Emulation of 3Sum, 4Sum, the FMA and the FD2 instructions in rounded-to-nearest floating-point arithmetic

Stef Graillat, Jean-Michel Muller

Emulation of 3Sum, 4Sum, the FMA and the FD2 instructions in rounded-to-nearest floating-point arithmetic

Stef Graillat* Jean-Michel Muller†

June 25, 2024

Abstract

We give high level algorithms that make it possible to compute the rounded-to-nearest sum of 3 or 4 floating-point numbers in binary floating-point arithmetic. They can easily be adapted to emulate the FMA ($ab + c$ correctly rounded) and FD2 ($ab + cd$ correctly rounded) instructions.

Keywords. Floating-point arithmetic, Sum3, Sum4, FMA, fused multiply-add, FD2, fused dot-product, Error-free transforms, double-word arithmetic.

1 Introduction

Recently, Graillat and Muller presented an "high-level" algorithm that emulates the fused multiply-add (FMA) instruction in binary floating-point (FP) arithmetic [6]. By "high level" we mean that their algorithm uses only floating-point instructions, and with only one rounding function (the default one, round to nearest, ties-to-even): there is no need to use masks and/or integer operations that would manipulate the binary representation of the FP numbers. Since the FMA can be viewed as a particular instance of the sum of three FP numbers ($ab + c$ is equal to $\sigma_h + \sigma_\ell + c$ where σ_h and σ_ℓ are obtained from a and b by the means of a "2Mult" error-free transformation), a natural question is whether their algorithm can be adapted to the sum of three arbitrary FP numbers. Lauter [14] shows that for the IEEE754 binary formats, the correctly-rounded sum of three FP numbers can be computed using 128-bit integer operations. As we will see, using only FP operations, the adaptation of the FMA algorithm presented in [6] is straightforward and poses no particular difficulty. A more interesting question is whether this work can be extended to the sum of *four* arbitrary FP numbers. This would allow (still using "2Mult" transformations), to emulate the FD2 operation ($ab + cd$ correctly rounded), which is of significant interest,

*Sorbonne Université, CNRS, LIP6, Paris, France, stef.graillat@lip6.fr
†CNRS, LIP, Université de Lyon, Lyon, France, jean-michel.muller@ens-lyon.fr

since this operation is central in complex multiplication and more generally in accurate complex arithmetic, Fourier transforms, Givens rotations, etc. (examples are given in [8]).

In this paper, we assume a binary, precision-p floating-point (FP) arithmetic, with an unbounded exponent range. This means that the results presented here apply to conventional binary IEEE 754 [1] floating-point arithmetic provided that underflow and overflow do not occur. A FP number is therefore 0 or a number of the form $x = M \cdot 2^e$, where $M, e \in \mathbb{Z}$ and $2^{p-1} \leq |M| \leq 2^p - 1$. The number e is the *floating-point exponent* of x. We assume that the rounding function is *round-to-nearest, ties-to-even*, noted RN, which is the default in IEEE 754 arithmetic [1, 2]. The *unit round-off* is $u = 2^{-p}$. This number u bounds the relative error due to rounding: when an arithmetic operation $x \top y$ is performed (with $\top \in \{+, -, \times, \div\}$) , the computed result $z = \mathrm{RN}(x \top y)$ satisfies

$$(1 - u) \cdot |(x \top y)| \leq |z| = |\mathrm{RN}(x \top y)| \leq (1 + u) \cdot |(x \top y)|. \tag{1}$$

The *unit in the last place* (ulp) or a real number x is the number

$$\mathrm{ulp}(x) = \begin{cases} 0 & \text{if } x = 0, \\ 2^{\lfloor \log_2 |x| \rfloor - p + 1} & \text{otherwise.} \end{cases}$$

When x is not a FP number, $\mathrm{ulp}(x)$ is equal to the distance between the two FP numbers straddling $|x|$, and when x is a FP number, it is equal to the distance between $|x|$ and the smallest FP number larger than $|x|$.

The errors of "atomic calculations" (such as the computation of $\exp(x)$ or $\sin(x)$) are often expressed in ulps (see for instance [5]), whereas the relative errors of more complex calculations are often expressed as a function of u.

A pair (x_h, x_ℓ) of FP numbers is a *double-word*[1] (DW) number representing a real number x if $x = x_h + x_\ell$ and $x_h = \mathrm{RN}(x)$. Some algorithms for manipulating double-word numbers, presented and analyzed in [10], will serve as a starting point for the algorithms presented in this paper.

1.1 Some classical results of floating-point arithmetic used in this paper

In this section, we just briefly present the results needed in the sequel of the paper. More detailed presentations and proofs can be found in [17].

