Exact and Approximated Error of the FMA

Sylvie Boldo and Jean-Michel Muller, *Senior Member, IEEE*

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, error-free transforms, error compensation, error of an FMA.

1 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 [2], [3]. 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 [4], [5], which explains why, on current chips offering such an instruction, there is no fully hardwired divider (see, e.g., [1]). An FMA also eases the design of an accurate range reduction algorithm for the trigonometric functions [6].

After the IBM RS/6000, FMA units were implemented in several general-purpose processors. Examples are the IBM PowerPC [7], the HP PA-8000 [8], [9], and the HP/Intel Itanium [10]. A survey on FMA architectures, along with suggestions for new architectures, is presented in [11].

The FMA instruction is included in the newly revised IEEE 754-2008 standard for floating-point arithmetic [12]. 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 [13], [14], [15], [16], [17] that (under some assumptions such as requiring rounding to nearest in the case of addition and square root, or assumptions on the radix; see [5] 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. Moreover, that error is computable using reasonably simple algorithms, some of which will be quickly recalled in Section 2. A natural question arises: Is there a similar property for the FMA operation?

We addressed this question in [18] 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 [19], [20] for building a fast *compensated* polynomial evaluation algorithm.

Nevertheless, the proofs of [18] 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[1] [21], and tried to get results as general as possible (for instance, we no longer require the radix to be two). This proof will be the first result presented in this paper, in Section 3.

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 4.

1.1 Notation

In a floating-point format of radix β, precision p, and extremal exponents $e_{\min}$ and $e_{\max}$, a finite floating-point (FP) number is a number for which there exists at least one representation (M, e) such that

$$x = M \cdot \beta^{e-p+1}, \tag{1}$$

where

- M is an integer of absolute value, less than or equal to $\beta^p - 1$. It is called the *integral significand* of the representation of x,
- e is an integer such that $e_{\min} \le e \le e_{\max}$, called the *exponent* of the representation of x.

The *significand* of the representation of x is the number $m = M \cdot \beta^{1-p}$, so that

$$x = m \cdot \beta^e.$$

1. http://coq.inria.fr/.

- *S. Boldo is with INRIA Saclay—Ile-de-France, Parc Orsay Universite—ZAC des Vignes; 4, rue Jacques Monod—Batiment N, 91893 Orsay Cedex, France. E-mail: Sylvie.Boldo@inria.fr.*
- *J.-M. Muller is with Ecole Normale Supérieure de Lyon, CNRS, INRIA, Universite Claude Bernard, Lyon 1, 46 allée d'Italie, 69364 Lyon Cedex 07, France. E-mail: Jean-Michel.Muller@ens-lyon.fr.*

Manuscript received 4 Aug. 2009; revised 16 Mar. 2010; accepted 19 Apr. 2010; published online 10 June 2010.
Recommended for acceptance by J. Bruguera, M. Cornea, and D. Das Sarma.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2009-08-0365.
Digital Object Identifier no. 10.1109/TC.2010.139.

The representation (M, e) is said *normalized*, if $\beta^{p-1} \le |M| \le \beta^p - 1$ (or, equivalently, $1 \le |m| < \beta$).

- When x has a normalized representation, that representation is unique, x is said *normal*, and we call integral significand, significand, and exponent of x the integral significand, significand, and exponent of its normalized representation.
- An FP number that has no normalized representation is said *subnormal*. If x is subnormal, then $|x| < \beta^{e_{\min}}$, and x has a unique representation, of exponent $e_{\min}$.

The smallest positive normal FP number is $\beta^{e_{\min}}$ and the smallest positive FP number is $\beta^{e_{\min}-p+1}$.

If x is normal and its normalized representation is (M, e), we define $\mathrm{ulp}(x)$ as β^{e-p+1}. If x is subnormal, we define $\mathrm{ulp}(x)$ as $\beta^{e_{\min}-p+1}$.

In the following, $\circ$ denotes the rounding operation under round-to-nearest mode. 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 integral significand is returned. Notice that all rounding operations defined by the IEEE 754-2008 standard are monotonic: if $a \le b$ then $\circ(a) \le \circ(b)$. We will use this property in our proofs.

We will also frequently use Sterbenz's Theorem:

Theorem 1 (Sterbenz [24]). *In a radix-β floating-point system with subnormal numbers available, if x and y are finite floating-point numbers such that*

$$\frac{y}{2} \le x \le 2y,$$

then $x - y$ is exactly representable.

