The CORE-MATH cos is correctly rounded

Paul Zimmermann
November 9, 2026

***Abstract*—We prove that the CORE-MATH binary64 cosine function is correctly rounded.**

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

We consider the CORE-MATH cos.c file from commit `9b821fc`, and the four rounding modes from the C23 standard: to nearest (ties to even), toward zero, toward $-\infty$, toward $+\infty$. Since $\cos(-x) = \cos(x)$, the CORE-MATH code first compute $\cos|x|$, thus we restrict our analysis to the case $x \geq 0$. Our proofs work both with and without FMA contraction: in case of an expression $a \cdot b + c$, we bound using Gappa or by hand the rounding error for the non-FMA case $\circ(\circ(ab) + c)$: using an FMA as in $\circ(ab + c)$ suppresses the error on rounding ab, and the computed bound for $\circ(ab) + c$ also holds for $ab + c$, thus the outer error cannot be larger. For reproducibility of our results, all scripts used are available in the public repository core-math-proofs, and which scripts are used for each statement is indicated in comment in the source file cos.tex of this document.

The CORE-MATH code for the cosine function is largely inspired from that for the sine function. In particular it uses the same tables, which can thus be shared between both codes.

I. Branch $0 \leq x < 2^{-27}$

For $|x| < 2^{-27}$, CORE-MATH returns $\mathrm{fma}(x, -x, 1)$. This special branch is only needed to avoid a spurious underflow in the polynomial evaluation used in the other branches.

Lemma 1: For $|x| < 2^{-27}$, $\mathrm{fma}(x, -x, 1)$ yields the correct rounding of $\cos(x)$, whatever the rounding mode.

Proof: Since the formula $\mathrm{fma}(x, -x, 1)$ is an even function, like $\cos(x)$, it suffices to consider $0 \leq x < 2^{-27}$. For $x = \pm 0$, it yields 1 whatever the rounding mode, which is the correct rounding. Now for $0 < x < 1$, we have:

$$1 - \frac{x^2}{2} < \cos(x) < 1. \tag{1}$$

It follows from Eq. (1) that $1 - 2^{-55} < \cos(x) < 1$ when $0 \leq x < 2^{-27}$. Similarly, $1 - 2^{-54} < 1 - x^2 < 1$. Since there is no rounding boundary in $(1 - 2^{-54}, 1)$, $\cos(x)$ and $\mathrm{fma}(x, -x, 1)$ round to the same value: 1 for rounding to nearest or upward, and $\mathrm{nextdown}(1)$ for rounding toward zero or downward. ■

II. Branch $2^{-27} \leq x < 2^{31}$

In this branch, Algorithm ModerateFastPath is used. It reuses the algorithm from [3], adapted to the computation of $\cos(x)$. The differences with the algorithm from [3] are the following:

- the sign of x is not needed, since $\cos(-x) = \cos(x)$. Instead, the final sign is swapped if the argument reduction is by an odd multiple of π (line 18);
- instead of computing a double-double approximation $S_h + S_\ell$ of $\sin(t_1 + t_2)$, we compute a double-double approximation $C_h + C_\ell$ of $\cos(t_1 + t_2)$ (line 11), where $t_1 = i_1\pi/2^7$ and $t_2 = i_2\pi/2^{14}$;
- instead of computing a double approximation C_h of $\cos(t_1 + t_2)$, we compute a double approximation S_h of $\sin(t_1 + t_2)$ (line 12);
- the computation at line 17 is changed: since $C_h + C_\ell$ approximates $\cos(z)$ where $z = t_1 + t_2$, S_h approximates $\sin(z)$, $1 + c_h$ approximates $\cos(r)$ and s_h approximates $\sin(r)$, $\cos(z + r)$ is approximated by:

$$(C_h + C_\ell)(1 + c_h) - S_h s_h \approx C_h + (C_\ell + C_h c_h - S_h s_h).$$

For the analysis of algorithms identical to the sine function, we refer to [3]. However, we keep all algorithms so that this paper is self-content. Like for the sine function, at lines 15-17, we omitted roundings for clarity, and in line 17, the terms in f_ℓ are accumulated from left to right, as prescribed by the C language.

Algorithm 1 (ModerateFastPath)

Input: x a binary64 number, $2^{-27} \leq |x| < 2^{31}$
Output: the correct rounding $\circ(\cos x)$ or FAIL

1: $i = \texttt{0x1.45f306dc9c883p+12}$
2: $p_h = \texttt{-0x1.921fb54442d18p-13}$
3: $p_\ell = \texttt{-0x1.1a62633145c07p-67}$
4: $k \leftarrow \mathrm{round}(\circ(i \cdot |x|))$
5: $r_h \leftarrow \mathrm{fma}(k, p_h, |x|), \quad r_\ell \leftarrow \circ(k \cdot p_\ell)$
6: $r = \circ(r_h + r_\ell)$
7: $r_2 \leftarrow \circ(r \cdot r)$
8: write $k \bmod 2^{15} = 2^{14}t + 2^7 i_1 + i_2$ with $t \in \{0, 1\}$ and $0 \leq i_1, i_2 < 2^7$
9: $c_{1h}, c_{1\ell} \leftarrow \mathrm{muldd}(C_{1h}[i_1], C_{1\ell}[i_1], C_{2h}[i_2], C_{2\ell}[i_2])$
10: $c_{2h}, c_{2\ell} \leftarrow \mathrm{muldd}(S_{1h}[i_1], S_{1\ell}[i_1], S_{2h}[i_2], S_{2\ell}[i_2])$
11: $C_h, C_\ell \leftarrow \mathrm{fastsum}(c_{1h}, c_{1\ell}, -c_{2h}, -c_{2\ell})$
12: $S_h \leftarrow \circ(\circ(S_{1h}[i_1]C_{2h}[i_2]) + \circ(C_{1h}[i_1]S_{2h}[i_2]))$
13: $s_1 = 1.0, \quad s_3 = \texttt{-0x1.55555553068fp-3}$
14: $c_2 = -0.5, \quad c_4 = \texttt{0x1.55555553bfd3p-5}$
15: $s_h \leftarrow r \cdot (s_1 + s_3 \cdot r_2)$
16: $c_h \leftarrow r_2 \cdot (c_2 + c_4 \cdot r_2)$
17: $f_h \leftarrow C_h, \quad f_\ell \leftarrow C_\ell + C_h c_h - S_h s_h$
18: $f_h \leftarrow (-1)^t \cdot f_h, \quad f_\ell \leftarrow (-1)^t \cdot f_\ell$
19: $\epsilon \leftarrow \texttt{0x1.dep-64}$
20: $\ell_b \leftarrow f_h + \circ(f_\ell - \epsilon), \quad u_b \leftarrow f_h + \circ(f_\ell + \epsilon)$
21: if $\circ(\ell_b) = \circ(u_b)$ return $\circ(\ell_b)$ else return FAIL

