

Exact and Approximated error of the FMA

Sylvie Boldo, Jean-Michel Muller, *Senior member, IEEE*.

This work was partially funded by the French national research organization (ANR) by the F ϕ st project (ANR-08-BLAN-0246-01) and the EvaFlo project and by the Hisseo project funded by Digiteo.

S. Boldo is with the INRIA Saclay - Île-de-France, ProVal, Orsay, F-91893 and LRI, Univ Paris-Sud, CNRS, Orsay, F-91405.
J.-M. Muller is with CNRS, Université de Lyon, Laboratoire LIP, and INRIA/Arénaire

Abstract

The fused multiply accumulate-add (FMA) instruction, specified by the IEEE 754-2008 Standard for Floating-Point Arithmetic, eases some calculations, and is already available on some current processors such as the Power PC or the Itanium. We first extend an earlier work on the computation of the exact error of an FMA (by giving more general conditions and providing a formal proof). Then, we present a new algorithm that computes an approximation to the error of an FMA, and provide error bounds and a formal proof for that algorithm.

Index Terms

Floating-Point arithmetic, FMA, fused multiply-add, computer arithmetic.

I. INTRODUCTION

The fused multiply-add (FMA) instruction makes it possible to evaluate $\pm ax \pm b$, where a , x , and b are floating-point numbers, with one final rounding only. That instruction was introduced in 1990 on the IBM RS/6000 processor [1], [2]. It allows for faster and, in general, more accurate dot products, matrix multiplications, and polynomial evaluations. It also makes it possible to design fast algorithms for correctly rounded division and square root [3], [4], which explains why, on current chips offering such an instruction, there is no hardwired division. An FMA also eases the design of an accurate range reduction algorithm for the trigonometric functions [5].

After the IBM RS/6000, FMA units were implemented in several general-purpose processors. Examples are the IBM PowerPC [6], the HP PA-8000 [7], [8], and the HP/Intel Itanium [9]. An interesting survey on FMA architectures, along with suggestions for new architectures, is presented in [10].

The FMA instruction is included in the newly revised IEEE 754-2008 standard for floating-point arithmetic [11]. An important consequence of this is that within a few years, this instruction will probably be available on most general-purpose processors.

It is well known [12], [13], [14], [15], [16] that (under some assumptions such as requiring rounding to nearest in the case of addition and square root, or assumptions on the radix, see [4] for more details) the error of a floating point addition or multiplication, or the remainder of a division or square root is exactly representable as a floating-point number of the same format,

and is computable using reasonably simple algorithms, some of which will be quickly recalled in Section II. A natural question arises: is there a similar property for the FMA operation?

We addressed this question in [17] in the case of radix-2 arithmetic and assuming rounding to nearest. We showed that two floating-point numbers always suffice for representing the error of an FMA, and we gave an algorithm for computing these two numbers. The total number of floating-point operations it requires is 20. That algorithm was for instance used by Louvet [18] for building a fast *compensated* polynomial evaluation algorithm.

Nevertheless, the proofs of [17] were only in radix 2, and were only pen-and-paper proofs. To increase the trust in this algorithm, we have formally proved it, using the Coq proof checker, and tried to get results as general as possible (for instance, we no longer require the radix to be 2). This proof will be the first result presented in this paper, in Section III.

Also, in many applications (compensated algorithms being a typical example), computing the error of an FMA *exactly* may not be necessary: if there exists a much faster algorithm that provides a good approximation to that error, it may be preferable to use it, provided we have a bound on the approximation error. We deal with this problem in Section IV

II. BASIC OPERATIONS

Let us now present some basic algorithms, called *error-free transforms* by Rump [19], that allow one, under some conditions, to compute the error of a floating-point addition or multiplication exactly.

In the following, \circ denotes the rounding operation. For instance, if a and b are floating-point (FP) numbers, $\circ(a + b)$ is the *computed*, floating-point approximation to $a + b$, whereas $a + b$ is the *exact*, real value of $a + b$. On systems compliant with IEEE 754-2008, the default rounding operation is round-to-nearest even: $\circ(x)$ is the floating-point number nearest to x , and in the case of a tie—i.e., if there are two FP numbers nearest x — the one with an even significand is returned.

The *radix* of the FP format will be denoted by β , and its *precision* (i.e., the number of digits of the significands) by p . The minimal exponent will be e_{\min} in the IEEE standard meaning. This means that the smallest normal FP number is $\beta^{e_{\min}}$ and the smallest positive FP number is $\beta^{e_{\min}-p+1}$.

We will assume in all our proofs that there is no Overflow. Nevertheless, we have looked into all our algorithms: they may create an unjustified Overflow (especially if $a \times x$ does overflow but $a \times x - b$ does not), but if so, they will forward infinities. They cannot be any hidden overflow in these algorithms: one will always get an infinity as result if an overflow occurs at any point.