1.1.1 Sterbenz's theorem

Theorem 1.1 below is frequently called "Sterbenz Lemma" in the literature. However, we feel that its usefulness in error analysis fully justifies the name "theorem". For instance, the most accurate algorithms for elementary function evaluation, and the proof of the double-word algorithms presented in [10] heavily rely on Sterbenz's theorem.

[1] We frequently see the name "double double" in the literature. We prefer "double word" because there is no reason to systematically assume that the underlying format is double precision/binary64.

Theorem 1.1 (Sterbenz Theorem [21]). *Let a, b be FP numbers. If $\frac{a}{2} \leq b \leq 2a$ then $a - b$ is an FP number. In particular, this implies that the subtraction $a - b$ will be performed exactly in FP arithmetic.*

1.1.2 The Fast2Sum and 2Sum algorithms

One easily shows that the error of a rounded-to-nearest floating-point addition is a FP number. Interestingly enough, that error can be computed using rather simple algorithms, due to Moller, Dekker, and Knuth [16, 4, 12], and qualified as *Error-Free Transformations* (EFT) by Ogita, Rump, and Oishi [20]

Algorithm 1 – Fast2Sum(a, b). The Fast2Sum algorithm [4].

$s \leftarrow \text{RN}(a + b)$
$z \leftarrow \text{RN}(s - a)$
$t \leftarrow \text{RN}(b - z)$
return (s, t)

In Algorithm 1, if the floating-point exponents e_a and e_b of a and b are such that $e_a \geq e_b$ then t is the error of the floating-point addition $\text{RN}(a + b)$ (i.e., the double word (s, t) is exactly equal to $a + b$). The condition on the exponents may be difficult to check, but it is satisfied if $|a| \geq |b|$.

Algorithm 2 – 2Sum(a, b). The 2Sum algorithm [16, 12].

$s \leftarrow \text{RN}(a + b)$
$a' \leftarrow \text{RN}(s - b)$
$b' \leftarrow \text{RN}(s - a')$
$\delta_a \leftarrow \text{RN}(a - a')$
$\delta_b \leftarrow \text{RN}(b - b')$
$t \leftarrow \text{RN}(\delta_a + \delta_b)$
return (s, t)

In Algorithm 2, for all FP numbers a and b, t is the error of the floating-point addition $\text{RN}(a + b)$. It is not necessary to test if $|a| \geq |b|$.

1.1.3 The 2MultFMA and Dekker-Veltkamp multiplication algorithms

The error of an FP multiplication is an FP number. As in the case of the addition, it can be computed using fairly simple EFT algorithms. If an FMA instruction is available, then the error of an FP multiplication can be computed very easily and quickly using Algorithm 3 below.

Algorithm 3 – 2MultFMA(a, b). Returns a pair (π_h, π_ℓ) of FP numbers such that $\pi_h = \mathrm{RN}(ab)$ and $\pi_h + \pi_\ell = ab$. Requires the availability of an FMA

$\pi_h \leftarrow \mathrm{RN}(a \cdot b)$
$\pi_\ell \leftarrow \mathrm{RN}(a \cdot b - \pi_h)$
return (π_h, π_ℓ)

As one of the possible applications of our study is the evaluation of an FMA (to find again the algorithm presented in [6]), we may have, at least in that case, to assume that such an instruction is not available. In that case, we have to resort to a more complex algorithm, Algorithm 5 below, due to Dekker and Veltkamp [4]. To compute the product ab, we first "split" the input operands a and b into sub-operands of precision around $p/2$, so that the product of two such sub-operands is computed exactly with one floating-point multiplication. This preliminary splitting is done by Algorithm 4. For a proof of these algorithms, see [17]. More on splitting algorithms can be found in [9].

Algorithm 4 – Split(x, s). Veltkamp's splitting algorithm [4]. Returns a pair (x_h, x_ℓ) of FP numbers such that the significand of x_h fits in $s - p$ bits, the significand of x_ℓ fits in $s - 1$ bits, and $x_h + x_\ell = x$.

Require: $K = 2^s + 1$
Require: $2 \leq s \leq p - 2$
$\gamma \leftarrow \mathrm{RN}(K \cdot x)$
$\delta \leftarrow \mathrm{RN}(x - \gamma)$
$a_h \leftarrow \mathrm{RN}(\gamma + \delta)$
$a_\ell \leftarrow \mathrm{RN}(x - a_h)$
return (x_h, x_ℓ)

Algorithm 5 – DekkerProd(a, b). Dekker's product [4]. Returns a pair (π_h, π_ℓ) of FP numbers such that $\pi_h = \mathrm{RN}(ab)$ and $\pi_h + \pi_\ell = ab$.

