The GNU libc atanh is correctly rounded

Paul Zimmermann
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

***Abstract*—We prove the binary64 hyperbolic arc-tangent function from GNU libc 2.43, released end of January 2026, is correctly rounded.**

***Index Terms*—IEEE 754, binary64 format, correct rounding.**

I. Introduction

The GNU libc 2.43 provides a new implementation of the `atanh` function, integrated from the CORE-MATH project [15], together with some other binary64 functions: `asinh`, `acosh`, `erf`, `erfc`, `tgamma` and `lgamma`. We prove this new implementation is correctly rounded, for any IEEE 754 rounding mode. As a consequence, users of the GNU libc will get the best possible result for this function, and they will also get reproducibility of their results.

Several implementations from the literature claim correct rounding for the IEEE 754 binary formats: MathLib [17], Sun's LIBMCR around 2004, CR-LIBM [3], RLIBM and RLIBM-ALL [11], LLVM libc [12], CORE-MATH [15] (see more details in [1]). However, very few implementations provide a proof that they are correctly rounded. In [7], the authors provide a full pen-and-paper proof of the CORE-MATH binary64 power function, the fast path being proven using a proof assistant in [8], but no details are given for the accurate path. To the best of our knowledge, there was so far no such proof of correctness for a function from a well-established library like the GNU libc or the Intel Math Library.

The contributions of this article are the following. We show that using a clever exhaustive search, it is possible to check a binary64 function is correctly rounded using academic resources, at least for a few binades. We also demonstrate how to prove error bounds for the fast path and for the accurate path using the Sollya and Gappa tools [2], [4], and we make our work reproducible by exhibiting the scripts we used. Finally, we provide a full proof of a concrete implementation from a well-established library.

The organization of this article is the following. In Section II we analyze the `atanh` function from the GNU libc, and prove it is correctly rounded. Then in Section III we conclude and discuss some possible future work. For reproducibility, some auxiliary scripts are given in appendix, together with some technical proofs.

II. Analysis of the GNU libc hyperbolic arc-tangent function

The GNU libc `atanh` function originates from the CORE-MATH implementation, which was integrated by Adhemerval Zanella into GNU libc 2.43. The algorithms are the same as in CORE-MATH, while the code is adapted to the GNU libc coding style. The CORE-MATH implementation was designed by Alexei Sibidanov. In GNU libc, the `atanh` function (Fig. 1), which is defined as

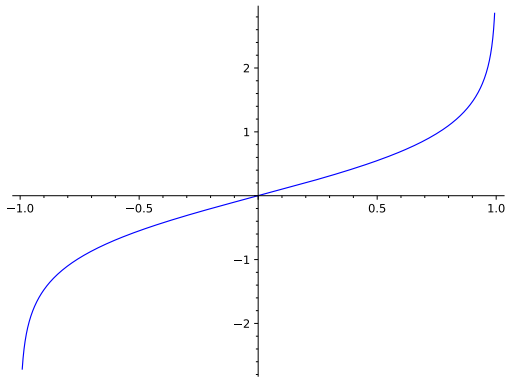

Fig. 1. The hyperbolic arc-tangent function.

$$\operatorname{atanh}(x) = \frac{1}{2}\log\frac{1+x}{1-x}, \tag{1}$$

is implemented in file `dbl-64/e_atanh.c`, which is 250-line long. This routine has three branches: one for $|x| < x_0$ where $x_0 \approx 1.35 \cdot 10^{-8}$, one for $x_0 \le |x| < 1/4$, and one for $1/4 \le |x| < 1$ (the hyperbolic arc-tangent function is only defined for $|x| \le 1$, and for $x = \pm 1$ it yields $\pm\infty$). We study the first branch in §II-A, the fast path of the second branch in §II-B, the third branch in §II-C, and the accurate path of the second branch in §II-D.

Since `atanh` is an odd function, and the mathematical approximation used in each branch, say $p(x)$, satisfies $p(-x) = -p(x)$, the following holds: for rounding to nearest (RN) or toward zero (RZ), the rounded approximation $\widehat{p}$ satisfies $\widehat{p}(-x) = -\widehat{p}(x)$. For rounding towards $+\infty$ (RU) and towards $-\infty$ (RD), $\widehat{p}_{\mathrm{RU}}(-x) = -\widehat{p}_{\mathrm{RD}}(x)$ and $\widehat{p}_{\mathrm{RD}}(-x) = -\widehat{p}_{\mathrm{RU}}(x)$. It thus suffices to analyze the case $x \ge 0$.

A. Branch $0 \le x < x_0$

This branch deals with $0 \le x < x_0$, where $x_0 =$ `0x1.d12ed0af1a27fp-27`. The GNU libc routine simply returns $\mathrm{fma}(x, 2^{-55}, x)$ in that case. For $0 < x \le 0.1$, we have:

$$x < \operatorname{atanh}(x) < x + \frac{x^3}{3} + 0.21x^5.$$

Let $x_1 =$ `0x1.bb67ae8584caap-27` $\approx 1.29 \cdot 10^{-8}$. If $0 < x \le x_1$, then $x^2/3 + 0.21x^4 < 2^{-54}$, thus $x < \operatorname{atanh}(x) < x + 2^{-54}x$, and since there is no rounding

boundary in $(x, x + 2^{-54}x)$, $\mathrm{atanh}(x)$ rounds to the same value as $x+2^{-55}x$, i.e., x except for rounding upwards where it rounds to nextabove(x). For x a binary64 number satisfying $x_1 < x < x_0$, the term $\frac{x^3}{3} + 0.21x^5$ lies in $[2^{-81}, 2^{-80})$, and $\mathrm{ulp}(x) = 2^{-79}$, thus $\frac{x^3}{3} + 0.21x^5 < \frac{1}{2}\mathrm{ulp}(x)$, and the same rounding rule as above applies. In conclusion, for $0 < x < x_0$, the formula $\mathrm{fma}(x, 2^{-55}, x)$ returns the correct rounding of $\mathrm{atanh}(x)$, whatever the rounding mode. Note that this formula also works for $x = \pm 0$, thus there is no need to check whether x is zero. Note also that x_0 is optimal, since $\mathrm{fma}(x, 2^{-55}, x)$ does not yield the correct rounding for $x = x_0$ and rounding to nearest.

This result was formally proven by Hyunsoo Lee, Hyunwoo Lee, Wonyeol Lee, and Guillaume Melquiond using the Rocq proof assistant [5].