A. Algorithm Fast2Sum

Fast2Sum first appears in a paper by Dekker [14] in 1971. Assume that \circ is round-to-nearest, and that $\beta \leq 3$. Let a and b be floating-point numbers such that the exponent of a is larger than or equal to that of b . The following algorithm computes two FP numbers s and ρ such that $s + \rho = a + b$ exactly, and $s = \circ(a + b)$ (i.e., ρ is the error of the FP addition of a and b).

Note that $|a| \geq |b|$ implies $e_a \geq e_b$, the needed requirement for this algorithm.

Algorithm 1 (Fast2Sum(a, b)):

$$\begin{aligned} s &= \circ(a + b) \\ t &= \circ(s - a) \\ \rho &= \circ(s - t) \end{aligned}$$

B. Algorithm 2Sum

Fast2Sum only requires 3 floating-points additions, and yet it has two drawbacks: first, it does not always work in radices larger than 3 (that is, in practice, in radix 10), and second, the condition “the exponent of a is larger than or equal to that of b ” may require a comparison of a and b : on recent processors, a wrong branch prediction when performing this comparison may cost much. Hence, in many cases it may be preferable to use the following algorithm, due to Knuth [13].

Algorithm 2 (2Sum(a, b)):

$$\begin{aligned} s &= \circ(a + b) \\ \hat{a} &= \circ(s - b) \\ \hat{b} &= \circ(s - \hat{a}) \\ \epsilon_a &= \circ(a - \hat{a}) \\ \epsilon_b &= \circ(b - \hat{b}) \\ \rho &= \circ(\epsilon_a + \epsilon_b) \end{aligned}$$

Knuth [13] showed that, if a and b are normal FP numbers, then for any value of β , provided that no underflow or overflow occurs, $a + b = s + \rho$. Boldo et al. [20] showed that in radix 2, this result still holds in the presence of underflow.

C. Algorithm Fast2Mult

If no FMA instruction is available, there exists an algorithm, due to Dekker, that computes the error of a FP multiplication using 17 FP operations (multiplications and additions/subtractions). On systems with an FMA instruction, the same calculation is performed much more quickly, using the following, straightforward, algorithm, that works for any value of β .

Algorithm 3 (Fast2Mult(a, b)):

$$\begin{aligned} t &= \circ(a \cdot b) \\ \rho &= \circ(a \cdot b - r) \end{aligned}$$

Let e_a and e_b be the floating-point exponents of a and b . If $e_a + e_b \geq e_{\min} + p - 1$ then the number ρ computed by Fast2Mult(a, b) is exactly equal to the error of the FP multiplication $\circ(a \cdot b)$. Notice that the condition $e_a + e_b \geq e_{\min} + p - 1$ cannot be avoided: if it is not satisfied, then $t - a \cdot b$ may not be a FP number.

III. EXACT ERROR OF THE FMA

A. Algorithm

We presented in [17] the following algorithm to compute the exact error of an FMA. The inputs values are three FP numbers a, x and y . The output values are r_1, r_2 and r_3 .

Algorithm 4 (ErrFma):

$$\begin{aligned} r_1 &= \circ(ax + y) \\ (u_1, u_2) &= \text{Fast2Mult}(a, x) \\ (\alpha_1, \alpha_2) &= \text{2Sum}(y, u_2) \\ (\beta_1, \beta_2) &= \text{2Sum}(u_1, \alpha_1) \\ \gamma &= \circ(\circ(\beta_1 - r_1) + \beta_2) \\ (r_2, r_3) &= \text{Fast2Sum}(\gamma, \alpha_2) \end{aligned}$$

Property 1 (ErrFma_correctness): Assuming radix 2, round-to-nearest and no underflows / overflows, we showed in [17] that Algorithm 4 satisfies:

- $ax + y = r_1 + r_2 + r_3$ exactly;
- $|r_2 + r_3| \leq \frac{1}{2}\text{ulp}(r_1)$;
- $|r_3| \leq \frac{1}{2}\text{ulp}(r_2)$.