The tables S_{1h}, $S_{1\ell}$, C_{1h}, $C_{1\ell}$, S_{2h}, $S_{2\ell}$, C_{2h}, $C_{2\ell}$ are exactly the same as in [3].

Algorithm 2 (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: $h \leftarrow \circ(x_h c_h)$
2: $\ell_1 \leftarrow \mathrm{fma}(x_h, c_h, -h)$
3: $\ell_2 \leftarrow \circ(x_h c_\ell)$
4: $\ell_3 \leftarrow \circ(x_\ell c_h)$
5: $\ell \leftarrow \circ(\circ(\ell_2 + \ell_3) + \ell_1)$

Algorithm 3 (fasttwosum) from [1]

Input: a, b binary64 numbers
Output: x, y such that $x + y$ approximates $a + b$
1: $x \leftarrow \circ(a + b)$
2: $z \leftarrow \circ(x - a)$
3: $y \leftarrow \circ(b - z)$

Lemma 2 (Lemma 2 from [3]): The integer k computed at line 4 of Algorithm ModerateFastPath satisfies:

$$|x| = \frac{k\pi}{2^{14}} + y,$$

with $k \leq 2^{44}$ and $|y| < 2^{-13.340}$.
See proof in [3].

Lemma 3 (Lemma 3 from [3]): At line 5, r_h is exact, i.e., $r_h = k \cdot p_h + |x|$, and after line 6:

$$|r - y| < 2^{-65.996},$$

with $|r|, |r_h| < 2^{-13.339}$.
See proof in [3].

Theorem 1: For $2^{-27} \leq |x| < 2^{31}$, and any rounding mode, Algorithm ModerateFastPath is correct, i.e., if not FAIL, it delivers $\circ(\cos(x))$.

Proof: Since c_{1h} and c_{2h} only depend on i_1, i_2 and the rounding mode, we can check by exhaustive search whether the FastTwoSum condition is fulfilled in $\mathrm{fastsum}(c_{1h}, c_{1\ell}, -c_{2h}, -c_{2\ell})$, which calls $\mathrm{fasttwosum}(c_{1h}, -c_{2h})$: this is the case. Similarly, since C_h, C_ℓ, S_h only depend on i_1, i_2 and the rounding mode, we can determine by exhaustive search a bound on the difference between $C_h + C_\ell$ and $\cos(t_1 + t_2)$, and between S_h and $\sin(t_1 + t_2)$, where $t_1 = i_1\pi/2^7$ and $t_2 = i_2\pi/2^{14}$. Without FMA contraction, the largest error for $\cos(t_1 + t_2)$ is obtained for $i_1 = 111$, $i_2 = 36$ and rounding downward:

$$|C_h + C_\ell - \cos(t_1 + t_2)| < 2^{-103.342}, \tag{2}$$

while the largest error for $\sin(t_1 + t_2)$ is obtained for $i_1 = 42$, $i_2 = 73$ and rounding upward:

$$|S_h - \sin(t_1 + t_2)| < 2^{-51.569}.$$

(We tried all the possible FMA contractions for the computation of S_h, and within muldd for the computation of C_h and C_ℓ.) This exhaustive search also shows that $|C_h| \leq 1$, $|C_\ell| < 2^{-51.569}$ and $|S_h| \leq 1$.

Algorithm 4 (fastsum)

Input: x_h, x_ℓ, y_h, y_ℓ binary64 numbers
Output: h, ℓ such that $h + \ell$ approximates $x_h + x_\ell + y_h + y_\ell$
1: $h, t \leftarrow \mathrm{fasttwosum}(x_h, y_h)$
2: $\ell \leftarrow \circ(\circ(x_\ell + y_\ell) + t)$

Since the argument reduction and the evaluation of s_h and c_h are identical to [3], we have as in the proof of Theorem 1 from [3]:

$$\begin{aligned} |s_h - \sin(y)| &< 2^{-64.516}, \\ |c_h - (\cos(y) - 1)| &< 2^{-78.316}. \end{aligned}$$

Let $x' = z + y$, where $z = t_1 + t_2$. The argument reduction gives $\cos x = (-1)^t \cos(x')$, including when $z + y$ crosses a multiple of π. We first bound the un-signed approximation:

$$\begin{aligned} &|C_h + C_\ell + C_h c_h - S_h s_h - \cos(x')| \\ &\leq |C_h + C_\ell - \cos(z)| + |C_h c_h - \cos(z)(\cos(y) - 1)| \\ &+ |S_h s_h - \sin(z)\sin(y)|. \end{aligned}$$