B. Branch $x_0 \le x < 1/4$

For $x_0 \le x < 1/4$, in the fast path, Algorithm AtanhSmall (detailed below) is used, with some variables renamed for clarity. It uses a degree-21 polynomial approximation of atanh:

$$p(x) = x + (h + \ell)x^3 + c_0x^5 + c_1x^7 + \cdots + c_8x^{21}. \quad (2)$$

To save space, we omit the roundings in all algorithms in this article: when we write $a \leftarrow b + c$ it means $a \leftarrow \circ(b + c)$, and when we write $a \leftarrow bc+d$, it means either $a \leftarrow \mathrm{fma}(b, c, d)$ or $a \leftarrow \circ(\circ(bc)+d)$ where $\circ$ denotes the current rounding mode. When we write $a \leftarrow \mathrm{fma}(b, c, d)$, it means a fused multiply-add (FMA) is required.[1] Since the form $\mathrm{fma}(b, c, d)$ yields a smaller error bound than $\circ(\circ(bc) + d)$, we assume the latter form in the error analysis below. Algorithm AtanhSmall uses two auxiliary routines, fasttwosum and muldd, which we detail in Algorithms 2 and 3. Algorithm fasttwosum is classical (see Algorithm 4.3 from [14] for rounding to nearest), and Algorithm muldd is exactly Algorithm 10 from [10]. In [10], when the inputs are normalized, i.e., $x_h = \mathrm{RN}(x_h + x_\ell)$, similarly for c_h+c_ℓ, and the precision is at least 4, the authors give a relative error bound of $7u^2$ for rounding to nearest-even, where $u = 2^{-53}$ is the unit roundoff. (A tight relative error bound of $5u^2$ is given for rounding to nearest-even in [13, Theorem 2.7].)

Theorem 1: Let x a binary64 number, $x_0 \le x < 1/4$. Let p_h, p_ℓ be the output of Algorithm AtanhSmall. Let $\epsilon = \circ(x \cdot \circ(\circ(x_4 \cdot 29/2^{57}) + 2^{-103}))$, $\ell_b = p_h + \circ(p_\ell - \epsilon)$, $u_b = p_h + \circ(p_\ell + \epsilon)$. Then if $\circ(\ell_b) = \circ(u_b)$, $\circ(\ell_b)$ is the correct rounding of $\mathrm{atanh}(x)$.

Note: here, ℓ_b and u_b denote different values than in the GNU libc code (where they denote the corresponding rounded values). In the rest of this subsection, we prove this theorem. In this proof we use at several places the fact that if t is a real not in the subnormal range, then $\mathrm{ulp}(t) \le 2u \cdot |t|$.

Proof: The overall structure of the proof is as follows. In a first part, we deal with the case $2^{-12} \le x < 1/4$. In a second part, we deal with the case $x_0 \le x < 2^{-12}$. In each part, we analyze the rounding error due to the different steps of Algorithm AtanhSmall.

First assume $2^{-12} \le x < 1/4$.

Analysis of the computation of t at line 19. The variable t approximates the right part of the polynomial $p(x)$ from Eq. (2), not taking into account the x and hx^3 terms, and before multiplication by x^3. Gappa proves (see details in Appendix C):

$$|t - (\ell + c_0x^2 + c_1x^4 + \cdots + c_8x^{18})| < 2^{-52.3079} \cdot x_2,$$

where x_2 is the variable at line 12. By modifying the Gappa script from Appendix C, we also prove:

$$2^{-54.9069} < |t| < 2^{-6.25576}.$$

Analysis of $a_h, a_\ell \leftarrow$ fasttwosum(h, t). Here $a_h + a_\ell$ is a double-double approximation of $(p(x) - x)/x^3$. Since $0.3333 < h < 0.33333334$ and $2^{-54.9069} < |t| < 2^{-6.25576}$, the conditions of Theorem 2 from [9] are satisfied, thus FastTwoSum is exact whatever the rounding mode:

$$\left| a_h + a_\ell - \frac{p(x) - x}{x^3} \right| < 2^{-52.3079} \cdot x_2,$$

[1] On processors with no hardware FMA, fused multiply-adds are emulated in software, thus one should use them only when required.

Algorithm 1 (AtanhSmall)

Input: x a binary64 number, $x_0 \le x < 1/4$
Output: p_h, p_ℓ approximating $\mathrm{atanh}(x)$

1: $h \leftarrow$ `0x1.5555555555555p-2`
2: $\ell \leftarrow$ `0x1.5555555555555p-56`
3: $c_0 \leftarrow$ `0x1.999999999999ap-3`
4: $c_1 \leftarrow$ `0x1.2492492492244p-3`
5: $c_2 \leftarrow$ `0x1.c71c71c79715fp-4`
6: $c_3 \leftarrow$ `0x1.745d16f777723p-4`
7: $c_4 \leftarrow$ `0x1.3b13ca4174634p-4`
8: $c_5 \leftarrow$ `0x1.110c9724989bdp-4`
9: $c_6 \leftarrow$ `0x1.e2d17608a5b2ep-5`
10: $c_7 \leftarrow$ `0x1.a0b56308cba0bp-5`
11: $c_8 \leftarrow$ `0x1.fb6341208ad2ep-5`
12: $x_2 \leftarrow x \cdot x, \quad d_2 \leftarrow \mathrm{fma}(x, x, -x_2)$
13: $x_4 \leftarrow x_2 \cdot x_2, \quad x_3 \leftarrow x_2 \cdot x, \quad x_8 \leftarrow x_4 \cdot x_4$
14: $d_3 \leftarrow \mathrm{fma}(x_2, x, -x_3) + d_2 \cdot x$
15: $p_0 \leftarrow c_0 + x_2c_1, \quad p_2 \leftarrow c_2 + x_2c_3$
16: $p_4 \leftarrow c_4 + x_2c_5, \quad p_6 \leftarrow c_6 + x_2c_7$
17: $p'_0 \leftarrow p_0 + x_4p_2, \quad p'_4 \leftarrow p_4 + x_4p_6$
18: $p''_4 \leftarrow p'_4 + x_8c_8, \quad p \leftarrow p'_0 + x_8p''_4$
19: $t \leftarrow \mathrm{fma}(x_2, p, \ell), \quad a_h, a_\ell \leftarrow \mathrm{fasttwosum}(h, t)$
20: $b_h, b_\ell \leftarrow \mathrm{muldd}(a_h, a_\ell, x_3, d_3)$
21: $p_h, t_\ell \leftarrow \mathrm{fasttwosum}(x, b_h)$
22: $p_\ell \leftarrow b_\ell + t_\ell$

Algorithm 2 (fasttwosum)

Input: a, b binary64 numbers
Output: h, ℓ such that $h + \ell$ approximates $a + b$

1: $h \leftarrow a + b$
2: $t \leftarrow h - a$
3: $\ell \leftarrow b - t$

Algorithm 3 (muldd)

Input: x_h, x_ℓ, c_h, c_ℓ binary64 numbers
Output: h, ℓ approximating $(x_h + x_\ell)(c_h + c_\ell)$
1: $\ell_1 \leftarrow c_h x_\ell$, $\ell_2 \leftarrow c_\ell x_h$
2: $h_1 \leftarrow x_h c_h$, $\ell_3 \leftarrow \mathrm{fma}(c_h, x_h, -h_1)$
3: $\ell_4 \leftarrow \ell_3 + (\ell_1 + \ell_2)$
4: $h, \ell \leftarrow \mathrm{fasttwosum}(h_1, \ell_4)$

and as a consequence $|a_\ell| < \mathrm{ulp}(a_h) = 2^{-54}$. We also deduce that $|a_h| < (0.33333334 + 2^{-6.25576})(1+2u)$, from which we deduce $|a_h + a_\ell| < 0.34642$.

