Correctly rounded multiplication by arbitrary precision constants

Nicolas Brisebarre[1] Jean-Michel Muller[1]
[1]Laboratoire LARAL, Université Jean Monnet (Saint-Étienne)
and Laboratoire LIP (CNRS/ENS Lyon/INRIA/Univ. Lyon 1),
[2]CNRS, Laboratoire LIP,
Projet Arénaire, 46 allée d'Italie, 69364 Lyon Cedex 07,
FRANCE
Nicolas.Brisebarre@ens-lyon.fr, Jean-Michel.Muller@ens-lyon.fr

***Abstract*— We introduce an algorithm for multiplying a floating-point number x by a constant C that is not exactly representable in floating-point arithmetic. Our algorithm uses a multiplication and a fused multiply accumulate instruction. We give methods for checking whether, for a given value of C and a given floating-point format, our algorithm returns a correctly rounded result for any x. When it does not, our methods give the values x for which the multiplication is not correctly rounded.**

INTRODUCTION

Many numerical algorithms require multiplications by constants that are not exactly representable in floating-point (FP) arithmetic. Typical constants that are used [1], [4] are 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 [4], page 133:

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

also naturally involve multiplications by constants.

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

In practice one usually defines a constant C_h, equal to the FP number that is closest to C, and actually computes $C_h x$ (i.e., what is returned is $\circ(C_h x)$). The obtained result is frequently different from $\circ(Cx)$ (see Section I for some statistics).

Our goal here is to be able – at least for some constants and some FP formats – to return $\circ(Cx)$ for all input FP numbers x (provided no overflow or underflow occur), and at a low cost (i.e., using a very few arithmetic operations only). To do that, we will used *fused multiply accumulate* instructions.

The fused multiply accumulate instruction (fused-mac for short) is available on some current processors such as the IBM Power PC or the Intel/HP Itanium. That instruction evaluates an expression $ax + b$ with one final rounding error only. This makes it possible to perform correctly rounded division using Newton-Raphson division [9], [3], [8]. Also, this makes evaluation of scalar products and polynomials faster and, generally, more accurate than with conventional (addition and multiplication) floating-point operations.

I. SOME STATISTICS

Let n be the number of mantissa bits of the considered floating-point format (usual values of n are 24, 53, 64, 113). For small values of n, it is possible to compute $\circ(C_h x)$ and $\circ(Cx)$ for all possible values of the mantissa of x. The obtained results are given in Table I, for $C = \pi$. They show that, at least for some values of n, the "naive" method that consists in computing $\circ(C_h x)$ returns an incorrectly rounded result quite often (in around 41% of the cases for $n = 7$).

II. THE ALGORITHM

We want to compute Cx with correct rounding (assuming rounding to nearest even), where C is a constant (i.e., C is known at compile time). C is not an FP number (otherwise the problem would be straightforward). We assume that a fused-mac instruction is available. We assume that the operands are stored in a binary FP format with n-bit mantissas.

We assume that the two following FP numbers are pre-computed:

$$\begin{cases} C_h &= \circ(C), \\ C_\ell &= \circ(C - C_h), \end{cases} \tag{1}$$

n	Proportion of correctly rounded results
4	0.62500
5	0.93750
6	0.78125
7	0.59375
8	0.96875
...	...
16	0.86765
17	0.73558
...	...
24	0.66805

TABLE I

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.

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

In the sequel of the paper, we will analyze the behavior of the following algorithm. We aim at being able to know for which values of C and n it will return a correctly rounded result for any x. When it does not, we wish to know for which values of x it does not.

Algorithm 1: (Multiplication by C with a multiplication and a fused-mac). From x, compute

$$\left\{ \begin{array}{lcl} u_1 & = & \circ(C_\ell x), \\ u_2 & = & \circ(C_h x + u_1). \end{array} \right. \tag{2}$$

The result to be returned is u_2.

$\square$

When C is the exact reciprocal of a FP number, this algorithm coincides with an algorithm for division by a constant given in [2].

Obviously (provided no overflow/underflow occur) if Algorithm 1 gives a correct result with a given constant C and a given input variable x, it will work as well with a constant $2^p C$ and an input variable $2^q x$, where p and q are integers. Also, if x is a power of 2 or if C is exactly representable (i.e., $C_\ell = 0$), or if $C - C_h$ is a power of 2 (so that u_1 is exactly $(C - C_h)x$), it is straightforward to show that $u_2 = \circ(Cx)$. Hence, *without loss of generality, we assume in the following 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.