The first term of the right-hand side $|C_h + C_\ell - \cos(z)|$ is bounded by $2^{-103.342}$ from Eq. (2). The second term is bounded as follows:

$$\begin{aligned} &|C_h c_h - \cos(z)(\cos(y) - 1)| \\ &\leq |C_h||c_h - (\cos(y) - 1)| + |C_h - \cos(z)||\cos(y) - 1| \\ &\leq 2^{-78.316} + (2^{-51.569} + 2^{-103.342}) \cdot 2^{-27.680} < 2^{-77.708}, \end{aligned}$$

since $|C_h| \leq 1$, $|C_h - \cos(z)| \leq |C_\ell| + 2^{-103.342}$, and $|y| \leq 2^{-13.340}$. Finally, the last term is bounded by:

$$\begin{aligned} &|S_h s_h - \sin(z)\sin(y)| \\ &\leq |S_h||s_h - \sin(y)| + |S_h - \sin(z)||\sin(y)| \\ &\leq 2^{-64.516} + 2^{-51.569} \cdot 2^{-13.340} < 2^{-63.699}. \end{aligned}$$

Summing up, we obtain:

$$\begin{aligned} &|C_h + C_\ell + C_h c_h - S_h s_h - \cos(x')| \\ &< 2^{-103.342} + 2^{-77.708} + 2^{-63.699} < 2^{-63.698}. \end{aligned}$$

We still have to take into account the rounding errors while computing f_ℓ in line 17, and that in $\circ(f_\ell \pm \epsilon)$ in line 20. Since $|C_\ell| < 2^{-51.569}$, $|C_h| \leq 1$, $|c_h| \leq |\cos(y) - 1| + 2^{-78.316} < 2^{-27.680}$, the rounding error on $C_h c_h$ is bounded by $\mathrm{ulp}(2^{-27.680}) = 2^{-80}$, and that on $C_\ell + \circ(C_h c_h)$ is bounded by $\mathrm{ulp}(2^{-27.679}) = 2^{-80}$ too. Let $\alpha = \circ(C_\ell + \circ(C_h c_h))$ and $\beta = \circ(S_h s_h)$. The difference $\alpha - \beta$ is thus bounded in absolute value by:

$$2^{-51.569} + 2^{-27.680} + 2^{-13.340} + 2 \cdot 2^{-80} + 2^{-66} < 2^{-13.339},$$

thus the rounding error on $f_\ell = \circ(\alpha - \beta)$ is bounded by $\mathrm{ulp}(2^{-14}) = 2^{-66}$ too. Finally adding $\pm\epsilon$ to f_ℓ does not make it exceed 2^{-13} in absolute value either, thus the rounding error in $\circ(f_\ell \pm \epsilon)$ is bounded by 2^{-66} too. In fact, since $\epsilon = 7.46875 \cdot \mathrm{ulp}(2^{-14})$, the rounding error on $f_\ell \pm \epsilon$ is bounded by $0.53125 \cdot \mathrm{ulp}(2^{-14}) < 2^{-66.912}$, which is the maximal distance from ϵ

to a multiple of $\mathrm{ulp}(f_\ell)$. If f_ℓ and $f_\ell \pm \epsilon$ are in a smaller binade, this bound also holds. If f_ℓ is in a smaller binade, but $|f_\ell \pm \epsilon|$ lies in the binade $[2^{-14}, 2^{-13})$, the sum of the rounding errors on f_ℓ and $f_\ell \pm \epsilon$ is bounded by $2^{-67} + 2^{-66}$. Thus in all cases, the accumulated rounding errors in lines 17 and 20 are bounded by:

$$2 \cdot 2^{-80} + 2 \cdot 2^{-66} + 2^{-66.912} < 2^{-64.659}.$$

We deduce (after fixing the sign of $\cos(x)$):

$$\ell_b - \cos(x) < -\epsilon + 2^{-63.698} + 2^{-64.659} < 0,$$

and similarly $u_b - \cos(x) > 0$, thus ℓ_b and u_b enclose $\cos(x)$, and if they round to the same value, so does $\cos(x)$. ∎

III. Branch $x \geq 2^{31}$

Like for the sine function, Payne-Hanek's argument reduction is used. This is Algorithm ReduceLarge, which is identical to the one used for the sine function.

Algorithm 5 (ReduceLarge)

Input: x a binary64 number, $x \geq 2^{31}$
Output: integer k and binary64 number r such that $x/(2\pi) \approx k/2^{15} + r \bmod 1$

1: use a precomputed table T such that $\sum_j T_j/2^{64j}$ is the 64-bit expansion of $1/(2\pi)$, with $T_0 = 0$
2: write x as $m \cdot 2^e$ with $2^{52} \leq m < 2^{53}$
3: $i \leftarrow \lceil (e+1)/64 \rceil$
4: $f \leftarrow e \bmod 64$ ▷ $0 \leq f < 64$
5: $U_0 \leftarrow (T_i, T_{i+1})$ shifted by f bits to the left
6: $U_1 \leftarrow (T_{i+1}, T_{i+2})$ shifted by f bits to the left
7: $u \leftarrow U_0 \cdot 2^{64} + U_1$ ▷ 128-bit variable
8: $v \leftarrow m \cdot u$ ▷ 128-bit multiplication
9: $v \leftarrow (v + 2^{112} + 2^{59} + 2^{52}) \bmod 2^{128}$
10: write $v = v_h \cdot 2^{113} + v_\ell$ with $0 \leq v_\ell < 2^{113}$
11: $k \leftarrow v_h, \quad r \leftarrow 2^{-68} \lfloor 2^{-60} v_\ell \rfloor - 2^{-16}$

Lemma 4 (Lemma 4 from [3]): Let x be a binary64 number, $x \geq 2^{31}$. Then the values k and r returned by Algorithm ReduceLarge satisfy:

$$\frac{x}{2\pi} = \frac{k}{2^{15}} + r + s \bmod 1, \tag{3}$$

with $|r| \leq 2^{-16}$ and $|s| < 2^{-68.988}$.
See proof in [3].

The fast path for this branch is detailed in Algorithm LargePath.