Require: $s = \lceil p/2 \rceil$
$(a_h, a_\ell) \leftarrow \mathrm{Split}(a, s)$
$(b_h, b_\ell) \leftarrow \mathrm{Split}(b, s)$
$\pi_h \leftarrow \mathrm{RN}(a \cdot b)$
$t_1 \leftarrow \mathrm{RN}(-\pi_h + \mathrm{RN}(a_h \cdot b_h))$
$t_2 \leftarrow \mathrm{RN}(t_1 + \mathrm{RN}(a_h \cdot b_\ell))$
$t_3 \leftarrow \mathrm{RN}(t_2 + \mathrm{RN}(a_\ell \cdot b_h))$
$\pi_\ell \leftarrow \mathrm{RN}(t_3 + \mathrm{RN}(a_\ell \cdot b_\ell))$
return (π_h, π_ℓ)

1.2 Contents

As we will see, guaranteeing the correct rounding of the sum of 3 or 4 floating-point numbers will require checking whether a given "remainder" is a power of 2. An al-

gorithm for doing this is presented in Section 2. The computation of the rounded to nearest sum of 3 FP numbers, RN$(a + b + c)$, will be examined in Section 3. Interestingly enough, a slight modification of the corresponding algorithm will make it possible to compute the error of this computation, i.e., $a+b+c-\text{RN}(a+b+c)$. This error is expressed as the unevaluated sum of two FP numbers. This will be presented in Section 3.6. This will make it possible, for example, to compute the error of an FMA operation, giving an alternative to the algorithm presented in [3]. This will also allow the computation of the correctly-rounded sum of four FP numbers, RN$(a+b+c+d)$, as shown in Section 4.

2 Determining if the absolute value of a FP number is a power of 2

The algorithm presented in this section and its analysis already appear in [6]. We give Theorem 2.1 and Algorithm 6 here for the sake of completeness.

To determine whether the absolute value of an FP number x is a power of 2 using only "high level" FP operations, we use the following property

Theorem 2.1 ([6])**.** *In binary, precision-p, floating-point arithmetic, assuming no underflow/overflow occurs, the absolute value of the nonzero FP number x is a power of 2 if and only if*

$$\text{RN}\left[\text{RN}\left(\left(2^{p-1}+1\right)\cdot x\right)-2^{p-1}x\right]=x. \tag{2}$$

The proof of Theorem 2.1 is given in [6].

From Theorem 2.1 we obtain the following algorithm

Algorithm 6 IsPowerOf2(x) [6].

Require: $P = 2^{p-1} + 1$
Require: $Q = 2^{p-1}$
  $L \leftarrow \text{RN}(P \cdot x)$
  $R \leftarrow \text{RN}(Q \cdot x)$
  $\Delta \leftarrow \text{RN}(L - R)$
  return $(\Delta = x)$

We note that,

Remark 2.2. *When $x = 0$, IsPowerOf2(x) returns **true.***

3 Correctly-rounded sum of 3 FP numbers

Sections 3.1 to 3.5 are a simple generalization of the work we present in [6]. Nevertheless, we give here a detailed presentation, because many details will be useful for obtaining the error of the computation of the sum of 3 FP numbers (Section 3.6) and for computing the correctly-rounded sum of 4 FP numbers (Section 4).

3.1 First step: addition of a DW and a FP number

Assume we wish to compute

$$\Sigma = \mathrm{RN}(\hat{\Sigma}), \text{ with } \hat{\Sigma} = a + b + c,$$

where a, b, and c are FP numbers, using only rounded-to-nearest floating-point additions, subtractions, and comparisons. We know from Theorem 8 in [13] that an algorithm that only uses rounded-to-nearest additions and subtractions cannot evaluate Σ, hence comparisons cannot be fully avoided. Using Algorithm 2Sum (Algorithm 2) we first compute a double-word (x_h, x_ℓ) such that

$$x_h + x_\ell = a + b.$$

We are therefore reduced to computing the sum of a double-word and an FP number. An algorithm that computes a DW number close to the sum of a DW number and an FP number, Algorithm 7 below, was implemented in Hida, Li and Bailey's QD library [7]. That algorithm will not suffice for our purpose because it will not always return a correctly-rounded result: z_h is not always equal to $\mathrm{RN}(x_h + x_\ell + c)$.

Algorithm 7 – DWPlusFP(x_h, x_ℓ, c). Computes a DW close to $(x_h, x_\ell)+c$ in binary, precision-p, floating-point arithmetic. Implemented in the QD library. The number $x = x_h + x_\ell$ is a double-word number (i.e., it satisfies $x_h = \mathrm{RN}(x_h + x_\ell)$.