Analysis of $b_h, b_\ell \leftarrow \mathrm{muldd}(a_h, a_\ell, x_3, d_3)$. Here $x_3 + d_3$ is a double-double approximation of x^3 computed at lines 13-14, and $b_h + b_\ell$ is a double-double approximation of $p(x) - x$, thus of $\mathrm{atanh}(x) - x$. At input with the names of the muldd variables, we have $|x_h| < 0.34642$, $|x_\ell| < 2^{-54}$, $|c_h| < 1.0001x^3$, and with Gappa we prove $|c_\ell| < 2^{-50.9999}x^3$ (see Appendix D). Then Gappa proves (see Appendix E):

$$|b_h + b_\ell - (a_h + a_\ell)(x_3 + d_3)| < 2^{-102.264}x^3.$$

Note: assuming rounding to nearest-even, and if the inputs were normalized (which is not necessarily the case here), the bound $5u^2$ from [13] would give a bound of $2^{-105.263}x^3$ (using $b_h + b_\ell \approx x^3/3$). As a comparison, for rounding to nearest, Gappa obtains a bound of $2^{-103.13}x^3$ for our non-normalized inputs.

Summarizing up to here, we have:

$$\begin{aligned} |b_h + b_\ell &- (p(x) - x)| \\ &\leq |b_h + b_\ell - (a_h + a_\ell)(x_3 + d_3)| \\ &+ |a_h + a_\ell||x_3 + d_3 - x^3| \\ &+ |(a_h + a_\ell)x^3 - (p(x) - x)|. \end{aligned}$$

The first term on the right-hand side is bounded by $2^{-102.264}x^3$, the second by $0.34642 \cdot 2^{-103}x^3$ (see Appendix D), and the third by $x^3 \cdot (2^{-52.3079}x_2)$. Using $x_2 \leq 1.00001x^2$, we obtain:

$$|b_h + b_\ell - (p(x) - x)| < 2^{-101.991}x^3 + 2^{-52.3078}x^5.$$

Analysis of $p_h, t_\ell \leftarrow \mathrm{fasttwosum}(x, b_h)$. After this instruction, $p_h + t_\ell$ is a double-double approximation of $p(x) - b_\ell$, and it will remain to add b_ℓ. Since $b_h < 0.347x^3$ (see Appendix E), the FastTwoSum precondition is satisfied. Moreover, since $b_h \approx x^3/3$, for $|x| \geq 2^{-25}$, the exponents of x and b_h differ by at most 53, thus by Theorem 2 from [9], FastTwoSum is exact whatever the rounding mode:

$$p_h + t_\ell = x + b_h.$$

Moreover since $p_h < (1 + 0.347 \cdot (1/4)^2)(1+2u)x < 1.022x$, we have $|t_\ell| < \mathrm{ulp}(p_h) < 2^{-51.968}x$.

Analysis of $p_\ell \leftarrow b_\ell + t_\ell$. Now $p_h + p_\ell$ is the final double-double approximation of $p(x)$. Since $|b_\ell| < 2^{-53.529}x^3$ (see Appendix E) and $|t_\ell| < 2^{-51.968}x$ from just above, we deduce $|p_\ell| < (2^{-53.529}x^3 + 2^{-51.968}x)(1 + 2u) < 2^{-51.967}x + 2^{-53.528}x^3$. The rounding error on p_ℓ is thus bounded by $\mathrm{ulp}(p_\ell) \leq 2u \cdot |p_\ell| < 2^{-103.967}x + 2^{-105.528}x^3$.

In summary:

$$|p_h + p_\ell - p(x)| < 2^{-103.967}x + 2^{-101.871}x^3 + 2^{-52.3078}x^5.$$

It remains to take into account the mathematical error, i.e., the difference between $p(x)$ and $\mathrm{atanh}(x)$. Ideally, we would like to incorporate the mathematical error into the term $2^{-52.3078}x^5$. Appendix A shows $|p(x) - \mathrm{atanh}(x)| < 2^{-56.261}x^5$ for $x_0 \leq x < 1/4$, thus:

$$\begin{aligned} |p_h + p_\ell &- \mathrm{atanh}(x)| < 2^{-103.967}x + 2^{-101.871}x^3 \\ &+ 2^{-52.2175}x^5. \end{aligned} \tag{3}$$

Since $\ell_b = p_h + \circ(p_\ell - \epsilon)$ and $u_b = p_h + \circ(p_\ell + \epsilon)$, ℓ_b is a lower approximation of $p_h + p_\ell$, and u_b an upper approximation. If we prove that $|p_h + p_\ell - \ell_b|$ is larger than the right-hand side of Eq. (3), and similarly for u_b, it will follow $\ell_b < \mathrm{atanh}(x) < u_b$, thus if both ℓ_b and u_b round to the same value, so will $\mathrm{atanh}(x)$.

Analysis of $\epsilon = \circ(x \cdot \circ(\circ(x_4 \cdot 29/2^{57}) + 2^{-103}))$. Let $y = x_4 \cdot 29/2^{57}$. Since $x_2 = x^2(1+\theta_1)$ and $x_4 = x_2^2(1+\theta_2)$ with $|\theta_1|, |\theta_2| < 2u$, we can write $x_4 = x^4(1+\theta_3)^3$ with $|\theta_3| < 2u$. It follows $x_4 > 0.9999x^4$, and thus $\circ(y) > 2^{-52.1422}x^4$. Now let $z = \circ(y) + 2^{-103}$. Since $z > 2^{-52.1422}x^4 + 2^{-103}$, it follows $\circ(z) > 2^{-52.1423}x^4 + 2^{-103.0001}$. Then:

$$\epsilon > 2^{-52.1424}x^5 + 2^{-103.0002}x.$$

A similar reasoning yields an upper bound for ϵ:

$$\epsilon < 2^{-52.1416}x^5 + 2^{-102.9998}x.$$

Bounding x^4 by 2^{-8}, we deduce $\epsilon < 2^{-60.1415}x$. Since $|p_\ell| < 2^{-51.967}x + 2^{-53.528}x^3 < 2^{-51.936}x$ for $x \leq 1/4$, if follows $|p_\ell \pm \epsilon| < 2^{-51.931}x$, thus the rounding error on $\circ(p_\ell \pm \epsilon)$ is bounded by $\mathrm{ulp}(2^{-51.931}x) \leq 2^{-103.931}x$. We deduce:

$$\begin{aligned} |p_h &+ p_\ell - \ell_b| > \epsilon - 2^{-103.931}x \\ &> 2^{-52.1424}x^5 + 2^{-103.0002}x - 2^{-103.931}x \\ &> 2^{-104.073}x + 2^{-52.1424}x^5, \end{aligned} \tag{4}$$

which holds for any $x < 1/4$, and the same bound holds for $|p_h + p_\ell - u_b|$. In summary, we know that $|p_h + p_\ell - \mathrm{atanh}(x)| < f(x)$ with $f(x)$ the right-hand side of Eq. (3), and $|p_h + p_\ell - \ell_b| > g(x)$ with $g(x)$ the right-hand side of Eq. (4). Appendix B shows that $f(x) < g(x)$ for $x \geq 2^{-12}$. Thus for $x \geq 2^{-12}$, the distance between ℓ_b and $p_h + p_\ell$ (resp. u_b and $p_h + p_\ell$) is larger than the distance between $\mathrm{atanh}(x)$ and $p_h + p_\ell$. This proves that ℓ_b and u_b enclose $\mathrm{atanh}(x)$.

We now prove the theorem for $x_0 \leq x < 2^{-12}$.

Write $x = \alpha \cdot \mathrm{ufp}(x)$ with $1 \leq \alpha < 2$, where $\mathrm{ufp}(x)$ denotes the unit-in-first-place of x. First assume $\alpha \in [1.052, 2 - 2^{-24}]$. Then since $b_h < 0.347x^3 < 2^{-25.5}x$ for $x < 2^{-12}$, we have $b_h < 2^{-24.5}\mathrm{ufp}(x)$, thus $x + b_h < (\alpha + 2^{-24.5})\mathrm{ufp}(x) < (2 - 2^{-26})\mathrm{ufp}(x)$, which proves that p_h lies in the same binade as x after $p_h, t_\ell \leftarrow \mathrm{fasttwosum}(x, b_h)$.

Then since $|t_\ell| < \mathrm{ulp}(p_h)$, we deduce $|t_\ell| < \mathrm{ulp}(x)$. After $p_\ell \leftarrow b_\ell + t_\ell$, we have $|p_\ell| \leq 1.5\mathrm{ulp}(x)$, since $|b_\ell| < 2^{-53.529}x^3 < 0.5\mathrm{ulp}(x)$ (see Appendix E). Thus the rounding error on p_ℓ is less than $2^{-104}\mathrm{ufp}(x)$. Indeed, if we had $|p_\ell| < \mathrm{ulp}(x)$, then $p_h + p_\ell$ would be a 106-bit approximation, thus the rounding error would be bounded by $2^{-105}\mathrm{ufp}(x)$; with $|p_\ell| < 1.5\mathrm{ulp}(x)$ we lose a factor two. Now $2^{-104}\mathrm{ufp}(x) = 2^{-104}/\alpha \cdot x \leq 2^{-104.0731}x$, and $2^{-104.0731}x + 2^{-101.991}x^3 < 2^{-104.073}x$, where the term $2^{-101.991}x^3$ comes from the error on $b_h + b_\ell$. Hence Eq. (3) becomes:

$$|p_h + p_\ell - \mathrm{atanh}(x)| < 2^{-104.073}x + 2^{-52.2175}x^5.$$

The right-hand side is clearly less than the lower bound from Eq. (4), which as above proves the correctness of the fast path rounding test.

It thus only remains to check the ranges $\alpha \in [1, 1.052]$ and $\alpha \in [2-2^{-24}, 2)$ for each binade in $[x_0, 2^{-12}]$, which amounts to about $2^{51.5}$ values to check. This was performed with the exhaustive search algorithm from §II-C.

■

Remark: the version of muldd which is used in Algorithm AtanhSmall delivers a double-double approximation with about 102 correct bits. This is too large for a fast path aiming at about 65 correct bits. One can use instead Algorithm 4 which avoids the final FastTwoSum call in muldd. Appendix F shows that our error analysis remains valid in that case, and we see in Table I that it yields a small but visible speedup. Another variant suggested by Claude-Pierre Jeannerod (personal communication) first computes h and ℓ_3 as in Algorithm muldd_fast, then accumulates $c_h x_\ell$ with $\ell_1 \leftarrow \mathrm{fma}(c_h, x_\ell, \ell_3)$, then accumulates $c_\ell x_h$ with $\ell \leftarrow \mathrm{fma}(c_\ell, x_h, \ell_1)$. However, while muldd and muldd_fast are commutative (they give the same result for inputs x_h, x_ℓ, c_h, c_ℓ and c_h, c_ℓ, x_h, x_ℓ), this variant is not.

Algorithm 4 (muldd_fast)

Input: x_h, x_ℓ, c_h, c_ℓ binary64 numbers
Output: h, ℓ approximating $(x_h + x_\ell)(c_h + c_\ell)$
1: $\ell_1 \leftarrow c_h x_\ell$, $\ell_2 \leftarrow c_\ell x_h$
2: $h \leftarrow x_h c_h$, $\ell_3 \leftarrow \mathrm{fma}(c_h, x_h, -h)$
3: $\ell \leftarrow \ell_3 + (\ell_1 + \ell_2)$

	rec. throughput	latency
GNU libc 2.42	52.3	100.
GNU libc 2.43	36.4	72.9
GNU libc 2.43 patched	35.7	71.7
Intel Math Library 2025.2.1	24.4	58.0

TABLE I
RECIPROCAL THROUGHPUT AND LATENCY (IN CYCLES) OF THE ATANH FUNCTION FROM VARIOUS LIBRARIES, ON A INTEL XEON SILVER 4214 WITH GCC 15.2.0. GNU LIBC 2.43 PATCHED DENOTES GNU LIBC 2.43 WITH MULDD REPLACED BY MULDD_FAST IN ALGORITHM ATANHSMALL. GNU LIBC WAS CONFIGURED WITH -O3 -MARCH=NATIVE, AND THESE TIMINGS WERE OBTAINED WITH THE CORE-MATH BENCHMARK.

C. Branch $1/4 \leq x < 1$

In this branch, $\mathrm{atanh}(x)$ is computed directly using Eq. (1): one first gets a double-double approximation of $(1+x)/(1-x)$, then of its logarithm. One could probably analyze this branch as in §II-A or §II-B, but since there are only two binades to check, we prefer to use a brute-force exhaustive search algorithm. We have to check correctness for each of the four rounding modes, and with or without the use of FMA. This makes a total of 2^{56} calls to the `atanh` function.