Theorem 2: For $x \geq 2^{31}$, and any rounding mode, Algorithm LargePath is correct, i.e., if not FAIL, it delivers $\circ(\cos(x))$.
Proof: From Lemma 4, we have

$$\frac{x}{2\pi} = \frac{k}{2^{15}} + r + s \bmod 1,$$

with $|r| \leq 2^{-16}$ and $|s| < 2^{-68.988}$. Let $x' = z + y$, with t defined in line 2. Since $k = 2^{14}t + 2^7 i_1 + i_2$:

$$\cos(x) = (-1)^t \cos(x'),$$

Algorithm 6 (LargePath)

Input: x a binary64 number, $x \geq 2^{31}$
Output: the correct rounding $\circ(\cos x)$ or FAIL

1: $k, r \leftarrow \mathrm{ReduceLarge}(x)$
2: write $k = 2^{14}t + 2^7 i_1 + i_2$ with $t \in \{0,1\}$, $0 \leq i_1, i_2 < 2^7$
3: $c_{1h}, c_{1\ell} \leftarrow \mathrm{muldd}(C_{1h}[i_1], C_{1\ell}[i_1], C_{2h}[i_2], C_{2\ell}[i_2])$
4: $c_{2h}, c_{2\ell} \leftarrow \mathrm{muldd}(S_{1h}[i_1], S_{1\ell}[i_1], S_{2h}[i_2], S_{2\ell}[i_2])$
5: $C_h, C_\ell \leftarrow \mathrm{fastsum}(c_{1h}, c_{1\ell}, -c_{2h}, -c_{2\ell})$
6: $S_h \leftarrow \circ(\circ(S_{1h}[i_1] C_{2h}[i_2]) + \circ(C_{1h}[i_1] S_{2h}[i_2]))$
7: $s_1 = \texttt{0x1.921fb54442d18p2}$
8: $s_3 = \texttt{-0x1.4abbcdb6b26d1p5}$
9: $c_2 = \texttt{-0x1.3bd3cc9be45dep4}$
10: $c_4 = \texttt{0x1.03c1eee483083p6}$
11: $r_2 \leftarrow \circ(r \cdot r)$
12: $s_h \leftarrow r \cdot (s_1 + s_3 \cdot r_2)$
13: $c_h \leftarrow r_2 \cdot (c_2 + c_4 \cdot r_2)$
14: $f_h \leftarrow C_h, \quad f_\ell \leftarrow C_\ell + C_h c_h - S_h s_h$
15: $f_h \leftarrow (-1)^t \cdot f_h, \quad f_\ell \leftarrow (-1)^t \cdot f_\ell$
16: $\epsilon \leftarrow \texttt{0x1.01p-63}$
17: $\ell_b \leftarrow f_h + \circ(f_\ell - \epsilon), \quad u_b \leftarrow f_h + \circ(f_\ell + \epsilon)$
18: if $\circ(\ell_b) = \circ(u_b)$ return $\circ(\ell_b)$ else return FAIL

with $z = t_1 + t_2$, $t_1 = \pi i_1/2^7$, $t_2 = \pi i_2/2^{14}$, and $y = 2\pi(r+s)$.

Since the reduced argument r is the same as in [3], and similarly for the polynomials $s_1 r + s_3 r^3$ and $c_2 r^2 + c_4 r^4$, we have as in [3]:

$$\begin{aligned} |s_h - \sin(y)| &< 2^{-64.468}, \\ |c_h - (\cos(y) - 1)| &< 2^{-78.267}. \end{aligned}$$

Then since $\cos(x') = \cos(z+y)$:

$$\begin{aligned} &|C_h + C_\ell + C_h c_h - S_h s_h - \cos(x')| \\ &\leq |C_h + C_\ell - \cos(z)| + |C_h c_h - \cos(z)(\cos(y) - 1)| \\ &+ |S_h s_h - \sin(z)\sin(y)|. \end{aligned}$$

The first term of the right-hand side $|C_h + C_\ell - \cos(z)|$ is bounded by $2^{-103.342}$ from Eq. (2). The second term is bounded as follows:

$$\begin{aligned} &|C_h c_h - \cos(z)(\cos(y) - 1)| \\ &\leq |C_h||c_h - (\cos(y) - 1)| + |C_h - \cos(z)||\cos(y) - 1| \\ &\leq 2^{-78.267} + (2^{-51.569} + 2^{-103.342}) \cdot 2^{-27.696} < 2^{-77.681}, \end{aligned}$$

since $|C_h| \leq 1$, $|C_h - \cos(z)| \leq |C_\ell| + 2^{-103.342}$, and $|y| \leq 2^{-13.348}$. Finally, the last term is bounded by:

$$\begin{aligned} &|S_h s_h - \sin(z)\sin(y)| \\ &\leq |S_h||s_h - \sin(y)| + |S_h - \sin(z)||\sin(y)| \\ &\leq 2^{-64.468} + 2^{-51.569} \cdot 2^{-13.348} < 2^{-63.675}. \end{aligned}$$

Summing up, we obtain:

$$\begin{aligned} &|C_h + C_\ell + C_h c_h - S_h s_h - \cos(x')| \\ &< 2^{-103.342} + 2^{-77.681} + 2^{-63.675} < 2^{-63.674}. \end{aligned}$$

We still have to take into account the rounding errors while computing f_ℓ in line 14, and that in $\circ(f_\ell \pm \epsilon)$ in line 17. Since $|C_\ell| < 2^{-51.569}$, $|C_h| \le 1$, $|c_h| \le |\cos(y) - 1| + 2^{-78.267} < 2^{-27.696}$, the rounding error on $C_h c_h$ is bounded by $\mathrm{ulp}(2^{-27.696}) = 2^{-80}$, and that on $C_\ell + \circ(C_h c_h)$ is bounded by $\mathrm{ulp}(2^{-27.695}) = 2^{-80}$ too. Now since $|S_h| \le 1$ and $|s_h| \le |\sin(y)| + 2^{-64.468} < 2^{-13.348}$, the rounding error on $S_h s_h$ is bounded by $\mathrm{ulp}(2^{-14}) = 2^{-66}$. Let $\alpha = \circ(C_\ell + \circ(C_h c_h))$ and $\beta = \circ(S_h s_h)$. The difference $\alpha - \beta$ is thus bounded in absolute value by:

$$2^{-51.569} + 2^{-27.696} + 2^{-13.348} + 2 \cdot 2^{-80} + 2^{-66} < 2^{-13.347},$$

thus the rounding error on $f_\ell = \circ(\alpha - \beta)$ is bounded by $\mathrm{ulp}(2^{-14}) = 2^{-66}$ too. Finally adding $\pm\epsilon$ to f_ℓ does not make it exceed 2^{-13} in absolute value either, thus the rounding error in $\circ(f_\ell \pm \epsilon)$ is bounded by 2^{-66} too. The accumulated rounding errors in lines 14 and 17 are thus bounded by:

$$2 \cdot 2^{-80} + 3 \cdot 2^{-66} < 2^{-64.414}.$$

We deduce (after fixing the sign of $\cos(x)$):

$$\ell_b - \cos(x) < -\epsilon + 2^{-63.674} + 2^{-64.414} < 0,$$

and similarly $u_b - \cos(x) > 0$, thus ℓ_b and u_b enclose $\cos(x)$, and if they round to the same value, so does $\cos(x)$. ∎

IV. Accurate path

Contrary to the sine function, a single algorithm is sufficient here for the accurate path (AccurateLarge). Indeed, Algorithm AccurateLarge computes an approximation of $\cos(x)$ (or of $\sin(x)$ in [3]) with *absolute* accuracy about 2^{-128}. When x is near 2^{-26}, $\sin(x)$ is near 2^{-26} too, thus the relative accuracy decreases to about 102 bits, which is not enough for hard-to-round cases. In contrary, $\cos(x)$ is near 1 for x small, thus the relative accuracy matches the absolute one.

Algorithm AccurateLarge first performs an argument reduction using Payne-Hanek's algorithm (Algorithm ReduceAccLarge), which is exactly the same as in [3], except the input can be as small as 2^{-27} here, instead of 2^{-16} in [3].

Lemma 5 (Lemma 2 from [3]): Let x be a binary64 number, $x \ge 2^{-27}$. Let k, σ, r be the values returned by Algorithm ReduceAccLarge, and denote $R = r/2^{128}$, then:

$$\frac{x}{2\pi} = \frac{k}{2^{13}} + (-1)^\sigma R + s \bmod 1, \tag{4}$$

with $|(-1)^\sigma R| \le 2^{-14}$ and $0 \le s < 2^{-127.999}$.

Since Algorithm ReduceAccLarge is almost identical to [3], we refer to the proof of Lemma 2 in [3]. (When $2^{-27} \le x < 2^{-16}$, we have $-79 \le e \le -69$, thus $i = -1$, and we take $T_{-1} = 0$.)

Algorithm AccurateLarge uses auxiliary algorithms mhUU, evalPS and evalPC described below. Some remarks about Algorithm AccurateLarge:

- this algorithm mainly deals with 128-bit unsigned integers representing values in $[0, 1)$. In the proof we use small letters to denote these 128-bit unsigned integers, for example r at line 1, and capital letters to denote the corresponding rational value, for example $R = r/2^{128}$;
- the precomputed values $S_1[i_1] := s_1[i_1]/2^{128}$ and $C_1[i_1] := c_1[i_1]/2^{128}$ approximate $\sin(t_1)$ and $|\cos(t_1)|$ where $t_1 = \pi i_1/2^6$. We have $\sin(t_1) \ge 0$, $\cos(t_1) \ge 0$ for $i_1 < 32$, and $\cos(t_1) \le 0$ for $32 \le i_1 < 64$;
- the precomputed values $S_2[i_2] := s_2[i_2]/2^{128}$ and $C_2[i_2] := c_2[i_2]/2^{128}$ approximate $\sin(t_2)$ and $|\cos(t_2)|$ where $t_2 = \pi i_2/2^{12}$. We have $\sin(t_2), \cos(t_2) \ge 0$;
- since all computations are done with integers (except the final rounding done by routine u128tod at line 16), the error analysis does not depend on the rounding mode;
- line 6 uses the relation $\sin(t_1 + t_2) = \sin(t_1)\cos(t_2) +$

Algorithm 7 (ReduceAccLarge)

Input: x a binary64 number, $x \ge 2^{-27}$
Output: integer k, sign bit σ and 128-bit integer r such that $x/(2\pi) \approx k/2^{13} + (-1)^\sigma \cdot r/2^{128} \bmod 1$

1: use a precomputed table T such that $\sum_j T_j/2^{64j}$ is the 64-bit expansion of $1/(2\pi)$, with $T_{-1} = T_0 = 0$
2: write x as $m \cdot 2^e$ with $2^{52} \le m < 2^{53}$
3: $i \leftarrow \lceil (e+1)/64 \rceil$
4: $f \leftarrow e \bmod 64$ ▷ $0 \le f < 64$
5: $U_0 \leftarrow (T_i, T_{i+1})$ shifted by f bits to the left
6: $U_1 \leftarrow (T_{i+1}, T_{i+2})$ shifted by f bits to the left
7: $U_2 \leftarrow (T_{i+2}, T_{i+3})$ shifted by f bits to the left
8: $u \leftarrow U_0 \cdot 2^{64} + U_1$ ▷ 128-bit variable
9: $v \leftarrow (m \cdot u + \lfloor (m \cdot U_2)/2^{64} \rfloor) \bmod 2^{128}$
10: write $v = k \cdot 2^{115} + r$ with $0 \le r < 2^{115}$
11: if $r < 2^{114}$ then $\sigma \leftarrow 0$
12: else $k \leftarrow (k+1) \bmod 2^{13}$, $\sigma \leftarrow 1$, $r \leftarrow 2^{115} - r$

Algorithm 8 (AccurateLarge)

Input: x a binary64 number, $x \ge 2^{-27}$
Output: the correct rounding of $\cos x$