1: $(s_h, s_\ell) \leftarrow \mathrm{2Sum}(x_h, c)$
2: $v \leftarrow \mathrm{RN}(x_\ell + s_\ell)$
3: $(z_h, z_\ell) \leftarrow \mathrm{Fast2Sum}(s_h, v)$
4: **return** (z_h, z_ℓ)

Algorithm 7 is analyzed in [10], where the following theorem is proven.

Theorem 3.1. *The pair (z_h, z_ℓ) returned by Algorithm 7 is a DW number. it safisfies:*

$$|(z_h + z_\ell) - (x + c)| \leq 2u^2 \cdot |x + c|\,. \tag{3}$$

Below, following the same steps as in [6], we analyze the various cases that may occur when trying to compute $\mathrm{RN}(x_h + x_\ell + c)$ using a modified version of Algorithm 7. The calculation will be simple, unless some intermediate variable (variable w in Algorithm 8) is a power of 2. We have already seen how that case can be detected in Section 2. We will examine how it can be dealt with in Section 3.3.

3.2 Computing $\mathrm{RN}(x_h + x_\ell + c)$: general case

Let us modify Algorithm 7 and compute

$$\begin{cases} (s_h, s_\ell) &= \mathrm{2Sum}(x_h, c) \\ (v_h, v_\ell) &= \mathrm{2Sum}(x_\ell, s_\ell) \\ (z_h, z_\ell) &= \mathrm{Fast2Sum}(s_h, v_h) \end{cases}$$

(one easily sees that v_h is the variable "v" of Algorithm 7). We obviously have

$$z_h + z_\ell + v_\ell = a + b + c = \hat{\Sigma}, \tag{4}$$

and Theorem 3.1 tells us that (z_h, z_ℓ) is a double-word, i.e., $z_h = \mathrm{RN}(z_h + z_\ell)$, and

$$|v_\ell| = |(z_h + z_\ell) - \hat{\Sigma}| \leq 2u^2|\hat{\Sigma}|, \tag{5}$$

so that

$$|\hat{\Sigma}| \leq \frac{|z_h + z_\ell|}{1 - 2u^2}. \tag{6}$$

Let us momentarily put aside the case $a+b+c = 0$. From (5), (6), and $|z_\ell| \leq u|z_h|$, we obtain

$$|v_\ell| \leq 2u^2|\hat{\Sigma}| \leq \frac{2u^2(1+u)}{1-2u^2}\,|z_h|\,. \tag{7}$$

- If $|z_h|$ is not a power of 2 then $|z_\ell| \leq \frac{1}{2}\mathrm{ulp}(z_h)$ and, as $\mathrm{ulp}(z_h) \geq \frac{u}{1-u}|z_h|$, (7) implies
$$|v_\ell| \leq \frac{2u(1-u^2)}{1-2u^2}\mathrm{ulp}(z_h) < \frac{1}{4}\mathrm{ulp}(z_h)$$
(as soon as $u \leq 1/16$), so that
$$|z_\ell + v_\ell| < \frac{3}{4}\mathrm{ulp}(z_h),$$
which also implies (as RN is an increasing function, and $\frac{3}{4}\mathrm{ulp}(z_h)$ is a FP number since $\mathrm{ulp}(z_h)$ is a power of 2)
$$|\mathrm{RN}(z_\ell + v_\ell)| \leq \frac{3}{4}\mathrm{ulp}(z_h). \tag{8}$$

- If $|z_h|$ is a power of 2 then
$$-\frac{1}{4}\mathrm{ulp}(z_h) \leq z_\ell \times \mathrm{sign}(z_h) \leq \frac{1}{2}\mathrm{ulp}(z_h),$$
and, as $\mathrm{ulp}(z_h) = 2u|z_h|$, (7) implies
$$|v_\ell| \leq \frac{u(1+u)}{1-2u^2}\mathrm{ulp}(z_h) < \frac{1}{8}\mathrm{ulp}(z_h)$$
(as soon as $u \leq 1/16$), so that
$$-\frac{3}{8}\mathrm{ulp}(z_h) < (z_\ell + v_\ell) \times \mathrm{sign}(z_h) < \frac{5}{8}\mathrm{ulp}(z_h),$$
which also implies (as RN is an increasing function, and $\frac{5}{8}\mathrm{ulp}(z_h)$ is a FP number since $\mathrm{ulp}(z_h)$ is a power of 2)
$$-\frac{3}{8}\mathrm{ulp}(z_h) \leq \mathrm{RN}(z_\ell + v_\ell) \times \mathrm{sign}(z_h) \leq \frac{5}{8}\mathrm{ulp}(z_h). \tag{9}$$

Therefore, as soon as $u \leq 1/16$,

$$z_h^- < \hat{\Sigma} < z_h^+,$$

where z_h^- and z_h^+ are the floating-point predecessor and successor of z_h, respectively.

