this using

$$\begin{aligned} a &= s_{hi}(r^3 - r_{hi}^3) + s_{lo}r^3 \\ &= s_{hi}r_{lo}(r^2 + rr_{hi} + r_{hi}^2) + s_{lo}r^3. \end{aligned}$$

Hence, the precise approximation is given by

$$\sin(r + c) \approx A + [a + (r^5p(r^2) + c)].$$

3.3 Computation of Cos(r+c)

The computation is similar in spirit to that of the sin(r + c). The approximation, $1 - r^2/2 + r^4/24$, suffices for $|r + c| < 2^{-14}$. The 4th degree term

can be ignored when $|r + c| < 2^{-30}$. Because c is in general quite small, cos(r + c) is close to

$$\cos(r) - \sin(r)c$$

which we in turn approximate with a polynomial

$$1 - r^2/2 + c_2r^4 + ... + c_kr^{2k} - \sin(r)c.$$

We can express this polynomial more simply as

$$1 - r^2/2 + r^4q(r^2) - \sin(r)c.$$

When $|r| < 2^{-3}$, we use $-(r)c$ to approximate $-\sin(r)c$ and compute the rest of the polynomial with ease. When $|r| \geq 2^{-3}$, we approximate $-\sin(r)c$ by $-c(r - r^3/6)$ and compute the dominant part, $1 - r^2/2$, carefully. As before, r is decomposed into r_{hi} and r_{lo}, and the crucial part of the result, $A = 1 - r_{hi}^2/2$, is therefore computed exactly. We need a correction term, a, and calculate it using

$$a = r_{hi}^2/2 - r^2/2 = -r_{lo}(r_{hi} + r)/2.$$

Hence, the final approximation to cos(r + c) is a composite, made up of a number of separate pieces;

$$\cos(r + c) \approx A + [a + (r^4q(r^2) - c(r - r^3/6))].$$

3.4 Computation of Tan(r+c)

The approximation $r + c + r^3/3 + 2r^5/15$ suffices when $|r + c| \leq 2^{-14}$. The 5th degree term is unnecessary when $|r + c| < 2^{-30}$.

For larger r's, $|r| < 2^{-2}$, we use a high-degree polynomial of the form

$$\begin{aligned} \tan(r + c) &\approx \tan(r) + \tan'(r)c \\ &\approx \tan(r) + (1 + r^2)c \\ &\approx r + p_1r^3 + ... + p_9r^{19} + (1 + r^2)c. \end{aligned}$$

The computation is straightforward because the rounding errors incurred by the high-order terms beyond p_1r^3 are negligible.

For $|r| \geq 2^{-2}$, we use a table driven method. Note that $\tan(r + c) \approx \tan(r) + \sec^2(r)c$ and that slight inaccuracies are allowed in $\sec^2(r)c$ because of its small magnitude. Moreover, $\tan(-x) = -\tan(x)$. So to begin, we concentrate on calculating tan(|r|) and the dominant part of $\sec^2(r)c$. Denote $|r|$ by

$$|r| = 2^k(1.b_1\, b_2\, b_3 \ldots b_{63}) \qquad k = -1, -2.$$

We define the breakpoint B by

$$B = 2^k(1.b_1\, b_2\, b_3\, b_4\, b_5\, 1)$$

and recognize that

$$|r| = B + y; \qquad |y| \leq 2^{k-6} \leq 2^{-7}.$$

The goal of this table-driven algorithm is to express tan(|r|) as tan(B) + *g*(B,y), whereby the computation of the function *g* is relatively fast and $|g(B,y)|$ is significantly smaller than $|\tan(B)|$. This difference in magnitude is necessary to assure that rounding errors, associated with the function g's computation, are insignificant. Because we know that

$$\tan(B+y) = [\tan(B) + \tan(y)] / [1 - \tan(B)\tan(y)],$$

isolating the tan(B) term yields

$$\begin{aligned} \tan(B + y) = {} & \tan(B) + \sec(B)\csc(B) \\ & \tan(y)/(\cot(B) - \tan(y)). \end{aligned}$$

Moreover,

$$\begin{aligned} &\sec^2(B + y)c \approx \sec^2(B)c \\ &= \tan(B)\sec(B)\csc(B)c. \end{aligned}$$

We pre-compute values for tan(B), cot(B), and a product, sec(B)csc(B), and store them in tables. We use these constants to compute cot(r + c). Both tan(B) and cot(B) need to be very precise, so are stored as the pairs (T_{hi}, T_{lo}) and (C_{hi}, C_{lo}). The sec(B)csc(B) values are stored as working-precision numbers.