1: $k, \sigma_r, r \leftarrow \mathrm{ReduceAccLarge}(x)$
2: if $k \ge 2^{12}$ then $\sigma \leftarrow 1$ else $\sigma \leftarrow 0$
3: write $k \bmod 2^{12} = i_1 2^6 + i_2$ with $0 \le i_1, i_2 < 2^6$
4: $s_{t_1}, c_{t_1} \leftarrow s_1[i_1], c_1[i_1]$, $s_{t_2}, c_{t_2} \leftarrow s_2[i_2], c_2[i_2]$
5: $\lambda \leftarrow \mathrm{mhUU}(s_{t_1}, c_{t_2})$, $\mu \leftarrow \mathrm{mhUU}(c_{t_1}, s_{t_2})$
6: if $i_1 < 32$ then $s_z \leftarrow \lambda + \mu$ else $s_z \leftarrow \lambda - \mu$
7: $\lambda' \leftarrow \mathrm{mhUU}(c_{t_1}, c_{t_2})$, $\mu' \leftarrow \mathrm{mhUU}(s_{t_1}, s_{t_2})$
8: if $i_1 < 32$ then $c_z \leftarrow \lambda' - \mu'$ else $c_z \leftarrow \lambda' + \mu'$
9: $s_r \leftarrow \mathrm{evalPS}(r)$, $c_r \leftarrow \mathrm{evalPC}(r)$
10: $w_1 \leftarrow \mathrm{mhUU}(c_z, c_r)$, $w_2 \leftarrow \mathrm{mhUU}(s_z, s_r)$
11: if $i_1 \ge 32$ then $\sigma \leftarrow 1 - \sigma$
12: if ($i_1 \ge 32$ and $\sigma_r = 0$) or ($i_1 < 32$ and $\sigma_r = 1$) then $c \leftarrow w_1 + w_2$
13: elif $w_1 \le w_2$ then $c \leftarrow w_2 - w_1$, $\sigma \leftarrow 1 - \sigma$
14: else $c \leftarrow w_1 - w_2$
15: if $2^{64} - 5 \le c \bmod 2^{64} \le 2^{64} - 2$ and $|x|$ is one of the four listed exceptional inputs then return its hard-coded correct rounding
16: return $\mathrm{u128tod}(c, \sigma)$

$\cos(t_1)\sin(t_2)$, adjusting the sign when $i_1 \geq 32$ since then $\cos(t_1) \leq 0$ but the value stored in c_{t_1} is nonnegative. Then S_z approximates $\sin(2\pi z)$ with $2\pi z = t_1 + t_2$;

- line 8 uses the relation $\cos(t_1 + t_2) = \cos(t_1)\cos(t_2) - \sin(t_1)\sin(t_2)$, adjusting the sign when $i_1 \geq 32$ since then $\cos(t_1) \leq 0$ but the value stored in c_{t_1} is nonnegative, and C_z approximates $|\cos(2\pi z)|$;
- line 11 flips the sign of the result when $i_1 \geq 32$, in which case $\cos(z) \leq 0$;
- line 15 checks for exceptional cases that would not be correctly rounded. There are only four such cases: `0x1.8000000000009p-23`, `0x1.8000000000024p-22`, `0x1.800000000009p-21` and `0x1.20000000000f3p-20`.

Algorithm mhUU is identical to [3].

Algorithm 9 (mhUU)

Input: a, b two integers, $0 \leq a, b < 2^{128}$
Output: c approximating $ab/2^{128}$

1: write $a = a_h 2^{64} + a_\ell$ with $0 \leq a_h, a_\ell < 2^{64}$
2: write $b = b_h 2^{64} + b_\ell$ with $0 \leq b_h, b_\ell < 2^{64}$
3: $h \leftarrow a_h b_h$
4: $m_1 \leftarrow a_h b_\ell, \quad m_2 \leftarrow a_\ell b_h$
5: $c \leftarrow h + \lfloor m_1/2^{64} \rfloor + \lfloor m_2/2^{64} \rfloor$

Lemma 6 (Lemma 6 from [3]): If a, b are two integers, $0 \leq a, b < 2^{128}$, $c = \mathrm{mhUU}(a, b)$, $A = a/2^{128}$, $B = b/2^{128}$, $C = c/2^{128}$, then:

$$0 \leq AB - C < 3 \cdot 2^{-128}.$$

We refer to [3] for the proof of this lemma.

Algorithm evalPS approximates $\sin(2\pi R)$ for $0 \leq R \leq 2^{-14}$. It is the same as in [3].

Algorithm 10 (evalPS)

Input: a 128-bit integer r, $0 \leq r \leq 2^{114}$
Output: s_r approximating $2^{128}\sin(2\pi r/2^{128})$

1: $c_1 = \texttt{0xc90fdaa22168c234c4c6628b80dbed80}$
2: $c_3 = \texttt{0x14abbce625be52beabb8e5506ca45b03}$
3: $c_5 = \texttt{0x00a335e33bad570e044c54995bb97635}$
4: $c_7 = \texttt{0x000265a5999d06c4}$
5: $u \leftarrow 8r$
6: $u_2 \leftarrow \mathrm{mhUU}(u, u)$
7: $u_4 \leftarrow \mathrm{mhUU}(u_2, u_2)$
8: $u_{2h} \leftarrow \lfloor u_2/2^{64} \rfloor$
9: $s_5 \leftarrow c_5 - c_7 u_{2h}$
10: $s_1 \leftarrow c_1 - \mathrm{mhUU}(c_3, u_2)$
11: $s_2 \leftarrow s_1 + \mathrm{mhUU}(s_5, u_4)$
12: $s_r \leftarrow \mathrm{mhUU}(s_2, u)$

Lemma 7 (Lemma 7 from [3]): Given as input $0 \leq r \leq 2^{114}$, the value s_r returned by Algorithm evalPS satisfies:

$$|S_r - \sin(2\pi R)| < 2^{-126.128}.$$

We refer to [3] for the proof of this lemma.