□

Therefore, if instead of exactly computing the error of an FMA as a sum of two FP numbers we just want to compute the FP number nearest that error, it is trivial to get it:

Algorithm 5 (ErrFmaNearest):

$$\begin{aligned}
 r_1 &= \circ(ax + y) \\
 (u_1, u_2) &= \text{Fast2Mult}(a, x) \\
 (\alpha_1, \alpha_2) &= \text{2Sum}(y, u_2) \\
 (\beta_1, \beta_2) &= \text{2Sum}(u_1, \alpha_1) \\
 \gamma &= \circ(\circ(\beta_1 - r_1) + \beta_2) \\
 r_2 &= \circ(\gamma + \alpha_2)
 \end{aligned}$$

From the results of [17], we easily deduce that

$$|r_1 + r_2 - (ax + y)| \leq \frac{1}{2}\text{ulp}(r_2).$$

B. Formal proof

Nevertheless, the proofs of [17] were only in radix 2, and were only pen-and-paper proofs. As the proof is complex and has many sub-cases (for example, $\beta_2 = 0$ or not) and to increase the trust in this algorithm, we have formally proved Algorithm 4, that directly gives us the correctness of Algorithm 5. Also, building a formal proof forces to detail all possible cases of underflow of an intermediate variable: this tedious (and somewhat error-prone) task is almost always skipped or overlooked in paper-and-pencil proofs.

The exact Coq theorem is given below. Its counterpart in mathematical language is the following:

Theorem 1: Let p be the number of digits with $p \geq 3$. Let β be the radix with $\beta > 1$. We assume that β is even, and that \circ is any coherent round-to-nearest mode. This means that the rounding must be a rounding to nearest, but done in a coherent way (a real number always

rounds to the same FP value). This is the case especially for the usual round-to-nearest, ties to even and for the round-to-nearest, ties away from zero defined by the IEEE 754-2008 standard.

Let a, b, x be floating-point numbers (either normal or subnormal!).

Let $r_1, u_1, u_2, \alpha_1, \alpha_2, \beta_1, \beta_2, \gamma$ be computed as in Algorithm 4.

Then we assume a few non-underflow hypotheses:

- either $\alpha_1 = 0$ or $\beta^{e_{\min}+1} \leq |\alpha_1|$
- either $u_1 = 0$ or $\beta^{e_{\min}+1} \leq |u_1|$
- either $\beta_1 = 0$ or $\beta^{e_{\min}+2} \leq |\beta_1|$
- either $r_1 = 0$ or r_1 is normal
- the exponents of a and b are such that $e_a + e_x \geq e_{\min} + p - 1$ (so the error of ax is a FP).

Then

$$a \times x + y = r_1 + \gamma + \alpha_2$$

□

Note that there is no requirement on the radix except being even: the algorithm works in radix 2, 10, 16. This limit is due to the fact that $\frac{1}{2}\text{ulp}(f)$ is considered a floating-point number. It greatly simplifies the proof as odd radices should be looked upon specifically. We do not believe this constraint is a problem for any real-life system. The only actually built odd-radix system we are aware of is the SETUN computer, built in USSR in the late fifties.

Note that we also proved an additional property for Algorithm 4. For $\beta \leq 3$, the correctness of the Fast2Sum Algorithm is guaranteed by the existence of floats α'_2 and γ' such that $\alpha'_2 = \alpha_2$ and $\gamma' = \gamma$ and α'_2 and γ' fit in the floating-point format and $e_{\alpha'_2} \leq e_{\gamma'}$.

```

Variable bo : Fbound.
Variable radix : Z.
Variable p : nat.

Hypothesis pGivesBound : Zpos (vNum bo) = Zpower_nat radix p.
Hypothesis radixMoreThanOne : (1 < radix)%Z.
Hypothesis precisionGreaterThenOne : 3 <= p.
Hypothesis Evenradix: (Even radix).

Variable P: R -> float -> Prop.
Hypothesis P1: forall (r:R) (f:float), (P r f) -> (Closest bo radix r f).
Hypothesis P2: forall (r1 r2:R) (f1 f2:float),

```

$$(P \ r1 \ f1) \rightarrow (P \ r2 \ f2) \rightarrow (r1=r2)\%R \rightarrow (FtoRradix \ f1=f2)\%R.$$

Variable a x y r1 u1 u2 a11 a12 be1 be2 gat ga :float.

Hypothesis Fa : Fbounded bo a.

Hypothesis Fx : Fbounded bo x.

Hypothesis Fy : Fbounded bo y.

Hypothesis Nbe1: Fcanonic radix bo be1.

Hypothesis Nr1 : Fcanonic radix bo r1.

Hypothesis Ca11: Fcanonic radix bo a11.

Hypothesis Cu1 : Fcanonic radix bo u1.

Hypothesis Exp1: (- dExp bo < Fexp a11)%Z \vee (FtoRradix a11=0)%R.

Hypothesis Exp2: (- dExp bo < Fexp u1)%Z \vee (FtoRradix u1=0)%R.

Hypothesis Exp3: (- dExp bo+1 < Fexp be1)%Z \vee (FtoRradix be1=0)%R.

Hypothesis Exp4: (Fnormal radix bo r1) \vee (FtoRradix r1=0)%R.

Hypothesis Exp5: (-dExp bo <= Fexp a+Fexp x)%Z.

Hypothesis u1Def: (Closest bo radix (a*x)%R u1).

