

Correctly rounded multiplication by arbitrary precision constants

N. Brisebarre, F. de Dinechin and J.-M. Muller
Arénaire, LIP, É.N.S. Lyon

Séminaire Algorithms
2/2/2009

Floating Point (FP) Arithmetic

Given

$$\left\{ \begin{array}{ll} \text{a radix} & \beta \geq 2, \\ \text{a precision} & n \geq 1, \\ \text{a set of exponents} & E_{\min} \cdots E_{\max}. \end{array} \right.$$

A finite FP number x is represented by 2 integers :

- integer mantissa : M , $\beta^{n-1} \leq |M| \leq \beta^n - 1$;
- exponent E , $E_{\min} \leq e \leq E_{\max}$

such that

$$x = \frac{M}{\beta^{n-1}} \times \beta^e.$$

We call real mantissa, or mantissa of x the number $m = M \times \beta^{1-n}$, such that $x = m \times \beta^e$.

We assume binary FP arithmetic (that is to say $\beta = 2$.)

IEEE precisions

<http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html>

	precision	minimal exponent	maximal exponent
single	24	-126	127
double	53	-1022	1023
extended double	64	-16382	16383
quadruple	113	-16382	16383

Two approaches

- Classical approach : the FP operations are approximations to real operations \Rightarrow successive upper bounds or probabilistic approaches to rounding errors.
 - ▶ Interval arithmetic : <http://www.cs.utep.edu/interval-comp/> ;
 - ▶ “stochastic” arithmetic : <http://www-anp.lip6.fr/cadna/>
- Correct rounding : FP arithmetic is a structure which we can study in itself, and not only an approximation of the field \mathbb{R} . We can prove interesting properties and develop useful algorithms.

Correct rounding

Definition 1. [Correct rounding] *The user defines an active rounding mode among : rounding to nearest (default mode), rounding towards $+\infty$, rounding towards $-\infty$, rounding towards zero.*

An operation whose entries are FP numbers should return the same result as if we had first computed the exact result, and then rounded it using the active rounding mode.

In the sequel, we use rounding to nearest : we round towards the closest FP number, if there are 2 such numbers, we choose the one whose integer mantissa is even. Let $x \in \mathbb{R}$, the rounding to nearest of x will be denoted $\circ(x)$.

IEEE 754 is the standard that rules correct rounding in FP binary arithmetic.

Rounding to nearest : an example

We consider $x = \pi$, IEEE double precision and rounding to nearest.

We want to find the number of the form $\frac{M}{2^{52}}2^E$, where $M \in \mathbb{Z}$, $2^{52} \leq |M| \leq 2^{53} - 1$, $-1022 \leq E \leq 1023$, that is as close as possible to π

First, we determine E : we have $E = \lfloor \log(\pi) / \log(2) \rfloor = 1$.

Then we compute M . We have $\pi 2^{52-E} = \pi 2^{51} = 7074237752028440.27\dots$.
Or, M is the closest integer to $\pi 2^{52-E}$. Therefore, $M = 7074237752028440$.

Hence, we have, in IEEE double precision,

$$\circ(\pi) = \frac{7074237752028440}{2^{51}} = \frac{884279719003555}{281474976710656}.$$

ULP : Unit in the Last Place

- Roughly speaking, $\text{ulp}(x) =$ distance between two consecutive FP numbers around x ;
- Ambiguous at the neighborhood of powers of radix 2 \Rightarrow several slightly different definitions exist ;
- Which one should be chosen ? The one that preserves in those “ambiguous zones” some properties that are true elsewhere.

ULP : Unit in the Last Place

There are 3 most commonly used definitions : one by W. Kahan, one by J. Harrison and another by D. Goldberg.

In this talk, we use the last one.

Definition 2. [Goldberg] *If $x \in [2^E, 2^{E+1})$ then $ulp(x) = 2^{E-n+1}$.*

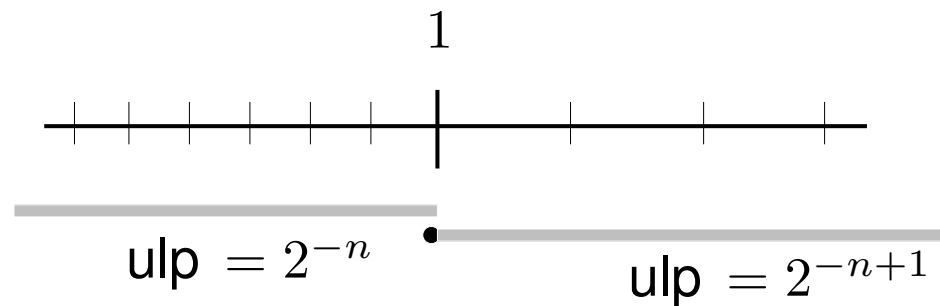


FIG. 1: *Values of $ulp(x)$ around 1, assuming radix 2 and precision n .*

ULP : Unit in the Last Place - examples

We assume IEEE double precision, i.e. $n = 53$.