When the absolute value of the number $w = \mathrm{RN}(v_\ell + z_\ell)$ is *not* a power of 2, the number $|v_\ell + z_\ell|$ is not a power of 2 either (otherwise it would round to itself), and in that case $|w| > \frac{1}{2}\mathrm{ulp}(z_h)$ (resp. $|w| > \frac{1}{4}\mathrm{ulp}(z_h)$) iff $|v_\ell + z_\ell| > \frac{1}{2}\mathrm{ulp}(z_h)$ (resp. $|v_\ell + z_\ell| > \frac{1}{4}\mathrm{ulp}(z_h)$). Also, note that (8) and (9) imply that

$$z_h^- < z_h + w < z_h^+.$$

Therefore,

if $u \leq 1/16$ then when $|w| = |\mathrm{RN}(v_\ell + z_\ell)|$ is not a power of 2, Σ is equal to $\mathrm{RN}(z_h + \mathrm{RN}(z_\ell + v_\ell))$.

3.3 When $|\mathrm{RN}(v_\ell + z_\ell)|$ is a power of 2

Now assume that the absolute value of $w = \mathrm{RN}(z_\ell + v_\ell)$ is a power of 2. We need to determine if $\mathrm{RN}(z_h + w)$ differs from $\mathrm{RN}(z_h + z_\ell + v_\ell)$. An easy case is when $|w| < \chi$, where χ is the "critical power of 2", defined as

- $\frac{1}{2}\mathrm{ulp}(z_h)$ if $|z_h|$ is not a power of 2; or if $|z_h|$ is a power of 2 and z_h and w have the same sign;
- $\frac{1}{4}\mathrm{ulp}(z_h)$ if $|z_h|$ is a power of 2 and z_h and w have opposite signs.

Note that (8) and (9) imply $|w| \leq \chi$ when $|w|$ is a power of 2. Let $w' = \mathrm{RN}\left(\frac{3}{2}w\right) = \frac{3}{2}w$. We have $|w| < \chi$ if and only if $\mathrm{RN}(z_h + w') = z_h$. In such a case, we are done: the result to be returned is z_h.

Figure 1 illustrates this discussion, in the general case where $|z_h|$ is not a power of 2.

Now, when $|w| = \chi$, we need to determine if $|z_\ell + v_\ell|$ is equal to, above, or below χ. Property 3.2 below implies that this can be done using the Fas2Sum algorithm. More precisely, if we compute

$$\begin{cases} \delta &= \mathrm{RN}(w - z_\ell) \\ t &= \mathrm{RN}(v_\ell - \delta), \end{cases}$$

then $w + t = z_\ell + v_\ell$. The choice is now simple:

- if $t = 0$ then $w = z_\ell + v_\ell$, so that $\Sigma = \mathrm{RN}(z_h + w)$;
- if $t \neq 0$ and w have opposite signs, then $|z_\ell + v_\ell| < \chi$, so that $\Sigma = z_h$;
- if $t \neq 0$ and w have the same sign, then $|z_\ell + v_\ell| > \chi$. As a consequence, Σ is the FP predecessor or successor of z_h (depending on the sign of w), which can be obtained as $\Sigma = \mathrm{RN}(z_h + w')$, using $w' = \mathrm{RN}\left(\frac{3}{2}w\right) = \frac{3}{2}w$, as previously.

The following property, presented and proven in [6], shows that we can use the Fast2Sum algorithm for adding z_ℓ and v_ℓ (which saves 3 operations since otherwise we would have needed to use the 2Sum algorithm).

Property 3.2 ([6]). *When $|w| = \chi$, we have $|v_\ell| \leq |z_\ell|$ as soon as $u \leq 1/16$.*