Hypothesis u2Def: (FtoRradix u2=a*x-u1)%R.

Hypothesis a11Def: (Closest bo radix (y+u2)%R a11).

Hypothesis a12Def: (FtoRradix a12=y+u2-a11)%R.

Hypothesis be2Def: (FtoRradix be2=u1+a11-be1)%R.

Hypothesis gatDef: (Closest bo radix (be1-r1)%R gat).

Hypothesis gaDef: (Closest bo radix (gat+be2)%R ga).

Hypothesis r1DefE: (P (a*x+y)%R r1).

Hypothesis be1DefE: (P (u1+a11)%R be1).

Theorem FmaErr: (a*x+y=r1+ga+a12)%R.

Theorem Fma_FTS: (exists ga_e:float, exists a12_e:float,
 (FtoRradix ga_e=ga)%R \wedge (FtoRradix a12_e=a12)%R
 \wedge (Fbounded bo ga_e) \wedge (Fbounded bo a12_e) \wedge
 (Fexp a12_e <= Fexp ga_e)%Z).

IV. APPROXIMATED ERROR OF THE FMA

Algorithm 5 uses 20 FP operations. We were not able to find an algorithm that returns the same result with fewer operations. And yet, for many applications, really getting the FP number that is nearest the error of an FMA is not necessary: a good approximation to that error may suffice.

Hence, in the following, we aim at being faster than Algorithm 5, and we accept to be (hopefully slightly) less accurate. Let us now present a new algorithm, that only requires 12

floating-point operations.

A. Algorithm

We make no assumption on the radix β (except, of course, that it is an integer larger than or equal to 2). We assume that the precision p is larger than or equal to 4. Even more general than previously, \circ is any round-to-nearest: not even coherence is needed here! Therefore, it works in rounding to nearest, ties to even and in rounding to nearest, ties away from zero.

Algorithm 6 (ErrFmaAppr):

$$\begin{aligned} z &= \circ(ax + b) \\ (p_h, p_l) &= \text{Fast2Mult}(a, x) \\ (u_h, u_l) &= 2\text{Sum}(b, p_h) \\ t &= \circ(u_h - z) \\ z' &= \circ(t + \circ(p_l + u_l)) \end{aligned}$$

We mean to prove that

Property 2 (ErrFmaAppr_correctness): $|z + z' - (ax + b)| \leq \left(\frac{3\beta}{2} + \frac{1}{2}\right) \beta^{2-2p}|z|$.

This implies that $|z + z' - (ax + b)| < 2\beta\beta^{2-2p}|z|$, therefore we have at least $p - 1 - \log_\beta(2)$ correct bits following z as shown in Figure 1.

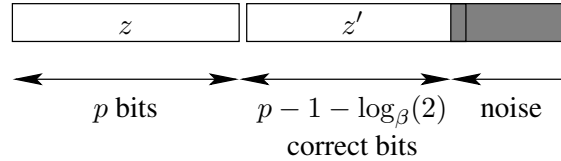


Fig. 1. Correct bits of Algorithm 6

B. Proof

We assume there is neither Underflow, nor Overflow and that the working precision is $p \geq 4$. We proved that, if $f = \circ(r)$ and f is normal, then $|f| \leq \frac{|r|}{1 - \beta^{1-p/2}}$ and $|r| \leq |f|(1 + \beta^{1-p}/2)$.

1) The computation of t is exact:

Property 3: t is computed without error.

First, u_h and z share the same sign: if $ax + b \geq 0$, then $z \geq 0$. Moreover, $ax \geq -b$, so $p_h \geq -b$ so $u_h \geq 0$. The same properties show that when $ax + b \leq 0$, then both z and u_h are non-positive. We split into two sub-cases depending on whether $u_h = o(b + p_h)$ is the result of a huge cancellation, or not.

a) Assuming $|p_l| \leq \frac{|b+p_h|}{4}$: This assumption of no or small cancellation is enough to guarantee that Sterbenz theorem¹ can be applied.

$$|z| \leq \frac{|ax+b|}{1-\beta^{1-p}/2} \leq \frac{|p_h+b|+|p_l|}{1-\beta^{1-p}/2} \leq \frac{5}{4} \frac{|p_h+b|}{1-\beta^{1-p}/2} \leq |u_h| \frac{5}{4} \frac{1+\beta^{1-p}/2}{1-\beta^{1-p}/2} \leq 2|u_h| \text{ as } p \geq 4.$$

Moreover, $|p_h + b| = |p_h + p_l + b - p_l| \leq |p_h + p_l + b| + |p_l| \leq |ax + b| + \frac{|b+p_h|}{4}$, so $|p_h + b| \leq \frac{4}{3}|ax + b|$.