- $\text{ulp}(\sqrt{2}) = ?$. Let $E = \lfloor \log(\sqrt{2}) / \log(2) \rfloor = 0$.

We have $\sqrt{2} \in [2^E, 2^{E+1})$ hence $\text{ulp}(\sqrt{2}) = 2^{E-52} = 2^{-52}$.

- $\text{ulp}(\pi) = ?$. Let $E = \lfloor \log(\pi) / \log(2) \rfloor = 1$.

We have $\pi \in [2^E, 2^{E+1})$ hence $\text{ulp}(\pi) = 2^{E-52} = 2^{-51}$.

- $\text{ulp}(1/1000) = ?$. Let $E = \lfloor \log(1/1000) / \log(2) \rfloor = -10$.

We have $1/1000 \in [2^E, 2^{E+1})$ hence $\text{ulp}(1/1000) = 2^{E-52} = 2^{-62}$.

ULP and Correct Rounding

Assume that the binary FP number X approximates the real number x .
Then,

- $|X - x| < \frac{1}{2} \text{ulp}(x) \Rightarrow X = \circ(x)$,
- $X = \circ(x) \Rightarrow |X - x| \leq \frac{1}{2} \text{ulp}(x)$,
- $X = \circ(x) \Rightarrow |X - x| \leq \frac{1}{2} \text{ulp}(X)$,
- $|X - x| < \frac{1}{2} \text{ulp}(X)$ implies $X = \circ(x)$ (except a few well known cases).

Multiplications by constants

Many numerical algorithms : multiplications by constants that are not exactly representable in floating-point (FP) arithmetic.

Typical constants that are used : π , $1/\pi$, $\ln(2)$, e , $B_k/k!$ (Euler-McLaurin summation), $\cos(k\pi/N)$ and $\sin(k\pi/N)$ (Fast Fourier Transforms). Some numerical integration formulas such as :

$$\int_{x_0}^{x_1} f(x)dx \approx h \left(\frac{55}{24}f(x_1) - \frac{59}{24}f(x_2) + \frac{37}{24}f(x_3) - \frac{9}{24}f(x_4) \right)$$

also naturally involve multiplications by constants.

Correctly rounded Multiplications by constants

For approximating Cx , where C is an infinite-precision constant and x is a FP number, desirable result = $\circ(Cx)$, where $\circ(u)$ is u rounded to the nearest FP number.

Our goal : We want to compute at low cost $\circ(Cx)$ for all input FP numbers x (provided no overflow or underflow occur).

Naive idea : let C_h be the FP number that is closest to C , we actually compute $\circ(C_h x)$. The obtained result is frequently different from $\circ(Cx)$.

Some statistics

Let n = number of mantissa bits of the binary FP format.

Comparison of $\circ(C_h x)$ and $\circ(Cx)$ for all possible values of the mantissa of x .

n	Proportion of correctly rounded results
4	0.62500
5	0.93750
6	0.78125
7	0.59375
...	...
16	0.86765
17	0.73558
...	...
24	0.66805

TAB. 1: *Proportion of input values x for which $\circ(C_h x) = \circ(Cx)$ for $C = \pi$ and various values of the number n of mantissa bits.*

Correctly rounded Multiplications by constants

Our goal – at least for some constants and some FP formats – is to return $\circ(Cx)$ for all input FP numbers x (provided no overflow or underflow occur), and at a low cost.

To do that, we will use *fused multiply and add* (`fma`) instructions.

`fma` : computes correct rounding of $ab + c$ where a, b and c are FP numbers. Available on Intel Itanium and IBM PowerPC. It makes polynomial evaluation (Horner) or dot products faster and, in general more accurate than with \pm and \times .