2 Basic Operations

Let us now present some basic algorithms, called *error-free transforms* by Rump [22], that allow one, under some conditions, to compute the error of a floating-point addition or multiplication exactly.

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. There cannot be any *hidden* overflow in these algorithms: one will always get an infinity as result, if an overflow occurs at any point.

2.1 Algorithm Fast2Sum

Fast2Sum was introduced in a paper by Dekker [15], [17] in 1971. Assume that $\circ$ is round-to-nearest, and that $\beta \le 3$. Let a and b be floating-point numbers such that the exponent of a, noted e_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 $|a| \ge |b|$ implies $e_a \ge e_b$, the needed requirement for this algorithm.

Algorithm 1 (Fast2Sum(a, b)):

```
$s = \circ(a+b)$
$t = \circ(s-a)$
$\rho = \circ(b-t)$
```

2.2 Algorithm 2Sum

Fast2Sum only requires three floating-points additions, and yet it has two drawbacks: first, it does not always work in radices larger than three (in particular, 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 [14]:

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

```
$s = \circ(a+b)$
$\hat{a} = \circ(s-b)$
$\hat{b} = \circ(s-\hat{a})$
$\epsilon_a = \circ(a-\hat{a})$
$\epsilon_b = \circ(b-\hat{b})$
$\rho = \circ(\epsilon_a+\epsilon_b)$
```

Knuth [14] 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. [23] showed that in radix 2, this result still holds in the presence of underflow.

2.3 Algorithm Fast2Mult

If no FMA instruction is available, there exists an algorithm, due to Dekker, that computes the error of an FP multiplication using 17 FP operations [15] (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)):

```
$t = \circ(a \cdot b)$
$\rho = \circ(a \cdot b - t)$
```

Let e_a and a_b be the floating-point exponents of a and b. If $e_a + e_b \ge e_{\min} + p - 1$, 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 $e_a + e_b \ge e_{\min} + p - 1$ cannot be avoided: if it is not satisfied, then there are cases when $t - a \cdot b$ is not an FP number.

3 Exact Error of the FMA

3.1 Algorithm

We presented in [18] the following algorithm to compute the exact error of an FMA. The input values are three FP numbers a, x, and y. The output values are r_1, r_2, and r_3.

Algorithm 4 (ErrFma):

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

Property 1 (ErrFma_correctness). *Assuming radix 2, round-to-nearest and no underflows/overflows, we showed in [18] that Algorithm 4 satisfies:*

- $ax + y = r_1 + r_2 + r_3$ *exactly,*
- $|r_2 + r_3| \le \frac{1}{2}\mathrm{ulp}(r_1)$, *and*
- $|r_3| \le \frac{1}{2}\mathrm{ulp}(r_2)$.

Using property 1, 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 straightforward to get it:

Algorithm 5 (ErrFmaNearest):

```
        $r_1 = \circ(ax + y)$
  $(u_1, u_2) = \mathrm{Fast2Mult}(a, x)$
  $(\alpha_1, \alpha_2) = \mathrm{2Sum}(y, u_2)$
  $(\beta_1, \beta_2) = \mathrm{2Sum}(u_1, \alpha_1)$
        $\gamma = \circ(\circ(\beta_1 - r_1) + \beta_2)$
       $r_2 = \circ(\gamma + \alpha_2)$
```

From the results of [18], we easily deduce that

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

3.2 Formal proof

Nevertheless, the proofs of [18] were only in radix 2, and were only pen-and-paper proofs. As the proof is complex and has many subcases (for example, $\beta_2 = 0$ or not), and to increase the trust in this algorithm, we have formally proved Algorithm 4, which 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 in Fig. 1. Its counterpart in mathematical language is the following:

Theorem 2. *Let p be the number of digits with $p \ge 3$. Let β be the radix with $\beta \ge 2$. We assume that β is even, and that $\circ$ is any consistent round-to-nearest mode. This means that the rounding must be a rounding to nearest, but done in a consistent 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, and x be floating-point numbers (either normal or subnormal).

Let r_1, u_1, u_2, α_1, α_2, β_1, β_2, and γ be computed as in Algorithm 4.

Then we assume a few nonunderflow hypotheses:

- either $\alpha_1 = 0$ or $\beta^{e_{\min}+1} \le |\alpha_1|$
- either $u_1 = 0$ or $\beta^{e_{\min}+1} \le |u_1|$
- either $\beta_1 = 0$ or $\beta^{e_{\min}+2} \le |\beta_1|$

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

Hypothesis pGivesBound : Zpos (vNum bo)
             = Zpower_nat radix p.
Hypothesis radixMoreThanOne : (1 < radix)%Z.
Hypothesis precisionGreaterThanOne : 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) -> (P r2 f2)
     -> (r1=r2)%R -> (FtoRradix f1=f2)%R.