$$|u_h| \leq \frac{|p_h+b|}{1-\beta^{1-p}/2} \leq \frac{4}{3} \frac{|ax+b|}{1-\beta^{1-p}/2} \leq |z| \frac{4}{3} \frac{1+\beta^{1-p}/2}{1-\beta^{1-p}/2} \leq 2|z| \text{ as } p \geq 4.$$

Therefore $|u_h| - |z| = \pm(u_h - z)$ is representable and $u_h - z$ is computed exactly. \square

b) Assuming $\frac{|b+p_h|}{4} < |p_l|$: This assumption means a huge cancellation, and $p_h \approx -b$. This also implies that $p_l \neq 0$, therefore the exponent of p_h cannot be the minimal exponent: $e_{p_h} > e_{\min} - p + 1$ and $e_{p_h} > e_a + e_x$ as p_l can be represented with exponent $e_a + e_x$ and is nonzero.

Moreover $|p_h + b| < 4|p_l| \leq 2\text{ulp}(p_h) = 2\beta^{e_{p_h}}$.

Also, $|b| \geq |p_h| - |p_h + b| \leq |p_h| - 2\beta^{e_{p_h}}$, therefore $|b| \geq 2|p_h|$ and $e_{p_h} - 1 \leq e_b$. We easily prove that $|p_h| \geq 2|b|$. Therefore the computation of $b + p_h$ is exact and the result can be expressed with the exponent $e_{p_h} - 1$.

Furthermore, $ax + b$ can be represented with exponent $e_a + e_x$, as $e_b \geq e_{p_h} - 1 \geq e_a + e_x$. So $z = o(ax + b)$ can be represented with exponent $e_a + e_x$.

Finally $u_h - z$ can be represented with exponent $e_a + e_x$. There is only left to prove that the significand implied is bounded. We have,

$$|u_h - z| \beta^{-e_a - e_x} = |p_h + b - z| \beta^{-e_a - e_x} \leq (|p_l| + |ax + b - z|) \beta^{-e_a - e_x} \leq \frac{1}{2} (\beta^{e_{p_h} - e_a - e_x} + \beta^{e_z - e_a - e_x})$$

Moreover, $e_z < e_{p_h}$ as $|ax + b| \leq |p_h + b| + |p_l| \leq 5|p_l| \leq 3\beta^{e_{p_h}}$ so $|z| \leq 3\beta^{e_{p_h}}$.

¹Sterbenz Theorem says that if two FP numbers x and y are such that $x/2 \leq y \leq 2x$ then $x - y$ is computed exactly.

Therefore $|u_h - z| \beta^{-e_a - e_x} < \beta^{e_{p_h} - e_a - e_x}$. There is left to prove that $e_{p_h} - e_a - e_x \leq p$, which is easy since $|ax| \leq (\beta^p - 1)^2 \beta^{e_a + e_x} \leq (\beta^p - 1) \beta^{p + e_a + e_x}$. Therefore $|p_h| \leq (\beta^p - 1) \beta^{p + e_a + e_x}$ and $e_{p_h} \leq p + e_a + e_x$. \square

2) *When $u_h = b + p_h$:* This is the easy case. Nevertheless, it eases the proof to split it into the two sub-cases $u_h = b + p_h$ and $u_h \neq b + p_h$.

Property 4: In this case, Theorem 2 holds.

In fact, the hypothesis means that $u_l = 0$ so $z' = \circ(t + p_l)$.

Therefore, $z + z' - (ax + b) = u_h - t + \circ(t + p_l) - p_h - p_l - b = \circ(t + p_l) - t - p_l$ and $|z + z' - (ax + b)| \leq \frac{1}{2} \text{ulp}(z')$.

Moreover, $|z'| \leq |z - (ax + b)| + |\circ(t + p_l) - t - p_l|$ so $|z'| - \frac{1}{2} \text{ulp}(z') \leq \frac{1}{2} \text{ulp}(z)$.

This easily implies that $|z'| \leq \text{ulp}(z)$ and that

$$|z + z' - (ax + b)| \leq \frac{1}{2} \text{ulp}(z') \leq \frac{1}{2} \beta^{1-p} |z'| \leq \frac{1}{2} \beta^{2-2p} |z| \leq \left(\frac{3\beta}{2} + \frac{1}{2}\right) \beta^{2-2p} |z|.$$

\square

3) *When $u_h \neq b + p_h$:* This assumption guarantees that there is no cancellation in the computation of $\circ(ax) + b$. It allows us to bound the relative exponents of the various FP numbers.