The algorithm

- We want Cx with correct rounding (assuming rounding to nearest even).
- C is not an FP number.
- We assume that a `fma` instruction is available. Operands stored in a binary FP format with n -bit mantissas.
- We assume that the two following FP numbers are pre-computed :

$$\begin{cases} C_h & = & \circ(C), \\ C_\ell & = & \circ(C - C_h), \end{cases}$$

where $\circ(t)$ stands for t rounded to the nearest FP number.

Algorithm 1. (*Multiplication by C with a multiplication and a `fma`*). From x , compute

$$\begin{cases} u_1 & = & \circ(C_\ell x), \\ u_2 & = & \circ(C_h x + u_1). \end{cases}$$

The result to be returned is u_2 .

Algorithm. (*Multiplication by C with a multiplication and a fma*). From x , compute

$$\begin{cases} u_1 &= \circ(C_\ell x), \\ u_2 &= \circ(C_h x + u_1). \end{cases}$$

The result to be returned is u_2 .

Without l.o.g., we assume that $1 < x < 2$ and $1 < C < 2$, that C is not exactly representable, and that $C - C_h$ is not a power of 2.

Warning ! There exist C and x s.t. $u_2 \neq \circ(Cx)$.

We give 3 methods for checking if for all FP number x , $u_2 = \circ(Cx)$.

Algorithm. (*Multiplication by C with a multiplication and a fma*). From x , compute

$$\begin{cases} u_1 &= \circ(C_\ell x), \\ u_2 &= \circ(C_h x + u_1). \end{cases}$$

The result to be returned is u_2 .

3 methods for checking if $\forall x, u_2 = \circ(Cx)$.

Methods 1 and 2 are simple but do not always give a complete answer :

- they either certify that our algorithm always returns a correctly rounded result,
- or give a “bad case”, i.e. an FP number x s.t. $u_2 \neq \circ(Cx)$.

Method 3 is a bit more complicated but gives a complete answer :

- it gives all “bad cases”,
- or certify that there are none, i.e. that our algorithm always gives the correct result.

Analyzing the algorithm

We will use the following property, that bounds the maximum possible distance between u_2 and Cx in the algorithm.

Property 3.