In Section IV, we give three methods. The first two ones either certify that Algorithm 1 always returns a correctly rounded result, or give a "bad case" (i.e., a number x for which $u_2 \neq \circ(Cx)$), or are not able to conclude. The third one is able to return all "bad cases", or certify that there are none. These methods use the following property, that bound the maximum possible distance between u_2 and Cx in Algorithm 1.

Property 1:
Define $x_{\text{cut}} = 2/C$ and

$$\epsilon_1 = |C - (C_h + C_\ell)| \tag{3}$$

- If $x < x_{\text{cut}}$ then $|u_2 - Cx| < 1/2\,\text{ulp}\,(u_2) + \alpha$,
- If $x \geq x_{\text{cut}}$ then $|u_2 - Cx| < 1/2\,\text{ulp}\,(u_2) + \alpha'$,

where

$$\left\{ \begin{array}{lcl} \alpha & = & \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}}) + \epsilon_1 x_{\text{cut}}, \\ \alpha' & = & \text{ulp}\,(C_\ell) + 2\epsilon_1. \end{array} \right.$$

$\square$

Proof.
From $1 < C < 2$ and $C_h = \circ(C)$, we deduce $|C - C_h| < 2^{-n}$, which gives (since $C - C_h$ is not a power of 2),

$$|\epsilon_1| \leq \frac{1}{2}\,\text{ulp}\,(C_\ell) \leq 2^{-2n-1}.$$

Now, we have,

$$\begin{array}{rcl} |u_2 - Cx| & \leq & |u_2 - (C_h x + u_1)| \\ & & + \,|(C_h x + u_1) - (C_h x + C_\ell x)| \\ & & + \,|(C_h + C_\ell)x - Cx| \\ & \leq & \frac{1}{2}\,\text{ulp}\,(u_2) + |u_1 - C_\ell x| + \epsilon_1 |x| \\ & \leq & \frac{1}{2}\,\text{ulp}\,(u_2) + \frac{1}{2}\,\text{ulp}\,(C_\ell x) + \epsilon_1 |x|. \end{array} \tag{4}$$

$\square$

If $|u_2 - Cx|$ is less than $1/2\,\text{ulp}\,(u_2)$, then u_2 is the FP number that is closest to xC. Hence our problem is to know if Cx can be at a distance larger than or equal to $\frac{1}{2}\,\text{ulp}\,(u_2)$ from u_2. From (4), this would imply that Cx would be at a distance less than $\frac{1}{2}\,\text{ulp}\,(C_\ell x) + \epsilon_1 |x| < 2^{-2n+1}$ from the middle of two consecutive FP numbers (see Figure 1).

If $x < x_{\text{cut}}$ then $xC < 2$, therefore the middle of two consecutive FP numbers around xC is of the form $A/2^n$, where A is an odd integer between $2^n + 1$ and $2^{n+1} - 1$. If $x \geq x_{\text{cut}}$, then the middle of two consecutive FP numbers around xC is of the form $A/2^{n-1}$. For the sake of clarity of the proofs we assume that x_{cut} is not an FP number (if x_{cut} is an FP number, it suffices to separately check Algorithm 1 with $x = x_{\text{cut}}$).

III. A reminder on continued fractions

We just recall here the elementary results that we need in the following, for the sake of completeness. For more

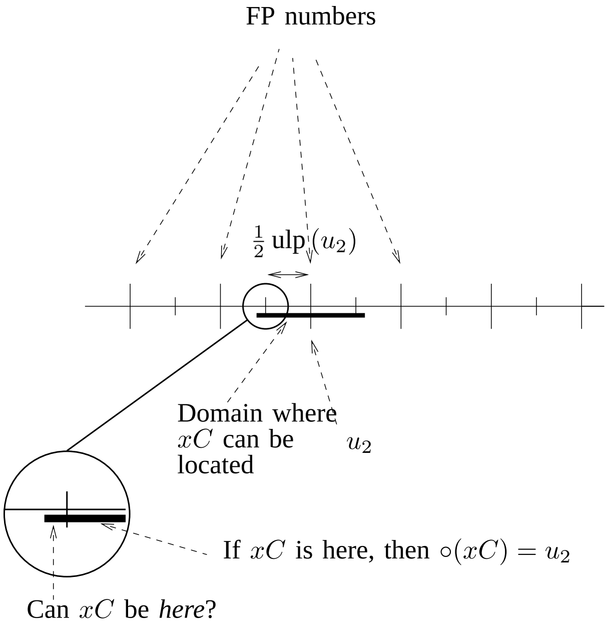

Fig. 1. *From (4), we know that xC is within $1/2\,ulp\,(u_2) + \alpha$ (or α') from the FP number u_2, where α is less than 2^{-2n+1}. If we are able to show that xC cannot be at a distance less than or equal to α (or α') from the middle of two consecutive floating-point numbers, then, necessarily, u_2 will be the FP number that is closest to xC.*

information on continued fractions, see [5], [11], [10], [6].