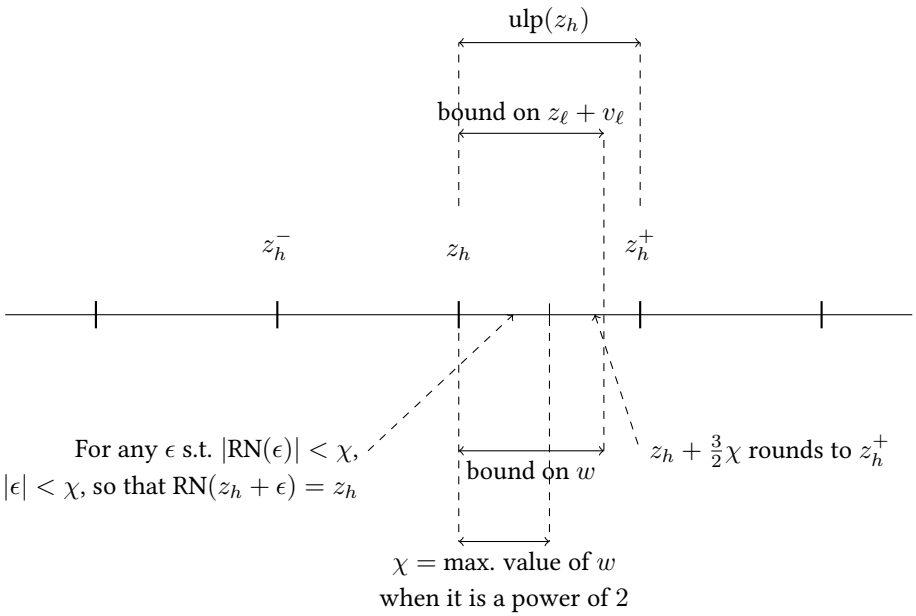

Figure 1: Illustration of the discussion presented in Sections 3.2 and 3.3, in the general case where $|z_h|$ is not a power of 2. For the sake of simplicity, we assume here that $z_\ell + v_\ell \geq 0$.

3.4 Putting all this together: the Sum3 algorithm

Algorithm 8 below derives from the analysis given in the previous sections. In the analysis, we put aside the case $a + b + c = 0$, but in that case, one easily checks that

$$z_h = z_\ell = v_h = v_\ell = w = w' = \sigma_{\text{temp2}} = 0.$$

As a consequence, from Remark 2.2, the test "IsPowerOf2(w)" at Line 7 of Algorithm 8 and the test "$\sigma_{\text{temp2}} = z_h$" at Line 12 both return **true** so that the algorithm returns z_h, i.e., 0, which is the right answer.

Algorithm 8 Sum3(a, b, c).

1: $(x_h, x_\ell) \leftarrow$ 2Sum(a, b)
2: $(s_h, s_\ell) \leftarrow$ 2Sum(x_h, c)
3: $(v_h, v_\ell) \leftarrow$ 2Sum(x_ℓ, s_ℓ)
4: $(z_h, z_\ell) \leftarrow$ Fast2Sum(s_h, v_h)
5: $w \leftarrow \mathrm{RN}(v_\ell + z_\ell)$
6: $\sigma_{\text{temp1}} \leftarrow \mathrm{RN}(z_h + w)$
7: **if not** IsPowerOf2(w) **then**
8: **return** σ_{temp1}
9: **else**
10: $w' \leftarrow \mathrm{RN}\left(\frac{3}{2} \cdot w\right)$
11: $\sigma_{\text{temp2}} \leftarrow \mathrm{RN}(z_h + w')$
12: **if** $\sigma_{\text{temp2}} = z_h$ **then**
13: **return** z_h
14: **else**
15: $\delta \leftarrow \mathrm{RN}(w - z_\ell)$
16: $t \leftarrow \mathrm{RN}(v_\ell - \delta)$
17: **if** $t = 0$ **then**
18: **return** σ_{temp1}
19: **else**
20: $g \leftarrow \mathrm{RN}(t \cdot w)$
21: **if** $g < 0$ **then**
22: **return** z_h
23: **else**
24: **return** σ_{temp2}
25: **end if**
26: **end if**
27: **end if**
28: **end if**

We have,

Theorem 3.3. *In a binary, precision-p, floating-point arithmetic with an unbounded exponent range, if $p \geq 4$, then Algorithm 8 returns* $\mathrm{RN}(a + b + c)$ *for all floating-point numbers a, b, and c.*

Proof. The theorem immediately follows from the analysis of Sections 3.2 and 3.3, and the fact that $p \geq 4$ implies $u \leq 1/16$. □

The primary disadvantage of our algorithm is the presence of tests. In the event that the branch prediction mechanism of the processor fails to make the correct prediction, these tests may result in a significant reduction in performance. However, it is important to note that the value of $|w|$ is very unlikely to be a power of 2. Consequently, when a large number of Sum3 are computed, the branch prediction should function effectively, whereas when a small number of Sum3 are computed, the performance loss is of minimal consequence. Secondly, and more importantly, tests cannot

be entirely avoided: the authors of [13] have shown that an algorithm that only uses rounded-to-nearest additions and subtractions cannot evaluate RN$(a+b+c)$ for all possible FP numbers a, b, and c (Theorem 8 in [13]).