To compute the reference value, we use the following algorithm. We scan intervals $[x_0, x_0 + n\nu)$, where $\nu = \mathrm{ulp}(x_0)$, and n is a large integer. Using GNU MPFR [6], we compute a double-double approximation $h+\ell$ of $\mathrm{atanh}(x_0)$, and a double approximation k of $\mathrm{atanh}'(x_0)$. Then:

$$\mathrm{atanh}(x_0 + i\nu) \approx h + \ell + i\nu k.$$

After bounding the mathematical and rounding error by ϵ, we obtain lower/upper bounds $y_\ell = h + \ell + i\nu k - \epsilon$ and $y_h = h + \ell + i\nu k + \epsilon$. If both y_ℓ and y_h round to the same value (with the target rounding mode), and this matches what returned the `atanh` function, everything is fine. This case happens most of the time, assuming the `atanh` function is correctly rounded. If any of these three values differs from the other ones, we use GNU MPFR to compute the correct rounding of $\mathrm{atanh}(x_0 + i\nu)$, and we report a failure if this differs from the value returned by the `atanh` function. This algorithm is implemented in CORE-MATH: for example `./check.sh --special -a 0x1p-2 -C 1000000 atanh` scans the interval of $n = 10^6$ values starting at $x_0 = 1/4$.

On a 64-core computer (Intel Xeon Silver 4214), with gcc 15.2.0, this check takes 27 seconds of wall-clock time for $x_0 = 1/4$ and $n = 10^{10}$, for all four rounding modes $(\mathrm{RN}, \mathrm{RZ}, \mathrm{RU}, \mathrm{RD})$. Checking all 2^{56} calls to the `atanh` function takes only about 1.5 year on such a computer. Since we have access to several such computers, checking all binary64 values in $[1/4, 1)$ took only a few days.

D. Accurate path for $x_0 \leq x < 1/4$

We only have to check the accurate path for the branch $x_0 \leq x < 1/4$, since in §II-A we fully proved the branch $0 \leq x < x_0$, and in §II-C we fully proved the branch $1/4 \leq x < 1$. The accurate path for this branch corresponds to function `as_atanh_zero` in the GNU libc code. It uses a degree-37 odd polynomial $q(x)$, with double-double coefficients up to degree 27, and double coefficients from degree 29 to 37. The relative error is bounded according to Sollya `supnorm` for $x_0 \leq x < 1/4$ as follows:

$$\left| \frac{q(x)}{\mathrm{atanh}(x)} - 1 \right| < 2^{-111.784}.$$

The polynomial $q(x) = c_1 x + c_3 x^3 + \cdots + c_{37} x^{37}$ is evaluated using Horner's scheme, where the upper part (degree 29 and more) is evaluated using double precision, and the lower part with double-double arithmetic (using Algorithms fasttwosum and muldd).

At the end of the accurate path, we obtain a triple-double approximation $y_0 + y_1 + y_2$ of $q(x)$, where y_0 is the main term, y_1 is a small correction on y_0, and y_2 is a correction on y_1. Using Gappa, we prove the following inequality (for all rounding modes):

$$|y_0 + y_1 - q(x)| < 2^{-101.681}x.$$

The corresponding Gappa script is about 750 lines long, and takes about 3 minutes to complete for a single rounding mode on an Intel Core i5-6500. Because Gappa does not allow subroutines, designing this script was quite painful, and it is quite hard to read. Anyway, since $x \leq \mathrm{atanh}(x)$, this yields:

$$|y_0 + y_1 - q(x)| < 2^{-101.681}\mathrm{atanh}(x).$$

Combined with the mathematical error, we deduce:

$$|y_0 + y_1 - \mathrm{atanh}(x)| < 2^{-101.679}\mathrm{atanh}(x). \tag{5}$$

At the end of the accurate path, an extra rounding test checks whether $y_1 = 0$ and if so adjusts the result according to the sign of the extra term y_2.

Theorem 2: For any binary64 number x such that $\mathrm{atanh}(x)$ has less than 47 identical bits after the round bit, if $y_0 + y_1$ is the double-double approximation computed by the accurate path, then $\circ(y_0 + y_1)$ is the correct rounding of $\mathrm{atanh}(x)$.

Proof: Let $y = \mathrm{atanh}(x)$. If y has less than 47 identical bits after the round bit, then y is at distance at least $2^{-48}\mathrm{ulp}(y)$ from any rounding boundary z: $|y - z| \geq 2^{-48}\mathrm{ulp}(y)$. Since $\mathrm{ulp}(y) > 2^{-53}|y|$, this yields $|y-z| \geq 2^{-101}|y|$. Thus Eq. (5) shows that $|y_0+y_1-\mathrm{atanh}(x)| < |\mathrm{atanh}(x)-z|$, i.e., y_0+y_1 is closer from $\mathrm{atanh}(x)$ than any rounding boundary. Thus $\circ(y_0 + y_1) = \circ(\mathrm{atanh}(x))$. ∎

We now state the main result of this article.

Theorem 3: The GNU libc `atanh` function is correctly rounded for any binary64 number and any rounding mode.

Proof: Let x be a binary64 number. As seen in §II, it suffices to consider $0 \leq x < 1$.

Section II-A proves correct rounding for $0 \leq x < x_0$, where $x_0 =$ `0x1.d12ed0af1a27fp-27`, and Section II-C for $1/4 \leq x < 1$.

For the branch $x_0 \leq x < 1/4$, Theorem 1 shows that if $\circ(\ell_b) = \circ(u_b)$ in Algorithm AtanhSmall, $\circ(\ell_b)$ is the correct rounding of $\mathrm{atanh}(x)$. This is precisely what is returned by the fast path in this case.

Otherwise the accurate path is called, and Theorem 2 proves that it yields the correct rounding of $\mathrm{atanh}(x)$ when it has less than 47 identical bits after the round bit.

Assume the fast path fails, and $\mathrm{atanh}(x)$ has 47 or more identical bits after the round bit. This means x is a *hard-to-round input*. Hard-to-round inputs for atanh are provided by the CORE-MATH project [16], which claims to include all inputs with at least 43 identical bits after the round bit. (We refer the reader to [1] for the classical literature on worst cases and hard-to-round inputs.) Within the CORE-MATH `atanh.wc` file, we found 1502 inputs in $[x_0, 1/4)$ with at least 47 identical bits after the round bit, and all of them are correctly rounded, for all rounding modes (likewise for their opposite). ∎

Note: The worst case for atanh in $[x_0, 1/4)$ is $x =$ `0x1.dfffffffffabap-21`, with 72 identical bits after the round bit.

III. Conclusion and future work

We have proven in this article that a concrete implementation of a binary64 function (atanh) in a well-established mathematical library (GNU libc) is correctly rounded. We have also proven a possible improvement in the GNU libc implementation. We used several methods to achieve this goal: an efficient brute-force exhaustive search algorithm in some intervals, the use of Sollya and Gappa in some other intervals, and a pen-and-paper proof otherwise. We can draw a few conclusions from this work. Firstly, an exhaustive check is possible to ensure correct rounding in binary64, at least for a few binades. This exhaustive check has to be done only once, and it will guarantee that all future calls of the function in scientific applications will be correctly rounded. Secondly, Sollya and Gappa are very useful tools for this kind of proof, and they can even tackle the accurate path, although we then reach the limits of the current version of Gappa.