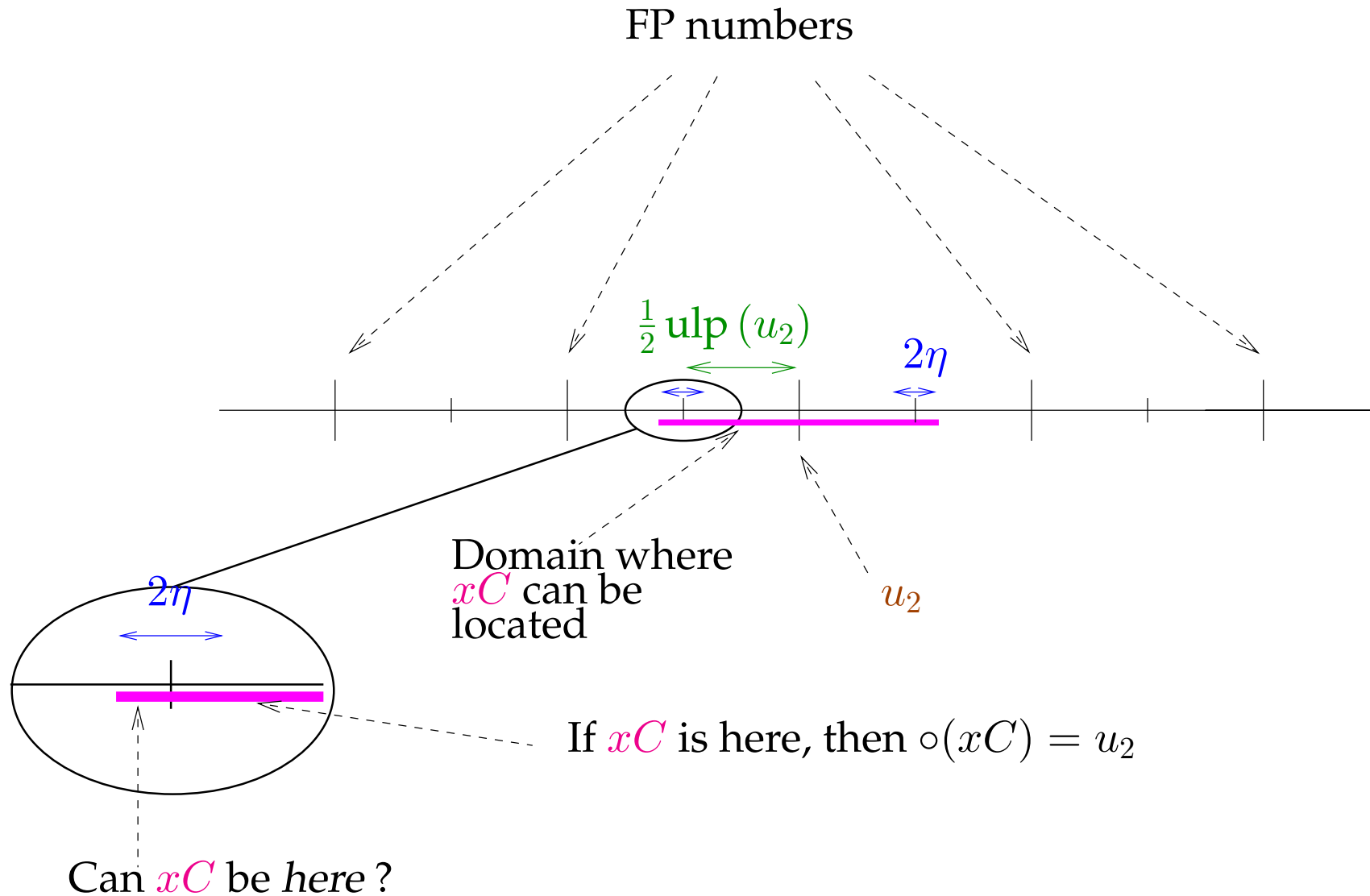
For all FP number x , we have

$$|u_2 - Cx| < \frac{1}{2} \text{ulp}(u_2) + 2 \text{ulp}(C_\ell).$$

[Remember that $C_h = \circ(C)$, $C_\ell = \circ(C - C_h)$, $u_1 = \circ(C_\ell x)$, $u_2 = \circ(C_h x + u_1)$.]

Analyzing the algorithm

Recall : we have $|u_2 - Cx| < 1/2 \text{ulp}(u_2) + \eta$ with $\eta := 2 \text{ulp}(C_\ell)$.



Analyzing the algorithm

Remark . We know that xC is within $1/2 \text{ulp}(u_2) + 2 \text{ulp}(C_\ell)$ from the FP number u_2 . If we prove that xC cannot be at a distance $\leq 2 \text{ulp}(C_\ell)$ from the middle of two consecutive FP numbers, then u_2 will be the FP number that is closest to xC .

A reminder on continued fractions

Let $\beta \in \mathbb{R}$. From β , two sequences (a_i) and (r_i) defined by :

$$\begin{cases} r_0 & = \beta, \\ a_i & = [r_i], \\ r_{i+1} & = 1/(r_i - a_i). \end{cases}$$

If $\beta \notin \mathbb{Q}$, these sequences are defined $\forall i$, and the rational number

$$\frac{p_i}{q_i} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{\dots + \frac{1}{a_i}}}}}$$

is the i th convergent to β . If $\beta \in \mathbb{Q}$, these sequences terminate for some i , and $p_i/q_i = \beta$ exactly.

We will use the following two results :

Theorem 4. Let $(p_j/q_j)_{j \geq 1}$ be the convergents of β . For any (p, q) , with $0 \leq q < q_{n+1}$, we have

$$|p - \beta q| \geq |p_n - \beta q_n|.$$

Theorem 5. Let p, q be nonzero integers, with $\gcd(p, q) = 1$. If

$$\left| \frac{p}{q} - \beta \right| < \frac{1}{2q^2}$$

then p/q is a convergent of β .

Method 3

Assume $x > x_{\text{cut}} := 2/C$ (the case $x < x_{\text{cut}} = 2/C$ is similar).

Let $X_{\text{cut}} := \lfloor 2^{n-1} x_{\text{cut}} \rfloor$.

We recall the notations : $C_h = \circ(C)$, $C_\ell = \circ(C - C_h)$, $u_1 = \circ(C_\ell x)$,
 $u_2 = \circ(C_h x + u_1)$.