3.5 Emulation of the FMA

As Algorithm 8 is essentially an algorithm that returns the correctly-rounded sum of a DW (x_h, x_ℓ) and a FP number c, to obtain an algorithm that computes RN$(ab+c)$, it suffices to make sure that $x_h+x_\ell = ab$. This is done by replacing Line 1 of Algorithm 8 by

$$(x_h, x_\ell) \leftarrow \text{DekkerProd}(a, b).$$

By doing this we find again the FMA emulation algorithm introduced in [6].

3.6 Computing RN$(a+b+c)$ and the error $(a+b+c)-$RN$(a+b+c)$ at the same time

Small modifications of Algorithm 8 make it possible to obtain, at the same time, RN$(a+b+c)$ and the error of that addition, namely $a+b+c-$RN$(a+b+c)$, at a very small additional cost. That error (which in general does not fit in only one FP number) is expressed as the unevaluated sum of two FP numbers. Let us detail the analysis:

In all cases, Algorithm 8 outputs $\Sigma = \text{RN}(z_h + z_\ell + v_\ell)$, and we know that $\Sigma \in \{z_h^-, z_h, z_h^+\}$. Hence, by Sterbenz Theorem (Theorem 1.1), the number

$$\alpha = \Sigma - z_h$$

is a FP number (so that it is exactly computed by a FP subtraction: $\alpha = \text{RN}(\Sigma - z_h)$). Also, α can be zero, $\pm\text{ulp}(z_h)$, or $\pm\frac{1}{2}\text{ulp}(z_h)$ (the last case being possible only when z_h is a power of 2 and the sign of $z_\ell + v_\ell$ differs from the sign of z_h).

- if $\Sigma = z_h$, as $a+b+c = z_h + z_\ell + v_\ell$, then the error is $z_\ell + v_\ell = w + t$;
- if $\Sigma \neq z_h$, we know that α has the same sign as $z_\ell + v_\ell$, i.e., the same sign as $w = \text{RN}(z_\ell + v_\ell)$. We also know that
  - when (z_h is not a power of 2) or (z_h is a power of 2 and $z_\ell+v_\ell$ has the same sign as z_h), we have $|\alpha| = \text{ulp}(z_h)$; $|w| \geq \frac{1}{2}\text{ulp}(z_h)$ (otherwise we would have $|z_\ell + v_\ell| < \frac{1}{2}\text{ulp}(z_h)$ and therefore $\Sigma = z_h$); and $|w| \leq \text{ulp}(z_h)$ (otherwise we would have $|z_\ell + v_\ell| > \text{ulp}(z_h)$, which is not allowed by the analysis of Section 3.2);
  - when z_h is a power of 2 and the sign of $z_\ell + v_\ell$ differs from the sign of z_h, we have $|\alpha| = \frac{1}{2}\text{ulp}(z_h)$; $|w| \geq \frac{1}{4}\text{ulp}(z_h)$ (otherwise we would have $|z_\ell + v_\ell| < \frac{1}{4}\text{ulp}(z_h)$ and therefore $\Sigma = z_h$); and $|w| \leq \frac{1}{2}\text{ulp}(z_h)$ (otherwise we would have $|z_\ell + v_\ell| > \frac{1}{2}\text{ulp}(z_h)$, which is not allowed by the analysis of Section 3.2).

Therefore we always have $\frac{1}{2}|\alpha| \leq |w| \leq |\alpha|$ so that (as α and w have the same sign), Sterbenz theorem implies that $\eta = w - \alpha$ is a floating-point number and is therefore exactly computed in FP arithmetic: $\eta = \mathrm{RN}(w - \alpha) = w - \alpha$. We therefore deduce

$$(a+b+c)-\Sigma = z_h+z_\ell+v_\ell-\Sigma = -\alpha+z_\ell+v_\ell = -\alpha+w+(z_\ell+v_\ell-w) = \eta+t.$$

Hence, in all cases, the error $(a + b + c) - \Sigma$ is equal to $\mathrm{RN}(w - \alpha) + t = \eta + t$. One can just output the pair (η, t). If one prefers returning a DW number, as η is a multiple of $\mathrm{ulp}(w)$ and $\mathrm{ulp}(t) \leq \mathrm{ulp}(w)$ (since the pair (w, t) is a DW), one can safely call $\mathrm{Fast2Sum}(\eta, t)$. If one only needs to know the FP number nearest the error (and therefore the sign of the error), it suffices to return $\mathrm{RN}(\eta+t)$. This gives the following algorithm.

Algorithm 9 Sum3-with-error(a, b, c).