Let α be a real number. From α, consider the two sequences (a_i) and (r_i) defined by:

$$\left\{ \begin{array}{lcl} r_0 & = & \alpha, \\ a_i & = & \lfloor r_i \rfloor, \\ r_{i+1} & = & \dfrac{1}{r_i - a_i}. \end{array} \right. \tag{5}$$

If α is irrational, then these sequences are defined for any i (i.e., r_i is never equal to a_i), and the rational number

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

is called the ith *convergent* to α. If α is rational, then these sequences finish for some i, and $p_i/q_i = \alpha$ exactly. The p_is and the q_is can be deduced from the a_i using the following recurrences,

$$\begin{array}{lcl} p_0 & = & a_0, \\ p_1 & = & a_1 a_0 + 1, \\ q_0 & = & 1, \\ q_1 & = & a_1, \\ p_n & = & p_{n-1} a_n + p_{n-2}, \\ q_n & = & q_{n-1} a_n + q_{n-2}. \end{array}$$

The major interest of the continued fractions lies in the fact that p_i/q_i is the best rational approximation to α among all rational numbers of denominator less than or equal to q_i.

We will use the following two results [5]

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

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

□

Theorem 2: Let p, q be nonzero integers, with $\gcd(p,q) = 1$. If

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

then p/q is a convergent of α.

□

IV. Three methods for analyzing Algorithm 1

A. Method 1: use of Theorem 1

Define $X = 2^{n-1}x$ and $X_{\text{cut}} = \lfloor 2^{n-1} x_{\text{cut}} \rfloor$. X and X_{cut} are integers between $2^{n-1}+1$ and $2^n - 1$. We separate the cases $x < x_{\text{cut}}$ and $x > x_{\text{cut}}$.

1) If $x < x_{cut}$: we want to know if there is an integer A between $2^n + 1$ and $2^{n+1} - 1$ such that

$$\left| Cx - \frac{A}{2^n} \right| < \alpha \tag{6}$$

where α is defined in Property 1. (6) is equivalent to

$$|2CX - A| < 2^n \alpha \tag{7}$$

Define $(p_i/q_i)_{i\geq 1}$ as the convergents of $2C$. Let k be the smallest integer such that $q_{k+1} > X_{\text{cut}}$, and define $\delta = |p_k - 2Cq_k|$. Theorem 1 implies that for any $A, X \in \mathbb{Z}$, with $0 < X \leq X_{\text{cut}}$, $|2CX - A| \geq \delta$. Therefore

1) if $\delta \geq 2^n\alpha$ then $|Cx - A/2^n| < \alpha$ is impossible. *In that case, Algorithm 1 returns a correctly rounded result for any $x < x_{cut}$*;
2) if $\delta < 2^n\alpha$ then we try Algorithm 1 with $y = q_k 2^{-n+1}$. If the obtained result is not $\circ(yC)$, then

we know that Algorithm 1 fails for at least one value[1]. Otherwise, we cannot conclude.

2) If $x > x_{cut}$: we want to know if there is an integer A between $2^n + 1$ and $2^{n+1} - 1$ such that

$$\left|Cx - \frac{A}{2^{n-1}}\right| < \alpha' \tag{8}$$

where α' is defined in Property 1. (8) is equivalent to

$$|CX - A| < 2^{n-1}\alpha' \tag{9}$$

Define $(p'_i/q'_i)_{i\geq 1}$ as the convergents of C. Let k' be the smallest integer such that $q'_{k'+1} \geq 2^n$, and define $\delta' = |p'_{k'} - Cq'_{k'}|$. Theorem 1 implies that for any $A, X \in \mathbb{Z}$, with $X_{\text{cut}} \leq X < 2^n$, $|CX - A| \geq \delta'$. Therefore

1) if $\delta' \geq 2^{n-1}\alpha'$ then $|Cx - A/2^{n-1}| < \alpha'$ is impossible. *In that case, Algorithm 1 returns a correctly rounded result for any $x > x_{cut}$;*
2) if $\delta' < 2^{n-1}\alpha'$ then we try Algorithm 1 with $y = q'_{k'}2^{-n+1}$. If the obtained result is not $\circ(yC)$, then *we know that Algorithm 1 fails for at least one value*. Otherwise, we cannot conclude.