Property 5: $e_{p_h} \leq e_{u_h} + 1$

Let us suppose that $e_{p_h} > e_{u_h} + 1$ so that $e_{u_h} \leq e_{p_h} - 2$. As $u_h = \circ(b + p_h)$, this means that this computation was a cancellation, therefore exact (following Sterbenz Theorem) and that $u_h = b + p_h$, which we assumed was wrong. \square

Property 6: $e_{u_h} \leq e_z + 1$

$$\begin{aligned} |u_h - z| &= |(u_h - (b + p_h)) + (p_h - ax) + (ax + b - z)| \leq \frac{1}{2} \text{ulp}(u_h) + \frac{1}{2} \text{ulp}(p_h) + \frac{1}{2} \text{ulp}(z) \\ &\leq \frac{\beta+1}{2} \text{ulp}(u_h) + \frac{1}{2} \text{ulp}(z) \text{ using the preceding property.} \end{aligned}$$

Therefore $|u_h| \left(1 - \frac{\beta+1}{2} \beta^{1-p}\right) \leq |z| \left(1 + \frac{\beta^{1-p}}{2}\right)$ and $|u_h| \leq \beta |z|$ as $p \geq 4$. \square

Property 7: It cannot happen that $e_{p_h} = e_{u_h} + 1 = e_z + 2$.

We assume these equalities hold.

To prove the absurdity, we will prove that $|z| \geq \beta^{p+e_z}$, which is impossible.

First $|ax + b| \geq |ax| - |b|$. As $p_h = \circ(ax)$, we know that $|ax| \geq (\beta^{p-1} - \frac{1}{2\beta})\beta^{e_{p_h}}$: the smallest possible real number to be rounded into a float with exponent e_{p_h} is the smallest float with this exponent $\beta^{p-1+e_{p_h}}$ minus half the difference between this float and its predecessor.

Furthermore, $|b| \leq (\beta^p - 1)\beta^{e_b}$. And $e_b \leq e_{u_h} - 1 = e_z$: if this was not the case, then the exponent of $u_h = \circ(b + p_h)$ would be smaller than the minimum of the exponents of b and p_h that would imply that the addition $b + p_h$ is correct, which is impossible by assumption.

Then $|ax + b| \geq |ax| - |b| \geq (\beta^{p-1} - \frac{1}{2\beta})\beta^{e_{p_h}} - (\beta^p - 1)\beta^{e_b} \geq (\beta^{p-1} - \frac{1}{2\beta})\beta^{e_z+2} - (\beta^p - 1)\beta^{e_z}$.

And $|ax + b| \geq \beta^{p+1+e_z} - \beta^{p+e_z} - \beta^{e_z+1}/2 + \beta^{e_z} = \beta^{p+e_z}(\beta - 1) - \beta^{e_z}(\beta/2 - 1)$.

When $\beta = 2$, this value is exactly equal to β^{p+e_z} . When $\beta \geq 3$, this value is greater than $\beta^{p+e_z} + \beta^{p+e_z} - \beta^{e_z}(\beta/2 - 1) \geq \beta^{p+e_z}$.

In all cases, we have $|z| \geq \beta^{p+e_z}$, which is impossible. \square

Property 8: $|u_h - z| \leq (\beta + 1)\text{ulp}(z)$

From the 3 last properties, we either have $e_{p_h} \leq e_{u_h} \leq e_z + 1$ or both $e_{p_h} \leq e_{u_h} + 1$ and $e_{u_h} \leq e_z$. It means that $\beta^{e_{u_h}} + \beta^{e_{p_h}} \leq 2 * \beta^{e_z+1}$ in all cases.

Then, $|u_h - z| = |(u_h - (b + p_h)) + (p_h - ax) + (ax + b - z)| \leq \frac{1}{2}\text{ulp}(u_h) + \frac{1}{2}\text{ulp}(p_h) + \frac{1}{2}\text{ulp}(z)$, so $|u_h - z| \leq \beta^{e_z+1} + \frac{1}{2}\beta^{e_z} \leq (\beta + 1)\beta^{e_z}$. \square

Property 9: In this case, Theorem 2 holds.

We first bound $|u_l + p_l| \leq \beta^{e_z+1}$ therefore $|\circ(u_l + p_l)| \leq \beta^{e_z+1}$.

We then bound $|t + \circ(p_l + u_l)| \leq (2\beta + 1)\beta^{e_z}$, therefore $|z'| \leq (2\beta + 1)\beta^{e_z}$.

More $z + z' - (ax + b) = u_h - t + z' - p_h - p_l - b = z' - t - p_l - u_l$ so the error only comes from the computations inside z' that occur on small numbers compared to z .

Therefore

$|z + z' - (ax + b)| \leq \frac{1}{2}\text{ulp}(\circ(p_l + u_l)) + \frac{1}{2}\text{ulp}(z') \leq \frac{1}{2}\beta^{1-p}\beta^{e_z}(3\beta + 1) \leq \beta^{2-2p}|z|(\frac{3\beta}{2} + \frac{1}{2})$. \square

C. Formal proof

The proof given in Section IV-B is rather long and tedious. Also, we assumed (to avoid making it even more tedious!) no underflows. This is the typical case where formal proof is helpful.