Similarly, Algorithm evalPC approximates $\cos(2\pi R)$.

Algorithm 11 (evalPC)

Input: a 128-bit integer r, $0 \leq r \leq 2^{114}$
Output: c_r approximating $2^{128}\cos(2\pi r/2^{128})$

1: $v_0 = \texttt{0xffffffffffffffffffffffffffffffff}$
2: $v_2 = \texttt{0x4ef4f326f91779692b71366cbcf0b096}$
3: $v_4 = \texttt{0x040f07c206d6b0ecd4c9076e3db78fed}$
4: $v_6 = \texttt{0x00155d3c7e3cbef9eec6962368fb7c41}$
5: $v_8 = \texttt{0x00003c3e845b5510}$
6: $u \leftarrow 8r$
7: $u_2 \leftarrow \mathrm{mhUU}(u, u)$
8: $u_4 \leftarrow \mathrm{mhUU}(u_2, u_2)$
9: $u_{2h} \leftarrow \lfloor u_2/2^{64} \rfloor$
10: $c_6 \leftarrow v_6 - v_8 u_{2h}$
11: $c_4 \leftarrow v_4 - \mathrm{mhUU}(c_6, u_2)$
12: $c_0 \leftarrow v_0 - \mathrm{mhUU}(v_2, u_2)$
13: $c_r \leftarrow c_0 + \mathrm{mhUU}(c_4, u_4)$

Lemma 8 (Lemma 8 from [3]): Given as input $0 \leq r \leq 2^{114}$, the value c_r returned by Algorithm evalPC satisfies:

$$|C_r - \cos(2\pi R)| < 2^{-125.966}.$$

We refer to [3] for the proof of this lemma.

Theorem 3: Given a binary64 number x, $x \geq 2^{-27}$, Algorithm AccurateLarge returns the correct rounding of $\cos x$, whatever the rounding mode.

Proof: We let the reader check that the sign of the result is correct: at line 1 we reduce x modulo 2π into $2\pi(z + (-1)^{\sigma_r} R + s)$, with $z = k/2^{13}$ and $R = r/2^{128}$. Then if $k \geq 2^{12}$, we flip the sign bit using $\cos(2\pi z) = -\cos(2\pi z - \pi)$, where z becomes $z - 1/2$. At the end, W_1 approximates $|\cos(2\pi z)\cos(2\pi R)|$ and W_2 approximates $|\sin(2\pi z)\sin(2\pi R)|$. If $i_1 < 32$ and $\sigma_r = 0$, then both z and R are in the first quadrant, so the cosine addition formula requires $w_1 - w_2$, with a sign adjustment if this difference is negative. It works similarly for the other cases. The routine u128tod first applies the sign bit to C, then rounds it according to the rounding mode.

Define $b = 1$ if $k \geq 2^{12}$, $b = 0$ otherwise, $z = (k \bmod 2^{12})/2^{13}$ and $R' = (-1)^{\sigma_r} R$. Lemma 5 gives

$$\frac{x}{2\pi} = b/2 + z + R' + s \bmod 1,$$

with $|s| < 2^{-127.999}$. Since $\cos x$ is contracting, it yields:

$$|(-1)^b \cos(2\pi(z + R')) - \cos(x)| < 2^{-125.347}. \qquad (5)$$

Since the computation of s_z and c_z at lines 5-8 is the same as in [3], we have from the proof of Theorem 4 of [3]:

$$\begin{aligned} |S_z - \sin(2\pi z)| &< 2^{-125.691}, \\ |C_z - |\cos(2\pi z)|| &< 2^{-125.691}, \end{aligned}$$

with $0 \leq s_z, c_z \leq 2^{128} - 3$.

After the computation of w_1 and w_2 at line 10, a Gappa script proves:

$$\begin{aligned} |W_1 - |\cos(2\pi z)\cos(2\pi R)|| &< 2^{-124.414}, \\ |W_2 - \sin(2\pi z)\sin(2\pi R)| &< 2^{-125.264}. \end{aligned}$$

Let $q = 0$ for $i_1 < 32$ and $q = 1$ otherwise, and let $R' = (-1)^{\sigma_r} R$. Before applying the half-turn factor $(-1)^b$, the signed approximation is $D = (-1)^q W_1 - (-1)^{\sigma_r} W_2$. For $(q, \sigma_r) = (0,0), (0,1), (1,0), (1,1)$ respectively this is $W_1 - W_2$, $W_1 + W_2$, $-(W_1 + W_2)$, $-(W_1 - W_2)$. Thus magnitudes are added exactly when $q + \sigma_r = 1$; otherwise the magnitude of the difference is used and its sign is adjusted according to the comparison of w_1 and w_2. The final sign also includes the half-turn bit $b = 1$ if $k \geq 2^{12}$, $b = 0$ otherwise. Define $d = 1$ if $q = \sigma_r$ and $w_1 \leq w_2$, $d = 0$ otherwise. Line 11 of Algorithm AccurateLarge supplies the quadrant bit, so $\sigma = (b + q + d) \bmod 2$ and $(-1)^\sigma C = (-1)^b D$. In all cases, the total error is bounded as follows:

$$|D - \cos(2\pi(z + R'))| < 2^{-124.414} + 2^{-125.264} < 2^{-123.777}.$$

Combined with the bound from Eq. (5), we obtain, taking into account the sign bit σ:

$$|(-1)^\sigma C - \cos(x)| < 2^{-123.777} + 2^{-125.347} < 2^{-123.358}. \quad (6)$$