B. Method 2: use of Theorem 2

Again, we use $X = 2^{n-1}x$ and $X_{\text{cut}} = \lfloor 2^{n-1}x_{\text{cut}} \rfloor$, and we separate the cases $x < x_{\text{cut}}$ and $x > x_{\text{cut}}$.

1) If $x > x_{cut}$: if

$$\left|Cx - \frac{A}{2^{n-1}}\right| < \epsilon_1 x + \frac{1}{2}\,\text{ulp}\,(C_\ell x)$$

then,

$$\left|C - \frac{A}{X}\right| < \epsilon_1 + \frac{2^{n-2}}{X}\,\text{ulp}\,(C_\ell x). \tag{10}$$

Now, if

$$2^{2n+1}\epsilon_1 + 2^{2n-1}\,\text{ulp}\,(2C_\ell) \leq 1, \tag{11}$$

then for any $X < 2^n$ (i.e., $x < 2$),

$$\epsilon_1 + \frac{2^{n-2}}{X}\,\text{ulp}\,(C_\ell x) < \frac{1}{2X^2}.$$

Hence, if (11) is satisfied, then (10) implies (from Theorem 2) that A/X is a convergent of C. This means that if (11) is satisfied, to find the possible bad cases for Algorithm 1 it suffices to examine the convergents of C of denominator less than 2^n. We can quickly eliminate most of them. A given convergent p/q (with $\gcd(p,q) = 1$) is a candidate for generating a value X for which Algorithm 1 does not work if there exist $X = mq$ and $A = mp$ such that

$$\begin{cases} X_{\text{cut}} < X \leq 2^n - 1, \\ 2^n + 1 \leq A \leq 2^{n+1} - 1, \\ \left|\frac{CX}{2^{n-1}} - \frac{A}{2^{n-1}}\right| < \epsilon_1 \frac{X}{2^{n-1}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x). \end{cases}$$

This would mean

$$\left|C\frac{mq}{2^{n-1}} - \frac{mp}{2^{n-1}}\right| < \epsilon_1 \frac{mq}{2^{n-1}} + \frac{1}{2}\,\text{ulp}\,(2C_\ell),$$

which would imply

$$|Cq - p| < \epsilon_1 q + \frac{2^{n-1}}{m^*}\,\text{ulp}\,(C_\ell), \tag{12}$$

where $m^* = \lceil X_{\text{cut}}/q \rceil$ is the smallest possible value of m. Hence, if Condition (12) is not satisfied, convergent p/q cannot generate a bad case for Algorithm 1.

Now, if Condition (12) is satisfied, we have to check Algorithm 1 will all values $X = mq$, with $m^* \leq m \leq \lfloor (2^n - 1)/q \rfloor$.

2) If $x < x_{cut}$: if

$$\left|Cx - \frac{A}{2^n}\right| < \epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}})$$

then

$$\left|2C - \frac{A}{X}\right| < 2^n \times \frac{\epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}})}{X}.$$

Therefore, since $X \leq X_{\text{cut}}$, if

$$\epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}}) \leq \frac{1}{2^{n+1}X_{\text{cut}}} \tag{13}$$

then we can apply Theorem 2: if $|Cx - A/2^n| < \epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}})$ then A/X is a convergent of $2C$.

In that case, we have to check the convergents of $2C$ of denominator less than or equal to X_{cut}. A given convergent p/q (with $\gcd(p,q) = 1$) is a candidate for generating a value X for which Algorithm 1 does not work if there exist $X = mq$ and $A = mp$ such that

$$\begin{cases} 2^{n-1} \leq X \leq X_{\text{cut}} \\ 2^n + 1 \leq A \leq 2^{n+1} - 1 \\ \left|\frac{CX}{2^{n-1}} - \frac{A}{2^n}\right| < \epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}}). \end{cases}$$

This would mean

$$\left|C\frac{mq}{2^{n-1}} - \frac{mp}{2^n}\right| < \epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}}),$$

which would imply

$$\begin{aligned} &|2Cq - p| \\ &< \frac{2^n}{m^*}\left(\epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}})\right), \end{aligned} \tag{14}$$

[1]It is possible that y be not between 1 and x_{cut}. It will anyway be a counterexample, i.e., an n-bit number for which Algorithm 1 fails.

where $m^* = \lceil 2^{n-1}/q \rceil$ is the smallest possible value of m. Hence, if (14) is not satisfied, convergent p/q cannot generate a bad case for Algorithm 1.

Now, if (14) is satisfied, we have to check Algorithm 1 will all values $X = mq$, with $m^* \le m \le \lfloor X_{\text{cut}}/q \rfloor$.