The formal proof was done using Coq. The exact theorem is given below. Its counterpart in mathematical language is the following:

Theorem 2: Let β be the radix with $\beta > 1$. Let p be the number of digits with $p \geq 4$. Let \circ be any round-to-nearest mode.

Let a, b, x be floating-point numbers (either normal or subnormal!).

Let $z, p_h, p_l, u_h, u_l, t, z'$ be computed as in Algorithm 6. Let $v = \circ(p_l + u_l)$ be the intermediate result in the computation of z' .

Then we assume that z, p_h, u_h, v and z' must either be normal or zero. We also assume that the exponents of a and b are such that $e_a + e_x \geq e_{\min} + p - 1$ (so that the error of $a \times x$ is a FP).

Then

$$|z + z' - (ax + b)| \leq \left(\frac{3\beta}{2} + \frac{1}{2} \right) \beta^{2-2p} |z|.$$

□

Note that there is no requirement on the radix: the algorithm works in radix 2, 3, 4, 5, 10, 16, 43...

```

Variable bo : Fbound.
Variable radix : Z.
Variable p : nat.

Hypothesis pGivesBound : Zpos (vNum bo) = Zpower_nat radix p.
Hypothesis radixMoreThanOne : (1 < radix)%Z.
Hypothesis precisionGreaterThanOrEqualToFour : 4 <= p.

Variables a x b ph pl uh ul z t v w:float.

Hypothesis Fb: Fbounded bo b.
Hypothesis Fa: Fbounded bo a.
Hypothesis Fx: Fbounded bo x.

Hypothesis Nph: Fnormal radix bo ph \ / (FtoRradix ph=0).
Hypothesis Nuh: Fnormal radix bo uh \ / (FtoRradix uh=0).
Hypothesis Nz: Fnormal radix bo z \ / (FtoRradix z =0).
Hypothesis Nw: Fnormal radix bo w \ / (FtoRradix w =0).
Hypothesis Nv: Fnormal radix bo v \ / (FtoRradix v =0).

Hypothesis Exp1: (- dExp bo <= Fexp a+Fexp x)%Z.

Hypothesis zDef : Closest bo radix (a*x+b)%R z.
Hypothesis phDef: Closest bo radix (a*x)%R ph.

```

Hypothesis plDef: $(FtoRradix\ pl=a*x-ph)\%R.$

Hypothesis uhDef: Closest bo radix $(ph+b)\%R\ uh.$

Hypothesis ulDef: $(FtoRradix\ ul=ph+b-uh)\%R.$

Hypothesis tDef : Closest bo radix $(uh-z)\%R\ t.$

Hypothesis vDef : Closest bo radix $(pl+ul)\%R\ v.$

Hypothesis wDef : Closest bo radix $(t+v)\%R\ w.$

Theorem ErrFmaApprox:
 $(Rabs\ (z+w-(a*x+b))\ <= (3*radix/2+/2)*powerRZ\ radix\ (2-2*p)*Rabs\ z)\%R.$

D. Limits

Note that this does not mean that z' is nearly correct: it can be very wrong! The error can be as much as $z'/4$ in some cases (with $p = 4$, $a = 1001$, $x = 1010$, then $ax = 100001110$ and $b = 1101$) but it implies $|z'| \ll ulp(z)$ as in Figure 2.

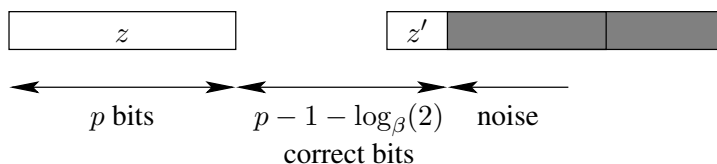


Fig. 2. Limited correctness of Algorithm 6

V. CONCLUSION

The cost of these algorithm is rather high, but it can be greatly improved if there are several FMAs available. The parallelization of Algorithm 4 is described in Figure 3 for two and three FMAs. The parallelization of Algorithm 6 is described in Figure 4.