Variable a x y r1 u1 u2 al1 al2 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 Cal1: Fcanonic radix bo al1.
Hypothesis Cu1 : Fcanonic radix bo u1.
Hypothesis Exp1: (- dExp bo < Fexp al1)%Z
                    \/ (FtoRradix al1=0)%R.
Hypothesis Exp2: (- dExp bo < Fexp u1)%Z
                    \/ (FtoRradix u1=0)%R.
Hypothesis Exp3: (- dExp bo+1 < Fexp be1)%Z
                    \/ (FtoRradix be1=0)%R.
Hypothesis Exp4: (Fnormal radix bo r1)
                    \/ (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 al1Def:(Closest bo radix (y+u2)%R al1).
Hypothesis al2Def:(FtoRradix al2=y+u2-al1)%R.
Hypothesis be2Def:(FtoRradix be2=u1+al1-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+al1)%R be1).

Theorem FmaErr: (a*x+y=r1+ga+al2)%R.

Theorem Fma_FTS: (exists ga_e:float, exists al2_e:float,
               (FtoRradix ga_e=ga)%R
                /\ (FtoRradix al2_e=al2)%R
                /\ (Fbounded bo ga_e)
                /\ (Fbounded bo al2_e)
                /\ (Fexp al2_e <= Fexp ga_e)%Z).
```

Fig. 1. Coq formulation of Theorem 2.

- either $r_1 = 0$ or r_1 is normal, and
- the exponents of a and b are such that $e_a + e_x \ge e_{\min} + p - 1$ (so the error when computing ax, namely $ax - \circ(ax)$, is an FP number).

Then

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

Note that there is no requirement on the radix except that it should be even: for instance, the algorithm works in radices 2, 10, 16. This limit is due to the fact that $\frac{1}{2}\mathrm{ulp}(f)$ is considered a floating-point number, which greatly simplifies the proof. 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, was the SETUN computer, built in the USSR in the late 1950s [25].

4 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

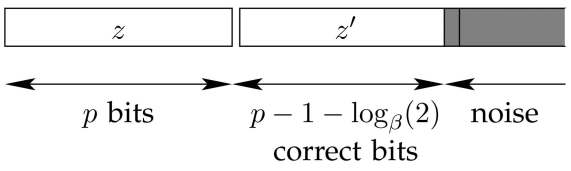

Fig. 2. Correct bits of Algorithm 6.

operations. And yet, for many applications such as compensated polynomial evaluation [19], [20], 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.

4.1 Algorithm

We make no assumption on the radix β (except, of course, that it is an integer larger than or equal to two). We assume that the precision p is larger than or equal to four. Even more general than previously, $\circ$ is any round-to-nearest: not even consistence is needed here! Therefore, it works in rounding to nearest, ties to even and in rounding to nearest, ties away from zero, and it is even possible to switch between these two rounding modes during the calculation.

Algorithm 6 (ErrFmaAppr):