As future work, we plan to prove other (presumably correctly-rounded) binary64 functions from GNU libc and/or CORE-MATH using the same approach. It would also be interesting to try to convert this proof to a mechanical proof that could be checked with a proof assistant (in particular, Gappa is able to extract a formal proof).

Acknowledgments. This work would not have been possible without the implementation of the CORE-MATH `atanh` function designed by Alexei Sibidanov, and without its integration into GNU libc by Adhemerval Zanella. The author thanks Guillaume Melquiond and Sylvain Chevillard for their help with the use of Gappa and Sollya respectively. Many thanks also to Claude-Pierre Jeannerod, Wonyeol Lee and Hyunsoo Lee (who pointed out an error in the Gappa script of Appendix C) for comments on an earlier version of this article. The three anonymous reviewers provided very valuable comments that enabled the author to greatly improve this article. The exhaustive tests from Section II-C were performed thanks to the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations (see https://www.grid5000.fr/).

References

[1] Brisebarre, N., Hanrot, G., Muller, J.-M., and Zimmermann, P. Correctly rounded evaluation of a function: why, how, and at what cost? *ACM Computing Surveys 58*, 1 (2026). https://hal.science/hal-04474530.

[2] Chevillard, S., Joldeş, M., and Lauter, C. Sollya: An environment for the development of numerical codes. In *Mathematical Software - ICMS 2010* (Heidelberg, Germany, September 2010), K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, Eds., vol. 6327 of *Lecture Notes in Computer Science*, Springer, pp. 28–31.

[3] DARAMY-LOIRAT, C., DEFOUR, D., DE DINECHIN, F., GALLET, M., GAST, N., LAUTER, C., AND MULLER, J.-M. CR-LIBM: A library of correctly rounded elementary functions in double-precision. Research report, LIP, 2006. https://hal-ens-lyon.archives-ouvertes.fr/ensl-01529804.
[4] DE DINECHIN, F., LAUTER, C., AND MELQUIOND, G. Certifying the floating-point implementation of an elementary function using Gappa. *Transactions on Computers 60*, 2 (2011), 242–253.
[5] ET AL., H. H. The Rocq Prover. https://zenodo.org/records/19256047, 2026.
[6] FOUSSE, L., HANROT, G., LEFÈVRE, V., PÉLISSIER, P., AND ZIMMERMANN, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (2007), article 13.
[7] HUBRECHT, T., JEANNEROD, C.-P., AND ZIMMERMANN, P. Towards a correctly-rounded and fast power function in binary64 arithmetic. In *2023 IEEE 30th Symposium on Computer Arithmetic (ARITH 2023)* (Portland, Oregon (USA), United States, 2023).
[8] HUBRECHT, T., JEANNEROD, C.-P., ZIMMERMANN, P., RIDEAU, L., AND THÉRY, L. Towards a correctly-rounded and fast power function in binary64 arithmetic. Extended version of an article published in the proceedings of ARITH 2023. Available at https://inria.hal.science/hal-04159652, 2024.
[9] JEANNEROD, C.-P., AND ZIMMERMANN, P. FastTwoSum revisited. In *32nd IEEE Symposium on Computer Arithmetic, ARITH 2025, El Paso, TX, USA, May 4-7* (2025).
[10] JOLDES, M., MULLER, J., AND POPESCU, V. Tight and rigorous error bounds for basic building blocks of double-word arithmetic. *ACM Trans. Math. Softw. 44*, 2 (2017).
[11] LIM, J. P., AND NAGARAKATTE, S. High performance correctly rounded math libraries for 32-bit floating point representations. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021* (2021), S. N. Freund and E. Yahav, Eds., ACM, pp. 359–374.
[12] The LLVM C Library. https://libc.llvm.org/.
[13] MULLER, J., AND RIDEAU, L. Formalization of double-word arithmetic, and comments on "tight and rigorous error bounds for basic building blocks of double-word arithmetic". *ACM Trans. Math. Softw. 48*, 1 (2022), 9:1–9:24.
[14] MULLER, J.-M., BRUNIE, N., DE DINECHIN, F., JEANNEROD, C.-P., JOLDES, M., LEFÈVRE, V., MELQUIOND, G., REVOL, N., AND TORRES, S. *Handbook of Floating-point Arithmetic (2nd edition)*. Birkhäuser Basel, July 2018.
[15] SIBIDANOV, A., ZIMMERMANN, P., AND GLONDU, S. The CORE-MATH Project. In *ARITH 2022 - 29th IEEE Symposium on Computer Arithmetic* (virtual, France, 2022).
[16] THE CORE-MATH PROJECT. Hard-to-round inputs for `atanh`. https://gitlab.inria.fr/core-math/core-math/-/blob/master/src/binary64/atanh/atanh.wc.
[17] ZIV, A. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Trans. Math. Softw. 17*, 3 (1991), 410–423.

APPENDIX

A. Bounding the mathematical error

We use the following Sollya script to bound the relative error of the polynomial $p(x)$ with respect to $\operatorname{atanh}(x)$ on $[x_0, 1/4)$:

```
x0=0x1.d12ed0af1a27fp-27;
c0=0x1.999999999999ap-3;
c1=0x1.2492492492244p-3;
c2=0x1.c71c71c79715fp-4;
c3=0x1.745d16f777723p-4;
c4=0x1.3b13ca4174634p-4;
c5=0x1.110c9724989bdp-4;
c6=0x1.e2d17608a5b2ep-5;
c7=0x1.a0b56308cba0bp-5;
c8=0x1.fb6341208ad2ep-5;
h=0x1.5555555555555p-2;
l=0x1.5555555555555p-56;
p=x+(h+l)*x^3+c0*x^5+c1*x^7+c2*x^9
+c3*x^11+c4*x^13+c5*x^15+c6*x^17
+c7*x^19+c8*x^21;
prec = 80;
dirtyinfnorm((p-atanh(x))/x^5, [x0,1/4]);
```

whose output with Sollya 7.0 is:

```
1.15755820722770303479518e-17
```

This relative error bound is less than $2^{-56.261}$. The command `dirtyinfnorm` is not rigorous, however a Sollya script kindly provided by Sylvain Chevillard rigorously proves this bound.

B. Bounding $f(x)/g(x)$

We show here that $f(x) < g(x)$ for $2^{-12} \le x < 1/4$, with $f(x) = 2^{-103.967}x + 2^{-101.871}x^3 + 2^{-52.2175}x^5$, the right-hand side of Eq. (3), and $g(x) = 2^{-104.073}x + 2^{-52.1424}x^5$, the right-hand side of Eq. (4). First since $x < 1/4$, the middle term of f is bounded by $2^{-105.871}x$, thus

$$f < f' := 2^{-103.625}x + 2^{-52.2175}x^5,$$

where $2^{-103.967} + 2^{-105.871} < 2^{-103.625}$. It follows:

$$\begin{aligned} g - f' &\ge (2^{-52.1424} - 2^{-52.2175})x^5 \\ &- (2^{-103.625} - 2^{-104.073})x \\ &\ge 2^{-56.5}x^5 - 2^{-105.5}x, \end{aligned}$$

which is positive for $x \ge 2^{-12}$ since $2^{-56.5}x^5 \ge 2^{-104.5}x$.