Fig. 3. Parallelization of Algorithm 4 on two and three FMAs

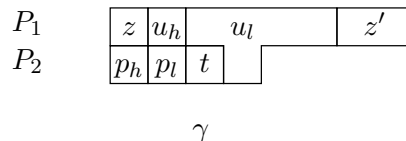


Fig. 4. Parallelization of Algorithm 6 on two FMAs

Then, the cost of the various algorithms is given in the following tabular:

Algorithm	ErrFmaNearest (Algo 4)	ErrFmaAppr (Algo 6)
Nb cycles	18	12
Nb cycles with 2 FMA	11	8
Nb cycles with 3 FMA	10	8

We have improved a previously obtained result on the computation of the (exact) error of an FMA, by providing a formal proof and showing that the algorithm actually works in a more general case than what was shown before. Also, we have provided and formally proved a faster algorithm that computes an approximate (yet, accurate) value of the error of an FMA. These algorithms may be used for building compensated algorithms (e.g., for polynomial evaluation) that use the FMA instruction. They might also be usable for performing accurate range reduction when computing some transcendentals. Also, this work illustrates the usefulness of formal proving in computer arithmetic: it allows one to really make sure that tedious and long proofs do not have flaws. It also makes it possible to check whether frequently made assumptions on the non-occurrence of possible intermediate underflows are necessary or not.

REFERENCES

- [1] E. Hokenek, R. K. Montoye, and P. W. Cook, “Second-generation RISC floating point with multiply-add fused,” *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1207–1213, Oct. 1990.
- [2] R. K. Montoye, E. Hokonek, and S. L. Runyan, “Design of the IBM RISC System/6000 floating-point execution unit,” *IBM Journal of Research and Development*, vol. 34, no. 1, pp. 59–70, 1990.
- [3] P. W. Markstein, “Computation of elementary functions on the IBM RISC System/6000 processor,” *IBM Journal of Research and Development*, vol. 34, no. 1, pp. 111–119, Jan. 1990.
- [4] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhauser Boston, 2009.
- [5] R.-C. Li, S. Boldo, and M. Daumas, “Theorems on efficient argument reduction,” in *Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH16)*, J.-C. Bajard and M. J. Schulte, Eds. IEEE Computer Society Press, Los Alamitos, CA, Jun. 2003, pp. 129–136.

- [6] R. M. Jessani and C. H. Olson, "The floating-point unit of the PowerPC 603e microprocessor," *IBM Journal of Research and Development*, vol. 40, no. 5, pp. 559–566, 1996.
- [7] G. Kane, *PA-RISC 2.0 Architecture*. Prentice Hall PTR, Upper Saddle River, NJ, 1995.
- [8] A. Kumar, "The HP PA-8000 RISC CPU," *IEEE Micro*, vol. 17, no. 2, pp. 27–32, March/April 1997.
- [9] M. Cornea, J. Harrison, and P. T. P. Tang, *Scientific Computing on Itanium[®]-based Systems*. Intel Press, Hillsboro, OR, 2002.
- [10] E. Quinell, E. E. Swartzlander, and C. Lemonds, "Floating-point fused multiply-add architectures," in *Forty-First Asilomar Conference on Signals, Systems, and Computers*, November 2007, pp. 331–337.
- [11] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, Aug. 2008, available at <http://ieeexplore.ieee.org/servlet/opac?punumber=4610933>.
- [12] O. Møller, "Quasi double-precision in floating-point addition," *BIT*, vol. 5, pp. 37–50, 1965.
- [13] D. Knuth, *The Art of Computer Programming*, 3rd ed. Addison-Wesley, Reading, MA, 1998, vol. 2.
- [14] T. J. Dekker, "A floating-point technique for extending the available precision," *Numerische Mathematik*, vol. 18, no. 3, pp. 224–242, 1971.
- [15] S. Boldo and M. Daumas, "Representable correcting terms for possibly underflowing floating point operations," in *Proceedings of the 16th Symposium on Computer Arithmetic*, J.-C. Bajard and M. Schulte, Eds. IEEE Computer Society Press, Los Alamitos, CA, 2003, pp. 79–86. [Online]. Available: <http://perso.ens-lyon.fr/marc.daumas/SoftArith/BolDau03a.pdf>
- [16] S. Boldo, "Pitfalls of a full floating-point proof: example on the formal proof of the Veltkamp/Dekker algorithms." in *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, ser. Lecture Notes in Computer Science, U. Furbach and N. Shankar, Eds., vol. 4130, 2006, pp. 52–66.
- [17] S. Boldo and J.-M. Muller, "Some functions computable with a fused-mac," in *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH-17)*. IEEE Computer Society Press, Los Alamitos, CA, Jun. 2005.
- [18] N. Louvet, "Algorithmes compensés en arithmétique flottante: Précision, validation, performances," Ph.D. dissertation, Université de Perpignan, Perpignan, France, Nov. 2007, in French.
- [19] T. Ogita, S. M. Rump, and S. Oishi, "Accurate sum and dot product," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 1955–1988, 2005.
- [20] S. Boldo, M. Daumas, C. Moreau-Finot, and L. Théry, "Computer validated proofs of a toolset for adaptable arithmetic," *École Normale Supérieure de Lyon, Tech. Rep.*, 2001, available at <http://arxiv.org/pdf/cs.MS/0107025>.