```
$z = \circ(ax + b)$
$(p_h, p_l) = \mathrm{Fast2Mult}(a, x)$
$(u_h, u_l) = \mathrm{2Sum}(b, p_h)$
$t = \circ(u_h - z)$
$z' = \circ(t + \circ(p_l + u_l))$
```

We are going to prove

Property 2 (ErrFmaAppr_correctness).

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

Property 2 implies that $|z + z' - (ax + b)| < 2 \cdot \beta^{3-2p} \cdot |z|$, therefore, we have at least $p - 1 - \log_\beta(2)$ correct digits following z, as shown in Fig. 2.

4.2 Proof

We assume there is neither Underflow nor Overflow, and that the working precision p is larger than or equal to four. Note that concerning underflow, this assumption is just here for simplifying the proof (see Section 4.3). We proved that, if $f = \circ(r)$ and f is normal, then $|f| \leq \frac{|r|}{1-\beta^{1-p}/2}$ and $|r| \leq |f|(1 + \beta^{1-p}/2)$.

4.2.1 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, in that case, $ax \geq -b$, 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 nonpositive. We now consider two subcases depending on whether $u_h = \circ(p_h + b)$ is the result of a significant cancellation, or not.

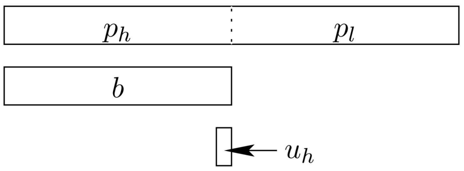

Fig. 3. Graphical representation of the respective values of p_h, p_l, b, and u_h, when $\frac{|p_h+b|}{4} < |p_l|$.

Assuming $|p_l| \leq \frac{|p_h+b|}{4}$: As we are going to see, this assumption of no or small cancellation will suffice to guarantee that Theorem 1 can be applied to the computation of t. We have,

$$\begin{aligned}|z| &\leq \frac{|ax + b|}{1 - \beta^{1-p}/2}\\ &\leq \frac{|p_h + b| + |p_l|}{1 - \beta^{1-p}/2}\\ &\leq \frac{5}{4} \cdot \frac{|p_h + b|}{1 - \beta^{1-p}/2}\\ &\leq |u_h| \cdot \frac{5}{4} \cdot \frac{1 + \beta^{1-p}/2}{1 - \beta^{1-p}/2}\\ &\leq 2 \cdot |u_h|,\end{aligned}$$

as $p \geq 4$.

Moreover,

$$\begin{aligned}|p_h + b| &= |p_h + p_l + b - p_l|\\ &\leq |p_h + p_l + b| + |p_l|\\ &\leq |ax + b| + \frac{|p_h + b|}{4},\end{aligned}$$

so $|p_h + b| \leq \frac{4}{3} \cdot |ax + b|$. Therefore,

$$\begin{aligned}|u_h| &\leq \frac{|p_h + b|}{1 - \beta^{1-p}/2}\\ &\leq \frac{4}{3} \cdot \frac{|ax + b|}{1 - \beta^{1-p}/2}\\ &\leq |z| \cdot \frac{4}{3} \cdot \frac{1 + \beta^{1-p}/2}{1 - \beta^{1-p}/2}\\ &\leq 2|z|,\end{aligned}$$

as $p \geq 4$.

Therefore, from Theorem 1, $|u_h| - |z| = \pm(u_h - z)$ is representable and $u_h - z$ is computed exactly.

Assuming $\frac{|p_h+b|}{4} < |p_l|$: This assumption means a significant cancellation during the computation of $u_h = \circ(p_h + b)$ as shown in Fig. 3. It, therefore, means that $p_h \approx -b$. This also implies that $p_l \neq 0$, therefore the exponent of p_h cannot be the minimal exponent (otherwise, $p_h + p_l$ would fit in one FP number only):

$$e_{p_h} > e_{\min}.$$

Furthermore, since a is a multiple of $\mathrm{ulp}(a) = \beta^{e_a-p+1}$ and x is a multiple of β^{e_x-p+1}, we deduce that ax (and, therefore, p_l) is a multiple of $\beta^{e_a+e_x-2p+2}$. Since p_l is nonzero, its absolute value is at least $\beta^{e_a+e_x-2p+2}$. From $|p_l| \leq 1/2\mathrm{ulp}(p_h)$ we immediately deduce

$$e_{p_h} > e_a + e_x - p + 1.$$

Moreover,

$$|p_h + b| < 4 \cdot |p_l| \le 2 \cdot \mathrm{ulp}(p_h) = 2\beta^{e_{p_h}-p+1}.$$

Also,

$$|b| \ge |p_h| - |p_h + b| \ge |p_h| - 2 \cdot \beta^{e_{p_h}-p+1} \ge |p_h| - 2 \cdot |p_h| \cdot \beta^{-p},$$

therefore (since we assumed $p \ge 4$),

$$|b| \ge \frac{1}{2} \cdot |p_h|,$$