We want to determine the integers X , $X_{\text{cut}} \leq X \leq 2^n - 1$ that satisfy

$$\left| u_2 - C \frac{X}{2^{n-1}} \right| < \frac{1}{2} \text{ulp}(u_2) + 2 \text{ulp}(C_\ell),$$

or equivalently, the integers X , $X_{\text{cut}} \leq X \leq 2^n - 1$ s.t. there exists an integer A with

$$\left| C \frac{X}{2^{n-1}} - \frac{2A + 1}{2^{n-1}} \right| \leq 2 \text{ulp}(C_\ell).$$

Once we know the X candidate, we compute u_2 and $\circ(Cx)$ to check if they coincide or not.

Method 3

We search for the $x = X/2^{n-1}$, $X_{\text{cut}} \leq X \leq 2^n - 1$ s.t. there exists an integer A with

$$\left| C \frac{X}{2^{n-1}} - \frac{2A+1}{2^{n-1}} \right| \leq 2 \text{ulp}(C_\ell).$$

We know that $\text{ulp}(C_\ell) \leq 2^{-2n}$.

We distinguish the cases $\text{ulp}(C_\ell) \leq 2^{-2n-1}$ and $\text{ulp}(C_\ell) = 2^{-2n}$.

Method 3

First, we assume $\text{ulp}(C_\ell) \leq 2^{-2n-1}$.

In that case, the integers $x = X/2^{n-1}$, $X_{\text{cut}} \leq X \leq 2^n - 1$ satisfy

$$\left| 2C - \frac{2A + 1}{X} \right| < \frac{1}{2X^2} :$$

$(2A + 1)/X$ is a convergent of $2C$ from Theorem 5. It suffices then to check the convergents of $2C$ of denominator less or equal to $2^n - 1$.

Method 3

Now, assume $\text{ulp}(C_\ell) = 2^{-2n}$.

Careful computations lead to the following problem : determine the X , $X_{\text{cut}} \leq X \leq 2^n - 1$ s.t.

$$\{X(C_h + C_\ell) + \frac{1}{2^{n+1}}\} \leq \frac{1}{2^n},$$

where $\{y\}$ is the fractional part of y : $\{y\} = y - \lfloor y \rfloor$. Ex. : $\{4/3\} = 4/3 - \lfloor 4/3 \rfloor = 4/3 - 1 = 1/3$, $\{-11/6\} = -11/6 - \lfloor -11/6 \rfloor = -11/6 - (-2) = 1/6$.

We use an efficient algorithm due to V. Lefèvre to determine all the integers X , $X_{\text{cut}} \leq X \leq 2^n - 1$ solution of this inequality.

Two other methods

- Methods 1 and 2 are simpler : they each give a criterion, easy to check, that guarantee that the algorithm always returns a correctly rounded result. They also may give some values of x such that $u_2 \neq \circ(Cx)$.
- Method 1 uses Theorem 4, Method 2 uses Theorem 5. We may need the examination of all convergents to $2C$ or C .

Two examples

Method 1 allows to prove

Theorem 6. [Correctly rounded multiplication by π] *The algorithm always returns a correctly rounded result in double precision with $C = 2^j \pi$, where j is any integer, provided no under/overflow occur.*

With $\ln(2)$, needs more work (uses Method 2 and examination of all convergents)

Theorem 7. [Correctly rounded multiplication by $\ln(2)$] *The algorithm always returns a correctly rounded result in double precision with $C = 2^j \ln(2)$, where j is any integer, provided no under/overflow occur.*

Example 3 : multiplication by $1/\pi$ in double precision

Consider the case $C = 4/\pi$ and $n = 53$, and assume we use Method 1. We find a counterexample : $x = 6081371451248382 \times 2^{\pm k}$.

Method 3 certifies that $x = 6081371451248382 \times 2^{\pm k}$ are the *only* FP values for which our algorithm fails.

Implementation

We have written Maple programs that implement Methods 1, 2 and 3, and a GP/PARI program that implements Method 3.

These programs can be downloaded from the url

`http://perso.ens-lyon.fr/jean-michel.muller/MultConstant.html`

Some results

C	n	Method 1	Method 2	Method 3
π	8	Does not work for 226	Does not work for 226	AW (c) unless $X =$ 226
π	24	unable	unable	AW
π	53	AW	unable	AW
π	64	unable	AW	AW (c)
π	113	AW	AW	AW (c)

TAB. 2: *Some results obtained using Methods 1, 2 and 3. The results given for constant C hold for all values $2^{\pm j}C$. “AW” means “always works” and “unable” means “the method is unable to conclude”. For Method 3, “(c)” means that we have needed to check the convergents.*