C. Bounding the error on t

We use the following Gappa script to bound the error on t at line 19 of Algorithm AtanhSmall, where t approximates $\ell + c_0x^2 + \cdots + c_8x^{18}$. Variables with an `r` suffix are rounded variables, and their equivalent without the `r` suffix are the exact variables they correspond to. The Gappa syntax `p0r rnd= c0 + x2r*c1` means that a rounding is performed in each operation on the right-hand side, thus it is equivalent to `p0r = rnd(c0 + rnd(x2r*c1))`.

```
@rnd = float<ieee_64,RND>;
h = 0x1.5555555555555p-2;
l = 0x1.5555555555555p-56;
c0 = 0x1.999999999999ap-3;
c1 = 0x1.2492492492244p-3;
c2 = 0x1.c71c71c79715fp-4;
c3 = 0x1.745d16f777723p-4;
c4 = 0x1.3b13ca4174634p-4;
c5 = 0x1.110c9724989bdp-4;
c6 = 0x1.e2d17608a5b2ep-5;
c7 = 0x1.a0b56308cba0bp-5;
c8 = 0x1.fb6341208ad2ep-5;
x2 = x*x;
x2r rnd= x*x;
d2r = -(x2r - x*x);
x4 = x2*x2;
```

XMIN	XMAX	STEP	bound
0.125	0.25	26	$2^{-52.3267}$
0.0625	0.125	26	$2^{-52.3628}$
2^{-5}	0.0625	26	$2^{-52.3894}$
2^{-6}	2^{-5}	26	$2^{-52.3900}$
2^{-7}	2^{-6}	26	$2^{-52.3901}$
2^{-8}	2^{-7}	26	$2^{-52.3902}$
...	...	...	...
2^{-22}	2^{-21}	26	$2^{-52.3902}$
2^{-23}	2^{-22}	26	$2^{-52.3396}$
2^{-24}	2^{-23}	26	$2^{-52.3396}$
2^{-25}	2^{-24}	52	$2^{-52.3643}$
$1.125 \cdot 2^{-26}$	2^{-25}	26	$2^{-52.3376}$
2^{-26}	$1.125 \cdot 2^{-26}$	208	$\mathbf{2^{-52.3079}}$
x_0	2^{-26}	26	$2^{-52.3589}$

TABLE II
BOUNDS FOR THE ERROR ON t IN TERMS OF x_2.

```
x4r rnd= x2r*x2r;
x3 = x2*x;
x3r rnd= x2r*x;
x8 = x4*x4;
x8r rnd= x4r*x4r;
p0 = c0+x2*c1;
p0r rnd= c0 + x2r*c1;
p2 = c2+x2*c3;
p2r rnd= c2 + x2r*c3;
p4 = c4+x2*c5;
p4r rnd= c4 + x2r*c5;
p6 = c6+x2*c7;
p6r rnd= c6+x2r*c7;
pp0 = p0 + x4*p2;
pp0r rnd= p0r + x4r*p2r;
pp4 = p4+x4*p6;
pp4r rnd= p4r + x4r*p6r;
ppp4 = pp4+x8*c8;
ppp4r rnd= pp4r + x8r*c8;
p = pp0+x8*ppp4;
pr rnd= pp0r+x8r*ppp4r;
t = x2*p+l;
tr = rnd(x2r*pr+l);
{ x in [XMIN,XMAX] -> |tr-t|/x2r in ? }
$ x in STEP;
```

We then have a script that substitutes RND by all four rounding modes (denoted `ne`, `zr`, `up`, `dn` in Gappa), XMIN and XMAX by bounds of the interval to be checked, and STEP by the number of subintervals to be considered for x. The script also checks the interval $[-\text{XMAX}, -\text{XMIN}]$. We ran this script with Gappa 1.6.1 and option `-Echange-threshold=0`. The results are shown in Table II, where we omit entries for binades between 2^{-21} and 2^{-8}, for which with STEP=26 we get the same bound $2^{-52.3902}$. In summary, Gappa proves that the error on t is bounded by $2^{-52.3079}x_2$ for $x_0 \le |x| \le 1/4$, where x_2 is the approximation of x^2 at line 12 of Algorithm AtanhSmall. This bound is obtained in the interval $[2^{-26}, 1.125 \cdot 2^{-26}]$ for rounding upwards.

D. Bounding the error in $x_3 + d_3$

The sum $x_3 + d_3$ is a double-double approximation of x^3. With the following Gappa script:

```
@rnd = float<ieee_64,RND>;
x2 = x*x;
x2r rnd= x*x;
d2r = -(x2r - x*x);
x3 = x2*x;
x3r rnd= x2r*x;
t1 = x2r*x-x3r; # exact
ur rnd= d2r*x;
d3r rnd= t1 + ur;
{ x in [1,2] -> |d3r/x3| in ? }
$ x in 8192;
```

we get that d_3 is bounded by $2^{-50.9999}x^3$, whatever the rounding mode. This bound is very close to optimal, since for $x =$ `0x1.000000fffffefp-26` and rounding toward zero, we get $d_3/x^3 \approx 2^{-51.000004}$.

With the following script,

```
@rnd = float<ieee_64,RND>;
x2 = x*x;
x2r rnd= x*x;
d2r = -(x2r - x*x);
x3 = x2*x;
x3r rnd= x2r*x;
t1 = x2r*x-x3r; # exact
ur rnd= d2r*x;
d3r rnd= t1 + ur;
{ x in [1,2] ->
  |x3-(x3r+d3r)|/x3 in ? }
x3-(x3r+d3r) ->
  (d2r*x-ur) + ((t1+ur)-d3r);
$ x in 5;
```

Gappa proves that $|x^3-(x_3+d_3)|$ is bounded by $2^{-102.415}x^3$, whatever the rounding mode. Using an exhaustive search over the binade $[1, 2)$, we found that $|x^3 - (x_3 + d_3)| < 2^{-103}x^3$ for all rounding modes.