and $e_{p_h} - 1 \le e_b$. We easily prove that $|b| \le 2|p_h|$. Therefore, from Theorem 1, the computation of $p_h + b$ is exact, i.e.,

$$u_h = p_h + b,$$

and the result is a multiple of $\beta^{e_{p_h}-p}$.

Furthermore, since ax is a multiple of $\beta^{e_a+e_x-2p+2}$ and $e_b \ge e_{p_h} - 1 \ge e_a + e_x - p + 1$ (which implies that b is a multiple of $\beta^{e_a+e_x-2p+2}$), we find that $ax + b$ is a multiple of $\beta^{e_a+e_x-2p+2}$. So, $z = \circ(ax + b)$ is a multiple of $\beta^{e_a+e_x-2p+2}$.

Finally, $u_h - z$ is a multiple of $\beta^{e_a+e_x-2p+2}$, say $u_h - z = T \cdot \beta^{e_a+e_x-2p+2}$. To show that $t = u_h - z$ exactly, it only remains to show that $u_h - z$ is a floating-point number. To that purpose, we show that $|T| \le \beta^p - 1$. We have,

$$\begin{aligned}|T| &= |u_h - z| \cdot \beta^{-e_a-e_x+2p-2} = |p_h + b - z| \cdot \beta^{-e_a-e_x+2p-2}\\ &\le (|p_l| + |ax + b - z|)\beta^{-e_a-e_x+2p-2}\\ &\le \frac{1}{2}\left(\beta^{e_{p_h}-e_a-e_x+p-1} + \beta^{e_z-e_a-e_x+p-1}\right).\end{aligned}$$

Moreover, $e_z < e_{p_h}$ as

$$\begin{aligned}|ax + b| &\le |p_h + b| + |p_l|\\ &\le 5 \cdot |p_l|\\ &\le \frac{5}{2}\beta^{e_{p_h}-p+1},\end{aligned}$$

so $|z| \le 3\beta^{e_{p_h}-p+1}$.

Therefore,

$$|T| = |u_h - z| \cdot \beta^{-e_a-e_x+2p-2} < \beta^{e_{p_h}-e_a-e_x+p-1}.$$

It remains to be proved that $e_{p_h} - e_a - e_x + p - 1 \le p$, i.e., that $e_{p_h} \le e_a + a_x + 1$, which is easy, since

$$\begin{aligned}|ax| &\le \left(\beta - \frac{1}{\beta^{p-1}}\right)^2 \cdot \beta^{e_a+e_x}\\ &\le \left(\beta - \frac{1}{\beta^{p-1}}\right) \cdot \beta^{e_a+e_x+1},\end{aligned}$$

which implies

$$|p_h| \le \left(\beta - \frac{1}{\beta^{p-1}}\right) \cdot \beta^{e_a+e_x+1}.$$

We have ended the proof of the fact that the computation of t is exact. We now separately consider the two subcases $u_h = p_h + b$ and $u_h \ne p_h + b$.

4.2.2 When $u_h = p_h + b$

Property 4. *When $u_h = p_h + b$, Theorem 2 holds.*

This is the easiest case and is represented by Fig. 4. In fact, the hypothesis means that $u_l = 0$ so that $z' = \circ(t + p_l)$.

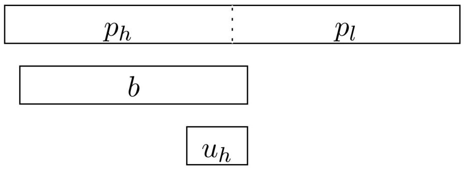

Fig. 4. Graphical representation of the respective values of p_h, p_l, b, and u_h when $u_h = p_h + b$.

Therefore,

$$\begin{aligned}z + z' - (ax + b) &= u_h - t + \circ(t + p_l) - p_h - p_l - b\\ &= \circ(t + p_l) - t - p_l,\end{aligned}$$

and $|z + z' - (ax + b)| \le \frac{1}{2}\mathrm{ulp}(z')$.

Moreover,

$$|z'| \le |z - (ax + b)| + |\circ(t + p_l) - t - p_l|,$$

so

$$|z'| - \frac{1}{2}\mathrm{ulp}(z') \le \frac{1}{2}\mathrm{ulp}(z).$$

This easily implies that $|z'| \le \mathrm{ulp}(z)$, and that

$$\begin{aligned}|z + z' - (ax + b)| &\le \frac{1}{2}\mathrm{ulp}(z') \le \frac{1}{2}\beta^{1-p}|z'|\\ &\le \frac{1}{2}\beta^{2-2p}|z|\\ &\le \left(\frac{3\beta}{2} + \frac{1}{2}\right)\beta^{2-2p}|z|.\end{aligned}$$

4.2.3 When $u_h \ne p_h + b$

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. $e_{p_h} \le e_{u_h} + 1$.

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

Property 6. $e_{u_h} \le e_z + 1$.

We have

$$\begin{aligned}|u_h - z| &= |(u_h - (p_h + b)) + (p_h - ax) + (ax + b - z)|\\ &\le \frac{1}{2}\mathrm{ulp}(u_h) + \frac{1}{2}\mathrm{ulp}(p_h) + \frac{1}{2}\mathrm{ulp}(z)\\ &\le \frac{\beta + 1}{2}\mathrm{ulp}(u_h) + \frac{1}{2}\mathrm{ulp}(z)\\ &\le \frac{\beta + 1}{2} \cdot |u_h| \cdot \beta^{1-p} + \frac{1}{2} \cdot |z| \cdot \beta^{1-p},\end{aligned}$$

using the preceding property.