This last result and (4) make it possible to deduce:

Theorem 3 (Conditions on C and n): Assume $1 < C < 2$. Let $x_{\text{cut}} = 2/C$, and $X_{\text{cut}} = \lfloor 2^{n-1} x_{\text{cut}} \rfloor$.

- If $X = 2^{n-1}x > X_{\text{cut}}$ and $2^{2n+1}\epsilon_1 + 2^{2n-1}\,\text{ulp}\,(2C_\ell) \le 1$ then Algorithm 1 will always return a correctly rounded result, except possibly if X is a multiple of the denominator of a convergent p/q of C for which $|Cq - p| < \epsilon_1 q + \frac{2^{n-1}}{\lceil X_{\text{cut}}/q \rceil}\,\text{ulp}\,(C_\ell)$;
- if $X = 2^{n-1}x \le X_{\text{cut}}$ and $\epsilon_1 x_{\text{cut}} + 1/2\,\text{ulp}\,(C_\ell x_{\text{cut}}) \le 1/(2^{n+1}X_{\text{cut}})$ then Algorithm 1 will always return a correctly rounded result, except possibly if X is a multiple of the denominator of a convergent p/q of $2C$ for which $|2Cq - p| < \frac{2^n}{\lceil 2^{n-1}/q \rceil}\left(\epsilon_1 x_{\text{cut}} + \frac{1}{2}\,\text{ulp}\,(C_\ell x_{\text{cut}})\right)$.

$\square$

C. *Method 3: refinement of Method 2*

When Method 2 fails to return an answer, we can use the following method.

We have $|C - C_h| < 2^{-n}$, hence $\text{ulp}\,(C_\ell) \le 2^{-2n}$.

1) If $x < x_{cut}$: if $\text{ulp}\,(C_\ell) \le 2^{-2n-2}$ then we have

$$|u_2 - Cx| < \frac{1}{2}\,\text{ulp}\,(u_2) + 2^{-2n-1}.$$

For any integer A, the inequality

$$\left| Cx - \frac{2A+1}{2^n} \right| \le \frac{1}{2^{2n+1}}$$

implies

$$|2CX - 2A - 1| \le \frac{1}{2^{n+1}} < \frac{1}{2X} :$$

$(2A+1)/X$ is necessarily a convergent of $2C$ from Theorem 2. It suffices then to check, as indicated in Method 2, the convergents of $2C$ of denominator less or equal to X_{cut}.

Now, assume $\text{ulp}\,(C_\ell) \ge 2^{-2n-1}$. We have,

$$-\,\text{ulp}\,(C_\ell) + C_\ell \frac{X}{2^{n-1}} \le u_1 \le \text{ulp}\,(C_\ell) + C_\ell \frac{X}{2^{n-1}}$$

i.e.,

$$\begin{aligned} &-2^{2n}\,\text{ulp}\,(C_\ell) + 2^{n+1}C_\ell X \\ &\quad \le u_1 2^{2n} \\ &\quad \le 2^{2n}\,\text{ulp}\,(C_\ell) + 2^{n+1}C_\ell X. \end{aligned} \tag{15}$$

We look for the integers X, $2^{n-1} \le X \le X_{\text{cut}}$, such that there exists an integer A, $2^{n-1} \le A \le 2^n - 1$, with

$$\left| C_h \frac{X}{2^{n-1}} + u_1 - \frac{2A+1}{2^n} \right| < 2\,\text{ulp}\,(C_\ell)$$

i.e.,

$$\left| \frac{C_h X}{2^n\,\text{ulp}\,(C_\ell)} + \frac{u_1}{2\,\text{ulp}\,(C_\ell)} - \frac{2A+1}{2^{n+1}\,\text{ulp}\,(C_\ell)} \right| < 1.$$

Since $u_1/(2\,\text{ulp}\,(C_\ell))$ is half an integer and $\frac{C_h X}{2^n\,\text{ulp}\,(C_\ell)}$ and $\frac{2A+1}{2^{n+1}\,\text{ulp}\,(C_\ell)}$ are integers, we have

$$\frac{C_h X}{2^n\,\text{ulp}\,(C_\ell)} + \frac{u_1}{2\,\text{ulp}\,(C_\ell)} - \frac{2A+1}{2^{n+1}\,\text{ulp}\,(C_\ell)} = 0, \pm 1/2.$$