Because $|y| \leq 2^{-7}$, we approximate the tan(y) by a short polynomial of the form

$$y + p_1y^3 + p_2y^5 + p_3y^7.$$

Finally, we employ an iterative procedure to compute $1/(\cot(B) - \tan(y))$. Since

$$\begin{aligned} &1/(\cot(B) - \tan(y)) \approx 1/(\cot(B) - y) \\ &\approx T_{hi}(1 + T_{hi}\, y + [T_{hi}y]^2), \end{aligned}$$

the latter provides a good initial approximation that requires only two subsequent iterations to converge on an accurate inverse.

3.5 Computation of Cot(r+c)

Recall that the tangent is equivalent to the negative cotangent if the argument falls in certain quadrants. For reduced arguments limited to $|r + c| \leq 2^{-14}$, the approximation $1/(r + c) - r/3 - r^3/45$ provides sufficient

accuracy. The 3rd degree term is not needed if $|r + c| < 2^{-30}$. However, unlike all the other trigonometric functions discussed hitherto, the dominant term poses a significant complication. The term $1/(r + c)$ cannot in general be expressed exactly. We can however express it accurately as two working-precision numbers. We do this by first computing, z_{hi}, a working-precision approximation to $1/(r + c)$. We then calculate z_{lo}, which serves as a correction with care;

$$z_{lo} = [(1 - z_{hi}r) - z_{hi}c]z_{hi}.$$

This gives $z_{lo} \approx 1/(r+c) - z_{hi}$ to within an acceptable number of rounding errors. The short approximation to $\cot(r + c)$ is in this case

$$z_{hi} + ((-r - c)/3 - r^3/45 + z_{lo}).$$

For $2^{-14} < |r + c| < 2^{-2}$, we utilize an expression of the form

$$\cot(r + c) \approx 1/(r + c) + q_1(r + c) + q_2r^3 + ... + q_7r^{13}.$$

Again, $1/(r + c)$ is computed to extra precision as a pair z_{hi} and z_{lo}, and the final approximation is given by these values combined with other terms of a high-degree polynomial.

For larger r's, $|r| \geq 2^{-2}$,

$$\begin{aligned} \cot(r+c) &\approx \cot(r) + \cot'(r)c \\ &\approx \cot(r) - \csc^2(r)c \end{aligned}$$

provides a basic approximation. We use a table-driven approach similar to the one used for the tangent. We decompose $|r|$ as B + y because we know

$$\begin{aligned} \cot(B+y) = {} & \cot(B) - \sec(B)\csc(B) \\ & \tan(y)/(\tan(B) + \tan(y)) \end{aligned}$$

and

$$\begin{aligned} &\csc^2(B + y)c \approx \csc^2(B)c \\ &= \cot(B)\sec(B)\csc(B)c. \end{aligned}$$

We can approximate $\tan(y)$ by the same polynomial used for computing $\tan(r + c)$, and the value of $1/(\tan(B) - \tan(y))$ iteratively with an initial guess given by

$$\begin{aligned} &1/(\tan(B) + \tan(y)) \approx 1/(\tan(B) + y) \\ &\approx C_{hi}(1 - C_{hi}y + [C_{hi}y]^2) \end{aligned}$$

4. The Function Atan2(y,x)

The two argument instruction FPATAN(y,x) approximates the phase angle of the complex number, $x + iy$, in the range $[-\pi, \pi]$. This function is closely related to the inverse tangent function of one variable, namely $\arctan(\alpha)$. Using simple trigonometry, one can show that FPATAN(y,x) is equivalent to $\pm\arctan(v/u)$, $\pm(\pi/2 \pm \arctan(v/u))$, or $\pm(\pi - \arctan(v/u))$ where $v = \min(|x|,|y|)$ and $u = \max(|x|,|y|)$ as long as v is non-zero. We are therefore able to focus on the computation of $\arctan(v/u)$ for $0 < v \leq u$. We denote v/u by α, for $0 < \alpha \leq 1$, and let z be its working-precision approximation. We also compute a correction term to z, denoted z_c, using a technique similar to one described in Section 3.5. When $z < 2^{-3}$, we approximate $\arctan(\alpha)$ by a simple polynomial of the form

$$z + z_c + p_1z^3 + p_2z^5 + ... + p_8z^{17}.$$

When $z \geq 2^{-3}$, we use a table-driven algorithm. Given that $z = 2^k(1.\ b_1\ b_2\ b_3\ ...\ b_{63})$, we define the leading portion of z as $B = 2^k(1.b_1b_2b_3...b_51)$. Our goal is to express $\arctan(\alpha)$ as the sum of $\arctan(B)$ and another term that is small in magnitude and efficient to compute. By using a little trigonometry, we have

$$\begin{aligned} \arctan(\alpha) = {} & \arctan(B) + \\ & \arctan([v - Bu]/[u + Bv]), \end{aligned}$$