Therefore, $|u_h| \cdot (1 - \frac{\beta+1}{2}\beta^{1-p}) \le |z|(1 + \frac{\beta^{1-p}}{2})$ and $|u_h| \le \beta \cdot |z|$ as $p \ge 4$.

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

Let us assume that these equalities hold.

To prove the absurdity, we will prove that $|z| \geq \beta^{e_z+1}$, which is impossible.

First, $|ax+b| \geq |ax| - |b|$. As $p_h = \circ(ax)$, we know that

$$|ax| \geq \left(\beta^{p-1} - \frac{1}{2\beta}\right)\beta^{e_{p_h}-p+1},$$

since the smallest possible real number to be rounded into a floating-point number with exponent e_{p_h} is the smallest floating-point number with this exponent, namely $\beta^{e_{p_h}}$, minus half the difference between this number and its predecessor.

Furthermore,

$$|b| \leq (\beta^p - 1)\beta^{e_b-p+1}.$$

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

Then,

$$\begin{aligned}|ax+b| &\geq |ax| - |b| \\ &\geq \left(\beta^{p-1} - \frac{1}{2\beta}\right)\cdot\beta^{e_{p_h}-p+1} - (\beta^p-1)\cdot\beta^{e_b-p+1} \\ &\geq \left(\beta^{p-1} - \frac{1}{2\beta}\right)\cdot\beta^{e_z-p+3} - (\beta^p-1)\cdot\beta^{e_z-p+1} \\ &= \beta^{e_z+1}(\beta-1) - \beta^{e_z-p+1}\left(\frac{\beta}{2}-1\right).\end{aligned}$$

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

$$2\cdot\beta^{e_z+1} - \beta^{e_z-p+1}\left(\frac{\beta}{2}-1\right) \geq \beta^{e_z+1}.$$

In all cases, we have

$$|z| \geq \beta^{e_z+1},$$

which is impossible.

Property 8. $|u_h - z| \leq (\beta+1)\cdot\mathrm{ulp}(z)$.

From the last three 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

$$e_{u_h} \leq e_z.$$

This means that $\beta^{e_{u_h}} + \beta^{e_{p_h}} \leq 2\cdot\beta^{e_z+1}$ in all cases.

Then,

$$\begin{aligned}|u_h - z| &= |(u_h - (p_h+b)) + (p_h - ax) + (ax+b-z)| \\ &\leq \frac{1}{2}\mathrm{ulp}(u_h) + \frac{1}{2}\mathrm{ulp}(p_h) + \frac{1}{2}\mathrm{ulp}(z),\end{aligned}$$

so that

$$\begin{aligned}|u_h - z| &\leq \frac{1}{2}\cdot\beta^{e_{u_h}-p+1} + \frac{1}{2}\cdot\beta^{e_{p_h}-p+1} + \frac{1}{2}\cdot\beta^{e_z-p+1} \\ &\leq \frac{1}{2}\cdot\beta^{-p+1}(\beta^{e_{u_h}} + \beta^{e_{p_h}}) + \frac{1}{2}\cdot\beta^{e_z-p+1} \\ &\leq (\beta+1)\cdot\beta^{e_z-p+1}.\end{aligned}$$

Property 9. *When $u_h \neq p_h + b$, Theorem 2 holds.*

We first bound $|u_l + p_l| \leq \beta^{e_z-p}$, which gives

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

We then bound $|t + \circ(p_l + u_l)| \leq (2\beta+1)\beta^{e_z}$, which gives

$$|z'| \leq (2\beta+1)\beta^{e_z}.$$

Moreover,

$$\begin{aligned}z + z' - (ax+b) &= u_h - t + z' - p_h - p_l - b \\ &= z' - t - p_l - u_l,\end{aligned}$$

so the error only comes from the computations inside z' that occur on numbers that are small compared to z.

Therefore,

$$\begin{aligned}|z + z' - (ax+b)| &\leq \frac{1}{2}\mathrm{ulp}(\circ(p_l+u_l)) + \frac{1}{2}\mathrm{ulp}(z') \\ &\leq \frac{1}{2}\beta^{1-p}\beta^{e_z-p+1}(3\beta+1) \\ &\leq \beta^{2-2p}\cdot|z|\cdot\left(\frac{3\beta}{2}+\frac{1}{2}\right).\end{aligned}$$

4.3 Formal Proof

The proof given in Section 4.2 is rather long and tedious. Also, we assumed no underflows, to avoid making it even more tedious. This is the typical case when formal proof is helpful.

The formal proof was done using Coq. The exact theorem is given in Fig. 5.

Its counterpart in mathematical language is the following:

Theorem 3. *Let β be the radix with $\beta \geq 2$. Let p be the significand, with $p \geq 4$. Let $\circ$ be any round-to-nearest mode.*