Then, combining these three equations with inequalities (15), we get the following three pairs of inequalities

$$\begin{aligned} 0 \le 2X(C_h + C_\ell) - (2A+1) + 2^n\,\text{ulp}\,(C_\ell) \\ \le 2^{n+1}\,\text{ulp}\,(C_\ell), \end{aligned}$$

$$\begin{aligned} 0 \le 2X(C_h + C_\ell) - (2A+1) \\ \le 2^{n+1}\,\text{ulp}\,(C_\ell), \end{aligned}$$

$$\begin{aligned} 0 \le 2X(C_h + C_\ell) - (2A+1) + 2^{n+1}\,\text{ulp}\,(C_\ell) \\ \le 2^{n+1}\,\text{ulp}\,(C_\ell). \end{aligned}$$

For $y \in \mathbb{R}$, let $\{y\}$ be the fractional part of y: $\{y\} = y - \lfloor y \rfloor$. These three inequalities can be rewritten as

$$\{2X(C_h + C_\ell) + 2^n\,\text{ulp}\,(C_\ell)\} \le 2^{n+1}\,\text{ulp}\,(C_\ell),$$

$$\{2X(C_h + C_\ell)\} \le 2^{n+1}\,\text{ulp}\,(C_\ell),$$

$$\{2X(C_h + C_\ell) + 2^{n+1}\,\text{ulp}\,(C_\ell)\} \le 2^{n+1}\,\text{ulp}\,(C_\ell).$$

We use an efficient algorithm due to V. Lefèvre [7] to determine the integers X solution of each inequality.

2) If $x > x_{cut}$: if $\text{ulp}\,(C_\ell) \le 2^{-2n-1}$ then we have

$$|u_2 - Cx| < \frac{1}{2}\,\text{ulp}\,(u_2) + 2^{-2n}.$$

Therefore, for any integer A, the inequality

$$\left| Cx - \frac{2A+1}{2^{n-1}} \right| \le \frac{1}{2^{2n}}$$

is equivalent to

$$|CX - 2A - 1| \le \frac{1}{2^{n+1}} < \frac{1}{2X},$$

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

Now, assume $\mathrm{ulp}(C_\ell) = 2^{-2n}$. We look for the integers X, $X_{\text{cut}} + 1 \le X \le 2^n - 1$, such that there exists an integer A, $2^{n-1} \le A \le 2^n - 1$, with

$$\left| C_h \frac{X}{2^{n-1}} + u_1 - \frac{2A+1}{2^{n-1}} \right| < \frac{1}{2^{2n}}$$

i.e.,

$$\left| 2^{n+1} C_h X + u_1 2^{2n} - 2^{n+1}(2A+1) \right| < 1.$$

Since $u_1 2^{2n}$, $2^{n+1} C_h X$ and $2^{n+1}(2A+1) \in \mathbb{Z}$, we have

$$2^{n+1} C_h X + u_1 2^{2n} - 2^n (2A+1) = 0.$$

Then, combining this equation with inequalities (15), we get the inequalities

$$0 \le X(C_h + C_\ell) - (2A+1) + \frac{1}{2^{n+1}} \le \frac{1}{2^n},$$

that is to say

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

Here again, we use Lefèvre's algorithm [7] to determine the integers X solution of this inequality.

V. Examples

A. Example 1: multiplication by π in double precision

Consider the case $C = \pi/2$ (which corresponds to multiplication by any number of the form $2^{\pm j}\pi$), and $n = 53$ (which corresponds to double precision), and assume we use Method 1. We find:

$$\begin{cases} C_h & = & 884279719003555/562949953421312, \\ C_\ell & = & 6.123233996\cdots \times 10^{-17}, \\ \epsilon_1 & = & 1.497384905\cdots \times 10^{-33}, \\ x_{\text{cut}} & = & 1.2732395447351626862\cdots, \\ \mathrm{ulp}(C_\ell x_{\text{cut}}) & = & 2^{-106}, \\ \mathrm{ulp}(C_\ell) & = & 2^{-106}. \end{cases}$$

Hence,

$$\begin{cases} 2^n \alpha & = & 7.268364390 \times 10^{-17}, \\ 2^{n-1}\alpha' & = & 6.899839541 \times 10^{-17}. \end{cases}$$

Computing the convergents of $2C$ and C we find

$$\frac{p_k}{q_k} = \frac{6134899525417045}{1952799169684491}$$

and $\delta = 9.495905771 \times 10^{-17} > 2^n\alpha$ (which means that Algorithm 1 works for $x < x_{\text{cut}}$), and

$$\frac{p'_{k'}}{q'_{k'}} = \frac{12055686754159438}{7674888557167847}$$

and $\delta' = 6.943873667 \times 10^{-17} > 2^{n-1}\alpha'$(which means that Algorithm 1 works for $x > x_{\text{cut}}$). We therefore deduce:

Theorem 4 (Correctly rounded multiplication by π): Algorithm 1 always returns a correctly rounded result in double precision with $C = 2^j\pi$, where j is any integer, provided no under/overflow occur.