C	n	Method 1	Method 2	Method 3
$1/\pi$	24	unable	unable	AW
$1/\pi$	53	Does not work for 6081371451248382	unable	AW unless $X =$ 6081371451248382
$1/\pi$	64	AW	AW	AW (c)
$1/\pi$	113	unable	unable	AW
$\ln 2$	24	AW	AW	AW (c)
$\ln 2$	53	AW	unable	AW (c)
$\ln 2$	64	AW	unable	AW (c)
$\ln 2$	113	AW	AW	AW (c)

TAB. 3: *Some results obtained using Methods 1, 2 and 3. The results given for constant C hold for all values $2^{\pm j}C$. “AW” means “always works” and “unable” means “the method is unable to conclude”. For Method 3, “(c)” means that we have needed to check the convergents.*

C	n	Method 1	Method 2	Method 3
$\frac{1}{\ln 2}$	24	unable	AW	AW (c)
$\frac{1}{\ln 2}$	53	AW	AW	AW (c)
$\frac{1}{\ln 2}$	64	unable	unable	AW
$\frac{1}{\ln 2}$	113	unable	unable	AW
$\ln 10$	24	unable	AW	AW (c)
$\ln 10$	53	unable	unable	AW
$\ln 10$	64	unable	AW	AW (c)
$\ln 10$	113	AW	AW	AW (c)

TAB. 4: *Some results obtained using Methods 1, 2 and 3. The results given for constant C hold for all values $2^{\pm j}C$. “AW” means “always works” and “unable” means “the method is unable to conclude”. For Method 3, “(c)” means that we have needed to check the convergents.*

C	n	Method 1	Method 2	Method 3
$\frac{2^j}{\ln 10}$	24	unable	unable	AW
$\frac{2^j}{\ln 10}$	53	unable	AW	AW (c)
$\frac{2^j}{\ln 10}$	64	unable	AW	AW (c)
$\frac{2^j}{\ln 10}$	113	unable	unable	AW
$\cos \frac{\pi}{8}$	24	unable	unable	AW
$\cos \frac{\pi}{8}$	53	AW	AW	AW (c)
$\cos \frac{\pi}{8}$	64	AW	unable	AW
$\cos \frac{\pi}{8}$	113	unable	AW	AW (c)

TAB. 5: *Some results obtained using Methods 1, 2 and 3. The results given for constant C hold for all values $2^{\pm j}C$. “AW” means “always works” and “unable” means “the method is unable to conclude”. For Method 3, “(c)” means that we have needed to check the convergents.*

Assuming intermediate calculations in a larger format

Frequently, intermediate calculations may be performed in an internal format significantly larger than the “target” format.

On Intel processors, intermediate calculations can be performed in double-extended precision (with 64-bit mantissas), with final result being converted to double precision.

This is done at no cost, since the only operators actually implemented are double-extended precision operators.

Hence, it is important to see what changes in the Algorithm and its properties in this case.

We assume a “target” n -bit format.

C_h and C_ℓ are computed in a larger format, with $n + g$ bits of mantissa.

The Algorithm is performed in that $n + g$ -bit format, before its final result is rounded to the nearest number with n -bit mantissa.

We still assume an `fma` instruction is available.

$\circ_k(x)$: x rounded to the nearest FP number with k bits of mantissa.

Let

$$\begin{cases} C_h & = & \circ_{n+g}(C), \\ C_\ell & = & \circ_{n+g}(C - C_h). \end{cases}$$

Algorithm 2. *(Multiplication by C using an intermediate format with $n + g$ bits of mantissa). From x , compute*

$$\begin{cases} u_1 & = & \circ_{n+g}(C_\ell x), \\ u_2 & = & \circ_n(C_h x + u_1). \end{cases}$$

The result to be returned is u_2 .

Results

Case $n = 53$ and $g = 11$ (corresponds to the very useful case of double precision as the “target” format and double-extended precision as the “internal” format)

Algorithm 2 always returns a correctly rounded product when used with C equal to any power of 2 times π , $1/\pi$, $\ln(2)$, $1/\ln(2)$, $\ln(10)$, $1/\ln(10)$ and $\cos(\pi/8)$.

And yet, there is still an ad hoc method for finding counterexamples for these values of n and g .

Conclusion

The four methods we have proposed allow to check whether correctly rounded multiplication by an “infinite precision” constant C is feasible at a low cost (one multiplication and one `fma`).