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

Let z, p_h, p_l, u_h, u_l, t, and 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 x are such that $e_a + e_x \geq e_{\min} + p - 1$ (so that the error of $a \times x$ is an FP).

Then

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

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

4.4 Limits

Note that Theorem 3 does not mean that z' is nearly correct. Indeed, it can be very wrong! The error can be as much as

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

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

Fig. 5. Coq formulation of Theorem 3.

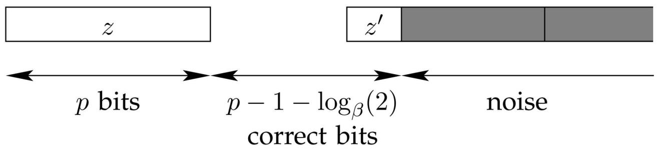

Fig. 6. Limited correctness of Algorithm 6.

$z'/4$ in some cases (with $p = 4$, $a = 1001$, $x = 1010$, then $ax = 100001110$ and $b = 1101$) but it implies $|z'| \ll \mathrm{ulp}(z)$ as exemplified in Fig. 6.

5 Conclusion

The cost of these algorithms is rather high, but it can be greatly improved if there are several FMAs available. A possible parallelization of Algorithm 4 is described in Fig. 7 for two and three FMAs. A possible parallelization of Algorithm 6 is described in Fig. 8.

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

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

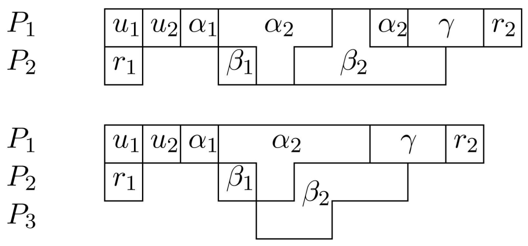
P_1
P_2
u_1 u_2 α_1 α_2 α_2 γ r_2
r_1 β_1 β_2
P_1
P_2
P_3
u_1 u_2 α_1 α_2 γ r_2
r_1 β_1 β_2

Fig. 7. Parallelization of Algorithm 4 on two and three FMAs.

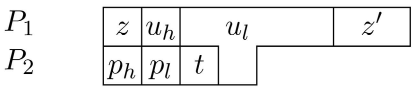

Fig. 8. Parallelization of Algorithm 6 on two FMAs.

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 such as the nonoccurrence of possible intermediate underflows are necessary or not.

Acknowledgments

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

References

[1] R.C. Agarwal, F.G. Gustavson, and M.S. Schmookler, "Series Approximation Methods for Divide and Square Root in the POWER3 Processor," *Proc. 14th IEEE Symp. Computer Arithmetic (ARITH-14 '99)*, pp. 116-123, Apr. 1999.

[2] E. Hokenek, R.K. Montoye, and P.W. Cook, "Second-Generation RISC Floating Point with Multiply-Add Fused," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1207-1213, Oct. 1990.

[3] R.K. Montoye, E. Hokonek, and S.L. Runyan, "Design of the IBM RISC System/6000 Floating-Point Execution Unit," *IBM J. Research and Development*, vol. 34, no. 1, pp. 59-70, 1990.

[4] P.W. Markstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor," *IBM J. Research and Development*, vol. 34, no. 1, pp. 111-119, Jan. 1990.

[5] 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, 2009.

[6] R.-C. Li, S. Boldo, and M. Daumas, "Theorems on Efficient Argument Reduction," *Proc. 16th IEEE Symp. Computer Arithmetic (ARITH-16 '03)*, pp. 129-136, June 2003.

[7] R.M. Jessani and C.H. Olson, "The Floating-Point Unit of the PowerPC 603e Microprocessor," *IBM J. Research and Development*, vol. 40, no. 5, pp. 559-566, 1996.

[8] G. Kane, *PA-RISC 2.0 Architecture.* Prentice Hall PTR, 1995.

[9] A. Kumar, "The HP PA-8000 RISC CPU," *IEEE Micro*, vol. 17, no. 2, pp. 27-32, Mar./Apr. 1997.

[10] M. Cornea, J. Harrison, and P.T.P. Tang, *Scientific Computing on Itanium-Based Systems.* Intel Press, 2002.

[11] E. Quinnell, E.E. Swartzlander, and C. Lemonds, "Floating-Point Fused Multiply-Add Architectures" *Proc. 41st Asilomar Conf. Signals, Systems, and Computers*, pp. 331-337, Nov. 2007.