□

Hence, in that case, multiplying by π with correct rounding only requires 2 consecutive fused-macs.

B. Example 2: multiplication by $\ln(2)$ in double precision

Consider the case $C = 2\ln(2)$ (which corresponds to multiplication by any number of the form $2^{\pm j}\ln(2)$), and $n = 53$, and assume we use Method 2. We find:

$$\begin{cases} C_h & = & \frac{6243314768165359}{4503599627370496}, \\ C_\ell & = & 4.638093628\cdots \times 10^{-17}, \\ x_{\text{cut}} & = & 1.442695\cdots, \\ \epsilon_1 & = & 1.141541688\cdots \times 10^{-33}, \\ \epsilon_1 x_{\text{cut}} & & \\ +\frac{1}{2}\,\mathrm{ulp}(C_\ell x_{\text{cut}}) & = & 7.8099\cdots \times 10^{-33}, \\ 1/(2^{n+1}X_{\text{cut}}) & = & 8.5437\cdots \times 10^{-33}. \end{cases}$$

Since $\epsilon_1 x_{\text{cut}} + 1/2\,\mathrm{ulp}(C_\ell x_{\text{cut}}) \le 1/(2^{n+1}X_{\text{cut}})$, to find the possible bad cases for Algorithm 1 that are less than x_{cut}, it suffices to check the convergents of $2C$ of denominator less than or equal to X_{cut}. These convergents are:

```
2, 3, 11/4, 25/9, 36/13, 61/22, 890/321, 2731/985,
25469/9186, 1097898/395983, 1123367/405169,
2221265/801152,16672222/6013233, 18893487/6814385,
35565709/12827618, 125590614/45297239,
161156323/58124857, 609059583/219671810,
1379275489/497468477, 1988335072/717140287,
5355945633/1931749051, 7344280705/2648889338,
27388787748/9878417065, 34733068453/12527306403,
62121856201/22405723468, 96854924654/34933029871,
449541554817/162137842952,
2794104253556/1007760087583,
3243645808373/1169897930535,
6037750061929/2177658018118,
39470146179947/14235846039243,
124448188601770/44885196135847,
163918334781717/59121042175090,
288366523383487/104006238310937,
6219615325834944/2243252046704767.
```

None of them satisfies condition (14). Therefore there are no bad cases less than x_{cut}. Processing the case $x > x_{\text{cut}}$ is similar and gives the same result, hence:

Theorem 5 (Correctly rounded multiplication by $\ln(2)$): Algorithm 1 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.

□

C. 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:

$$\begin{cases} C_h & = \frac{5734161139222659}{4503599627370496}, \\ C_\ell & = -7.871470670\cdots \times 10^{-17}, \\ \epsilon_1 & = 4.288574513\cdots \times 10^{-33}, \\ x_{\text{cut}} & = 1.570796\cdots, \\ C_\ell x_{\text{cut}} & = -1.236447722\cdots \times 10^{-16}, \\ \text{ulp}\,(C_\ell x_{\text{cut}}) & = 2^{-105}, \\ 2^n\alpha & = 1.716990939\cdots \times 10^{-16}, \\ p_k/q_k & = \frac{15486085235905811}{6081371451248382}, \\ \delta & = 7.669955467\cdots \times 10^{-17}. \end{cases}$$

Consider the case $x < x_{\text{cut}}$. Since $\delta < 2^n\alpha$, there can be bad cases for Algorithm 1. We try Algorithm 1 with X equal to the denominator of p_k/q_k, that is, 6081371451248382, and we find that it does not return $\circ(cX)$ for that value. Hence, *there is at least one value of x for which Algorithm 1 does not work.*

Method 3 certifies that $X = 6081371451248382$, i.e., $6081371451248382 \times 2^{\pm k}$ are the *only* FP values for which Algorithm 1 fails.

D. Example 4: multiplication by $\sqrt{2}$ in single precision

Consider the case $C = \sqrt{2}$, and $n = 24$ (which corresponds to single precision), and assume we use Method 1. We find:

$$\begin{cases} C_h & = 11863283/8388608, \\ C_\ell & = 2.420323497\cdots \times 10^{-8}, \\ \epsilon_1 & = 7.628067479\cdots \times 10^{-16}, \\ X_{\text{cut}} & = 11863283, \\ \text{ulp}\,(C_\ell x_{\text{cut}}) & = 2^{-48}, \\ 2^n\alpha & = 4.790110735\cdots \times 10^{-8}, \\ p_k/q_k & = 22619537/7997214, \\ \delta & = 2.210478490\cdots \times 10^{-8}, \\ 2^{n-1}\alpha' & = 2.769893477\cdots \times 10^{-8}, \\ p_{k'}/q_{k'} & = 22619537/15994428, \\ \delta' & = 2.210478490\cdots \times 10^{-8}. \end{cases}$$