E. Bounding the error from the `muldd` call

At input of $b_h, b_\ell \leftarrow \text{muldd}(a_h, a_\ell, x_3, d_3)$, we have $|x_h| < 0.34642$, $|x_\ell| < 2^{-54}$, $|x_3| < 1.0001x^3$, and $|d_3| < 2^{-50.9999}x^3$. With the notations of Algorithm muldd, we tell Gappa that $|c_h/z| < 1.0001$ and $|c_\ell/z| < 2^{-50.9999}$, where z is a parameter representing x^3. Since the analysis is invariant when x^3 is multiplied by a power of two, we assume $z \in [1, 2]$. Gappa proves that the error of the `muldd` call, i.e., $|b_h + b_\ell - (a_h + a_\ell)(x_3 + d_3)|$, is bounded by $2^{-102.264}z$, whatever the rounding mode. The corresponding Gappa script is the following (run with `-Echange-threshold=0`):

```
@rnd = float<ieee_64,RND>;
@ufp = float<1,aw>;
l1 = ch*xl;
l1r rnd= ch*xl;
```

```
l2 = cl*xh;
l2r rnd= cl*xh;
h1 = xh*ch;
h1r rnd= xh*ch;
l3 = -(h1-xh*ch);
l3r = -(h1r - xh*ch);
t = l2+l1;
tr rnd= l2r+l1r;
l4 = l3+t;
l4r rnd= l3r+tr;
hr rnd= h1r+l4r;
ur rnd= h1r-hr;
lr = rnd(h1r+l4r-hr);
ufp_hr ufp= h1r+l4r;
{ z in [1,2] /\
  xh in [0.33,0.34642] /\
  xl in [-1b-54,1b-54] /\
  ch/z in [-1.0001, 1.0001]
  /\ cl/z in
  [-4.4412e-16,4.4412e-16] ->
  |(hr + lr - (ch+cl)*(xh+xl)) / z|
  in ? }
hr + lr - (ch+cl)*(xh+xl) ->
(lr - (h1r+l4r-hr))
+ (l4r - (l3r + tr))
+ (tr - (l2 + l1)) - cl * xl;
$ z in 600;
```

The largest error we were able to find is about $2^{-102.967}x^3$ with $x = 0\text{x}1.020\text{fab}521\text{a}868\text{p}{-}18$ and rounding towards zero, which is about 61% of the Gappa bound.

Replacing the goal by `|hr/z|` or `|lr/z|`, we get respectively $b_h < 0.347x^3$ and $|b_\ell| < 7.6929 \cdot 10^{-17}x^3 < 2^{-53.529}x^3$.

F. Replacing `muldd` by `muldd_fast`

We analyze here the impact of replacing `muldd` by `muldd_fast` in line 20 of Algorithm AtanhSmall.

Since `muldd_fast` simply consists in returning in place of h and ℓ the values h_1 and ℓ_4 in `muldd`, we simplify the Gappa script from §E as follows, where for clarity we keep the names h_1 and ℓ_4 for the return values:

```
@rnd = float<ieee_64,RND>;
l1 = ch*xl;
l1r rnd= ch*xl;
l2 = cl*xh;
l2r rnd= cl*xh;
h1 = xh*ch;
h1r rnd= xh*ch;
l3 = -(h1-xh*ch);
l3r = -(h1r - xh*ch);
t = l2+l1;
tr rnd= l2r+l1r;
l4 = l3+t;
l4r rnd= l3r+tr;
{ z in [1,2] /\
  xh in [0.33,0.34642] /\
  xl in [-1b-54,1b-54] /\
  ch/z in [-1.0001, 1.0001]
  /\ cl/z in
  [-4.4412e-16,4.4412e-16] ->
  |(h1r + l4r - (ch+cl)*(xh+xl)) / z|
  in ? }
h1r + l4r - (ch+cl)*(xh+xl) ->
  (l4r - (l3r + tr))
  + (tr - (l2 + l1)) - cl * xl;
$ z in 300;
```

Note that we split z into 300 sub-ranges instead of 600 in §E, this is enough here. This script proves, with the notations of Algorithm AtanhSmall:

$$|b_h + b_\ell - (a_h + a_\ell)(x_3 + d_3)| < 2^{-102.334}x^3.$$

Note this bound is better than the bound $2^{-102.264}x^3$ we obtained for `muldd`. This is because the final fasttwosum call in `muldd` might yield some additional rounding error. On the other hand, since there is no normalization using fasttwosum, the value of b_ℓ might be larger in absolute value. Indeed, by modifying the goal, Gappa proves $|b_h| < 0.347x^3$ (like for `muldd`) and $|b_\ell| < 2.8630 \cdot 10^{-16}x^3 < 2^{-51.633}x^3$, which is worse than the bound $|b_\ell| < 2^{-53.529}x^3$ for `muldd`.

Replacing $2^{-102.264}x^3$ by $2^{-102.334}x^3$, the term $2^{-101.991}x^3$ in the bound for $|b_h + b_\ell - (p(x) - x)|$ improves to $2^{-102.049}$.

In the analysis of $p_\ell \leftarrow b_\ell + t_\ell$, now $|b_\ell|$ is bounded by $2^{-51.633}x^3$ instead of $2^{-53.529}x^3$, from which we deduce $|p_\ell| < 2^{-51.967}x + 2^{-51.632}x^3$. The rounding error on p_ℓ is thus bounded by $2^{-103.967}x + 2^{-103.632}x^3$. Thus the x^3 term in the upper bound for $|p_h + p_\ell - p(x)|$ changes from $2^{-101.871}x^3$ to $(2^{-102.049} + 2^{-103.632})x^3 < 2^{-101.633}x^3$. This term propagates unchanged in Eq. (3).

Then in the analysis of ϵ, the bound for $|p_\ell|$ when $x \leq 1/4$ increases from $2^{-51.936}x$ to $2^{-51.857}x$, the bound for $|p_\ell \pm \epsilon|$ increases from $2^{-51.931}x$ to $2^{-51.852}x$, and thus the rounding error on $p_\ell \pm \epsilon$ is now bounded by $2^{-103.852}x$ instead of $2^{-103.931}x$. Finally, the lower bound for $|p_h + p_\ell - \ell_b|$ becomes:

$$|p_h + p_\ell - \ell_b| > 2^{-104.166}x + 2^{-52.1424}x^5.$$

The new ratio $f(x)/g(x)$ is still smaller than 1 for $x \geq 2^{-12}$, with $g - f' \geq 2^{-56.5}x^5 - 2^{-105.1}x$ using the same notations as in §B, thus Theorem 1 still holds for $x \geq 2^{-12}$.

For $|x| < 2^{-12}$, the analysis of the end of the proof of Theorem 1 is modified as follows. We use the slightly larger range $\alpha \in [1.122, 2 - 2^{-24}]$. The rounding error on p_ℓ is still bounded by $2^{-104}\text{ufp}(x)$, where $2^{-104}\text{ufp}(x) < 2^{-104}/\alpha \cdot x < 2^{-104.16607}x$, thus Eq. (3) becomes:

$$|p_h + p_\ell - \text{atanh}(x)| < 2^{-104.166}x + 2^{-52.2175}x^5,$$

and again this is smaller than the above lower bound for $|p_h + p_\ell - \ell_b|$.

Like in the proof of Theorem 1, the remaining ranges $\alpha \in [1, 1.122]$ and $\alpha \in [2 - 2^{-24}, 2)$ were checked using the exhaustive search algorithm from §II-C.