[12] IEEE CS, *IEEE Standard for Floating-Point Arithmetic,* IEEE Standard 754-2008, http://ieeexplore.ieee.org/servlet/opac?punumber=4610933, Aug. 2008.

[13] O. Møller, "Quasi Double-Precision in Floating-Point Addition," *BIT Numerical Math.,* vol. 5, pp. 37-50, 1965.

[14] D. Knuth, *The Art of Computer Programming,* third ed. vol. 2. Addison-Wesley, 1998.

[15] T.J. Dekker, "A Floating-Point Technique for Extending the Available Precision," *Numerische Mathematik,* vol. 18, no. 3, pp. 224-242, 1971.

[16] S. Boldo and M. Daumas, "Representable Correcting Terms for Possibly Underflowing Floating Point Operations," *Proc. 16th Symp. Computer Arithmetic (ARITH-16 '03),* pp. 79-86, http://perso.ens-lyon.fr/marc.daumas/SoftArith/BolDau03a.pdf, 2003.

[17] S. Boldo, "Pitfalls of a Full Floating-Point Proof: Example on the Formal Proof of the Veltkamp/Dekker algorithms," *Proc. Third Int'l Joint Conf. Automated Reasoning,* pp. 52-66, 2006.

[18] S. Boldo and J.-M. Muller, "Some Functions Computable with a Fused-Mac," *Proc. 17th IEEE Symp. Computer Arithmetic (ARITH-17 '05),* June 2005.

[19] N. Louvet, "Algorithmes Compensés en Arithmétique Flottante: Précision, Validation, Performances," PhD dissertation, Univ. de Perpignan, Nov. 2007.

[20] S. Graillat, P. Langlois, and N. Louvet, "Improving the Compensated Horner Scheme with a Fused Multiply and Add," *Proc. 21st Ann. ACM Symp. Applied Computing (ACM '06),* pp. 1323-1327, 2006.

[21] Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Springer Verlag, 2004.

[22] T. Ogita, S.M. Rump, and S. Oishi, "Accurate Sum and Dot Product," *SIAM J. Scientific Computing,* vol. 26, no. 6, pp. 1955-1988, 2005.

[23] S. Boldo, M. Daumas, C. Moreau-Finot, and L. Théry, "Computer Validated Proofs of a Toolset for Adaptable Arithmetic," École Normale Supérieure de Lyon, technical report, http://arxiv.org/pdf/cs.MS/0107025, 2001.

[24] P.H. Sterbenz, *Floating-Point Computation.* Prentice-Hall, 1974.

[25] W. Ware et al., "Soviet Computer Technology," *Comm. ACM,* vol. 3, pp. 131-166, Mar. 1960.

Sylvie Boldo received the MSc and PhD degrees in computer science from the École Normale Supérieure de Lyon, France, in 2001 and 2005, respectively. She is now a researcher for the INRIA Saclay—Île-de-France in the ProVal team (Orsay, France), whose research focuses on formal certification of programs. Her main research interests include floating-point arithmetic, formal methods, and formal verification of numerical programs.

Jean-Michel Muller received the PhD degree in 1985 from the Institut National Polytechnique de Grenoble. He is directeur de recherches (senior researcher) at CNRS, France, and the former head of the LIP laboratory (LIP is a joint laboratory of CNRS, Ecole Normale Supérieure de Lyon, INRIA and Université Claude Bernard Lyon 1). His research interests are in computer arithmetic. He was coprogram chair of the 13th IEEE Symposium on Computer Arithmetic (Asilomar, June 1997), general chair of SCAN 1997 (Lyon, France, Sept. 1997), and general chair of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia, Apr. 1999). He is the author of several books, including *Elementary Functions, Algorithms and Implementation* (second ed., Birkhäuser Boston, 2006), and he coordinated the writing of the *Handbook of Floating-Point Arithmetic* (Birkhäuser Boston, 2010). He served as associate editor of the *IEEE Transactions on Computers* from 1996 to 2000. He is a senior member of the IEEE.