where $\arctan(B)$ is computed beforehand and stored in a table as a pair, (T_{hi}, T_{lo}). Next, we let $\xi = [v - Bu]/[u + Bv]$ and approximate $\arctan(\xi)$ by a polynomial of the form

$$\xi + q_1\xi^3 + q_2\xi^5 + q_3\xi^7 + q_4\xi^9.$$

As with α, ξ is computed as a working-precision number w with a correction w_c. Hence, we compute the polynomial above as

$$w + w_c + q_1w^3 + q_2w^5 + q_3w^7 + q_4w^9.$$

If we denote the polynomial approximation as w + Q, then $\arctan(v/u)$ is given accurately by

$$T_{hi} + T_{lo} + w + Q.$$

Now, we may further combine this last expression with $\pi/2$ or π as required to approximate FPATAN(y,x) precisely.

5 Accuracy

These new functions are designed to be more accurate than their predecessors on the Pentium™ as the error bound analyses done by both theoretical estimation and actual testing will attest.

The current theoretical error bounds are obtained using a traditional approach (see [13]). We are also developing a new automatic method that is somewhat different from that of [6]. The error bound provided for F2XM1 is based on this new method; the other bounds are found with the traditional method. We expect to have more precise error bounds for all of the IA-64 implementations by conference time based on the new techniques. This new method will be the subject of a future paper.

Intel is also working on proving these algorithms using a mechanical theorem prover. Currently, only the algorithm for F2XM1 was proved correct by mechanical means (see [4]).

Figure 1 summarizes the error bound analyses for the new and old algorithms (see [7] and [9]). Note that the estimated maximum errors for the sine, cosine, and tangent on the Pentium™ processor are given for the modified trigonometric function with a 66-bit period.

Instruction	Pentium™ (1992) Estimated	IA-64 (1998) Estimated	IA-64 (1998) Observed
F2XM1	.95	.53	.51
FYL2X	.85 (y = 1)	.60	.55
FYL2XP1	.85 (y = 1)	.60	.56
FSIN	.75	.65	.56
FCOS	.75	.65	.56
FPTAN	.98	.68	.57
FPATAN	.85	.65	.52

Figure 1: Error Bounds Given in Maximum Ulps

Monotonicity was proved for all of the functions – in the two argument functions, we held one argument constant and varied the other.

6 Conclusion

In order to take full advantage of the IA-64 architecture, a set of new algorithms were developed for each of the traditional IA-32 transcendental instructions.

In writing these functions, we wanted both high accuracy and excellent performance. We believe that both objectives were achieved.

We hope to see some of these new algorithmic techniques play a role in future transcendental function implementations in both hardware and software.

References

[1] E. W. Cheney, *Introduction to Approximation Theory*, Chelsea, New York, Second Edition, 1986.

[2] Carole Dulong, The IA-64 Architecture at Work, *IEEE Computer*, pp. 25-32, July, 1998.

[3] Gal, et al., An Accurate Elementary Mathematical Library for the IEEE Floating Point Standard, *ACM Transactions on Mathematical Software*, pp. 26-45, Vol 17, No 1, March, 1991.

[4] John Harrison, Floating point verification in {HOL}{L}ight: the exponential function, *Technical Report 428*, University of Cambridge Computer Laboratory, New Museums Site, Cambridge, UK. 1997.

[5] *Intel Architecture Software Developer's Manual*, Volumes 1, 2, and 3, Order Numbers 243190-001, 243191-001, 243192-001, 1997.

[6] Walter Krämer, A priori Worst-Case Error Bounds for Floating-Point Computations, *Proceedings of the 13th Symposium on Computer Arithmetic*, pp. 64-71, 1997.

[7] Ted Kubaska, Accuracy Plots for the IVE Transcendental Functions, *Intel Internal Document*, July, 1998.

[8] Lynch, et al., The K5 Transcendental Functions. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 163-170, 1995.

[9] *Pentium Family User's Manual*, Volume 3, Order Number 241430-003, Appendix G, pp. 1-17, 1994.

[10] Roger Alan Smith, A Continued-Fraction Analysis of Trigonometric Argument Reduction, *IEEE Transactions on Computers*, pp. 1348-1351, Vol 44, No 11, November, 1995.

[11] Ping Tak Peter Tang, Table-Driven Implementation of the Exponential Function in IEEE Floating-point Arithmetic, *ACM Transactions on Mathematical Software*, pp. 144-157, Vol 15, No 2, June, 1989.

[12] Ping Tak Peter Tang, Table-Driven Implementation of the Logarithm Function in IEEE Floating-point Arithmetic, *ACM Transactions on Mathematical Software*, pp. 378-400, Vol 16, No 2, December, 1990.

[13] Ping Tak Peter Tang, Table-Lookup Algorithms for Elementary Functions and Their Error Analysis. *Proceedings of the 10th Symposium on Computer Arithmetic*, pp. 232-236, 1991.