Since $2^n\alpha > \delta$ and $X = q_k = 7997214$ is not a bad case, we cannot conclude in the case $x < x_{\text{cut}}$. Also, since $2^{n-1}\alpha' > \delta'$ and $X = q_{k'} = 15994428$ is not a bad case, we cannot conclude in the case $x \geq x_{\text{cut}}$. Hence, in the case $C = \sqrt{2}$ and $n = 24$, Method 1 does not allow us to know if the multiplication algorithm works for any input FP number x. In that case, Method 2 also fails. And yet, Method 3 or exhaustive testing (which is possible since $n = 24$ is reasonably small) show that Algorithm 1 always works.

VI. Implementation and results

As the reader will have guessed from the previous examples, using Method 1 or Method 2 by paper and pencil calculation is fastidious and error-prone (this is even worse with Method 3). We have written Maple programs that implement Methods 1 and 2, and a GP/PARI[2] program that implements Method 3. They allow any user to quickly check, for a given constant C and a given number n of mantissa bits, if Algorithm 1 works for any x, and Method 3 gives all values of x for which it does not work (if there are such values). These programs can be downloaded from the url

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

These programs, along with some examples, are given in the appendix. Table II presents some obtained results. They show that implementing Method 1, Method 2 *and* Method 3 is necessary: Methods 1 and 2 do not return a result (either a bad case, or the fact that Algorithm 1 always works) for the same values of C and n. For instance, in the case $C = \pi/2$ and $n = 53$, we know thanks to Method 1 that the multiplication algorithm always works, whereas Method 2 fails to give an answer. On the contrary, in the case $C = 1/\ln(2)$ and $n = 24$, Method 1 does not give an answer, whereas Method 2 makes it possible to show that the multiplication algorithm always works. Method 3 always returns an answer, but is and more complicated to implement: this is not a problem for getting in advance a result such as Theorem 4, for a general constant C. And yet, this might make method 3 difficult to implement in a compiler, to decide at compile-time if we can use our multiplication algorithm.

VII. Conclusion

The three 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 fused-mac). For instance, in double precision arithmetic, we can multiply by π or $\ln(2)$ with correct rounding. Interestingly enough, although it is always possible to build *ad hoc* values of C for which Algorithm 1 fails, for "general" values of C, our experiments show that Algorithm 1 works for most values of n.

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions with formulas, graphs and mathematical tables.* Applied Math. Series 55. National Bureau of Standards, Washington, D.C., 1964.

[2] N. Brisebarre, J.-M. Muller, and S. Raina. Accelerating correctly rounded floating-point division when the divisor is known in advance. *IEEE Transactions on Computers*, 53(8):1069–1072, August 2004.

[2]`http://pari.math.u-bordeaux.fr/`

C	n	method 1	method 2	method 3
π	8	Does not work for 226	Does not work for 226	AW 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)
$1/\pi$	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)
$\frac{1}{\ln 2}$	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)
$\frac{2^j}{\ln 10}$	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)

TABLE II

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.*

[3] M. A. Cornea-Hasegan, R. A. Golliver, and P. Markstein. Correctness proofs outline for newton-raphson based floating-point divide and square root algorithms. In Koren and Kornerup, editors, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia)*, pages 96–105, Los Alamitos, CA, April 1999. IEEE Computer Society Press.

[4] B. P. Flannery, W. H. Press, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes in C*. Cambridge University Press, 2 edition, 1992.

[5] G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers*. Oxford University Press, 1979.

[6] A. Ya. Khinchin. *Continued Fractions*. Dover, New York, 1997.

[7] V. Lefèvre. *Developments in Reliable Computing*, chapter An Algorithm That Computes a Lower Bound on the Distance Between a Segment and $\mathbb{Z}^2$, pages 203–212. Kluwer, Dordrecht, Netherlands, 1999.

[8] P. Markstein. *Ia-64 and Elementary Functions : Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000. ISBN: 0130183482.

[9] P. W. Markstein. Computation of elementary functions on the IBM risc system/6000 processor. *IBM Journal of Research and Development*, 34(1):111–119, January 1990.

[10] O. Perron. *Die Lehre von den Kettenbruchen, 3. verb. und erweiterte Aufl*. Teubner, Stuttgart, 1954-57.

[11] H. M. Stark. *An Introduction to Number Theory*. MIT Press, Cambridge, MA, 1981.