We have to check that the computation of $w_1 + w_2$ does not overflow at line 12, since then the result would be near 0 instead of near 1 (up to sign). We know from [4] that the distance from a binary64 number to a multiple of $\pi/2$ is larger than $2^{-60.888}$, where the minimum distance is attained at $x =$ `0x1.6ac5b262ca1ffp+849`. Hence, the smallest distance from a binary64 number to a multiple of π is also larger than $2^{-60.888}$:

$$|\cos x| \leq |\cos(2^{-60.888})| \leq 1 - 2^{-122.78}.$$

Since $W_1 + W_2$ approximates $|\cos(2\pi(z + R'))|$ with error bounded by $2^{-123.777}$, and $|(-1)^b \cos(2\pi(z+R')) - \cos(x)| < 2^{-125.347}$ from Eq. (5), we obtain:

$$W_1 + W_2 < 1 - 2^{-122.78} + 2^{-123.777} + 2^{-125.347} < 1.$$

Now assume $|\cos x| \geq 2^{-27}$ and $y = \cos x$ has less than 43 identical bits after the round bit. Then if ζ is a rounding boundary, $|y - \zeta| \geq 2^{-44}\mathrm{ulp}(y) \geq 2^{-123}$, since $|y| \geq 2^{-27}$ implies $\mathrm{ulp}(y) \geq 2^{-79}$. Thus Eq. (6) shows that $(-1)^\sigma C$ is closer to $\cos x$ than any rounding boundary, thus rounding $(-1)^\sigma C$ yields the correct rounding of $\cos x$.

All inputs such that $\cos x$ has 43 or more identical bits after the round bit were computed in [2], and are included in the `cos.wc` file. The CORE-MATH test suite checked they are correctly rounded, for all rounding modes, and with or without FMA contraction.

It remains to prove correct rounding when $|\cos x| < 2^{-27}$. In this case, necessarily $k = 2^{11}$ or $k = 2^{12} + 2^{11}$ in Algorithm ReduceAccLarge, i.e., x is near an odd multiple of $\pi/2$, since otherwise from Eq. (4), the circular distance from $k/2^{13} + (-1)^{\sigma_r} R$ to either $1/4$ or $3/4$ is at least 2^{-14}. The corresponding distance for $x/(2\pi) \bmod 1$ is therefore at least $2^{-14} - 2^{-127.999}$, which from Eq. (5) is incompatible with $|\cos x| < 2^{-27}$. If $2^{-27} \leq x < 1$, this cannot happen, thus necessarily we have $x > 1$. Using Lefèvre's algorithm, in each binade $[2^{e-1}, 2^e)$, for $1 \leq e \leq 1024$, we generated all binary64 inputs such that $|\cos x| < 2^{-27}$, which implies $|x \bmod (\pi/2)| < |\mathrm{acos}(2^{-27}) - \pi/2| < 2^{-26.999}$, and kept those for which $\cos x$ is at distance less than 2^{-106} from a rounding boundary (the bound 2^{-123} would have been enough, but we wanted to find a non-empty set). We found 16778 such inputs, added them to `cos.wc`, and checked they are correctly rounded for all rounding modes, with and without FMA contraction. For example, in the binade $[2^{1023}, 2^{1024})$, we found 10 such inputs, from `0x1.5140b9e068a61p+1023` to `0x1.ffa528720daedp+1023`. Among the supplied additional inputs with $|\cos x| < 2^{-27}$, the smallest measured boundary distance occurs at $x =$ `0x1.6477e472c5074p+715` and is about $2^{-120.390}$. ■

V. Efficiency

Figure 1 compares the efficiency of several implementations of the binary64 cosine function for $1 \leq x \leq 2^{100}$. We see all implementations have two regimes: one for small inputs with an ad-hoc argument reduction, and one for large inputs, with Payne-Hanek's argument reduction. The threshold between the two regimes differs between implementations: around 2^{15} for LLVM libc, around 2^{19} for the Intel library, around 2^{21} for AMD libm (with a spike around 2^{14}), around 2^{25} for GNU libc, and around 2^{31} for CORE-MATH. Note that only CORE-MATH and LLVM libc are correctly rounded; they also have the fastest Payne-Hanek reductions.

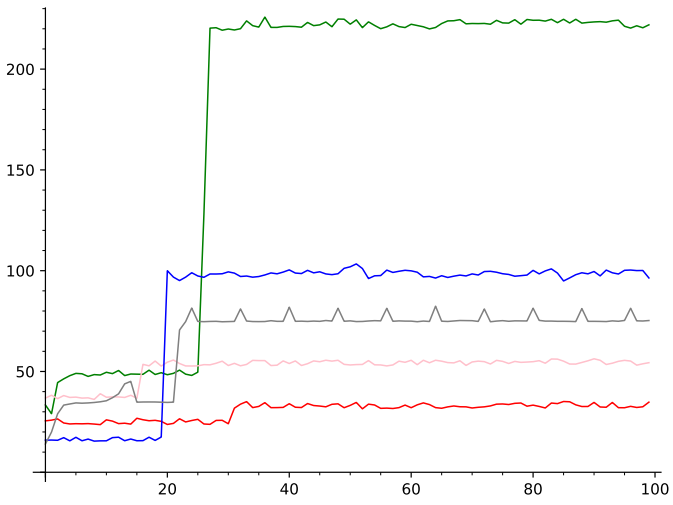

Fig. 1. Reciprocal throughput in cycles of several implementations of the binary64 cosine function around 2^e for $0 \leq e \leq 100$, on an Intel Xeon Silver 4214. The red curve is CORE-MATH compiled with gcc 16.2.0, the pink curve is LLVM libc (dc3e262) compiled with clang 21.1.8, the grey curve is AMD libm 5.3, the blue curve is the Intel math library 2026.1.0, and the green curve is GNU libc 2.43 from Debian.

Acknowledgements. Eric Hennenhoefer (using AI) helped to improve that document.

REFERENCES

[1] 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).

[2] LEFÈVRE, V., LY, T., AND ZIMMERMANN, P. Computing hard-to-round cases of sin, cos, tan in double precision. In *ARITH 2026 - 33rd IEEE International Symposium on Computer Arithmetic* (Fulda, Germany, June 2026).

[3] MELQUIOND, G., AND ZIMMERMANN, P. The CORE-MATH sin is correctly rounded, 2026. https://core-math.gitlabpages.inria.fr/sin.pdf.

[4] 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.