$(x_h, x_\ell) \leftarrow$ 2Sum(a, b)
$(s_h, s_\ell) \leftarrow$ 2Sum(x_h, c)
$(v_h, v_\ell) \leftarrow$ 2Sum(x_ℓ, s_ℓ)
$(z_h, z_\ell) \leftarrow$ Fast2Sum(s_h, v_h)
$w \leftarrow \mathrm{RN}(v_\ell + z_\ell)$
$\sigma_{\text{temp1}} \leftarrow \mathrm{RN}(z_h + w)$
$\delta \leftarrow \mathrm{RN}(w - z_\ell)$
$t \leftarrow \mathrm{RN}(v_\ell - \delta)$
if not IsPowerOf2(w) **then**
  $\Sigma \leftarrow \sigma_{\text{temp1}}$
else
  $w' \leftarrow \mathrm{RN}\left(\frac{3}{2} \cdot w\right)$
  $\sigma_{\text{temp2}} \leftarrow \mathrm{RN}(z_h + w')$
  if $\sigma_{\text{temp2}} = z_h$ **then**
    $\Sigma \leftarrow z_h$
  else
    if $t = 0$ **then**
      $\Sigma \leftarrow \sigma_{\text{temp1}}$
    else
      $g \leftarrow \mathrm{RN}(t \cdot w)$
      if $g < 0$ **then**
        $\Sigma \leftarrow z_h$
      else
        $\Sigma \leftarrow \sigma_{\text{temp2}}$
      end if
    end if
  end if
end if
$\alpha \leftarrow \mathrm{RN}(\Sigma - z_h)$
$\eta \leftarrow \mathrm{RN}(w - \alpha)$
$(\eta, t) \leftarrow$ Fast2Sum(η, t) % omitted if not required that the error is a DW
% replaced by $\eta \leftarrow \mathrm{RN}(\eta + t)$ if we only need the FP number nearest the error
return (Σ, η, t)

Obviously, the very same adaptation as the one done in Section 3.5 will give an algorithm that returns an FMA and the error of that FMA.

4 Computation of $\mathrm{RN}(a + b + c + d)$ and $\mathrm{RN}(ab + cd)$

Let us now focus on the calculation of the correctly-rounded sum of two double-word numbers: we wish to evaluate

$$\Sigma = \mathrm{RN}\left(\hat{\Sigma}\right), \text{ with } \hat{\Sigma} = x_h + x_\ell + y_h + y_\ell,$$

where (x_h, x_ℓ) and (y_h, y_ℓ) are DW numbers. This will allow us to evaluate the correctly-rounded sum of four arbitrary floating-point numbers a, b, c, and d by choosing $(x_h, x_\ell) = \text{2Sum}(a, b)$ and $(y_h, y_\ell) = \text{2Sum}(c, d)$. This will also allow us to evaluate

$$\text{RN}(ab + cd),$$

where, again, a, b, c, and d are four arbitrary floating-point numbers, by choosing $(x_h, x_\ell) = \text{2Mult}(a, b)$ and $(y_h, y_\ell) = \text{2Mult}(c, d)$. Here, "2Mult" is either DekkerProd (Algorithm 5) if no FMA instruction is available, or the much simpler algorithm 2MultFMA (Algorithm 3, called Fast2Mult in [11, 19, 17]) if we can use an FMA instruction.[2]

To design an algorithm that evaluates Σ we will start from the following double-word addition Algorithm, presented in [15], and analyzed in [10].

Algorithm 10 – AccurateDWPlusDW$(x_h, x_\ell, y_h, y_\ell)$. Calculation of a DW number close to $(x_h, x_\ell) + (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic.

1: $(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, y_h)$
2: $(t_h, t_\ell) \leftarrow \text{2Sum}(x_\ell, y_\ell)$
3: $\gamma \leftarrow \text{RN}(s_\ell + t_h)$
4: $(v_h, v_\ell) \leftarrow \text{Fast2Sum}(s_h, \gamma)$
5: $w \leftarrow \text{RN}(t_\ell + v_\ell)$
6: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(v_h, w)$
7: **return** (z_h, z_ℓ)

The authors of [10] have shown that as soon as $p \geq 3$, the pair (z_h, z_ℓ) returned by Algorithm 10 is a DW number, and it satisfies

$$|(z_h + z_\ell) - (x_h + x_\ell + y_h + y_\ell)| \leq \frac{3u^2}{1 - 4u} \cdot |x_h + x_\ell + y_h + y_\ell| \,. \tag{10}$$

(that relative error bound becomes less than $3u^2 + 13u^3$ as soon as $p \geq 6$). A formal proof of that result, and the proof that the bound is asymptotically optimal, are presented in [18]. The two operations that are not error-free in Algorithm 10 are the ones of lines 3 and 5. Let us replace them by a 2Sum and a Fast2Sum operations respectively,[3] and obtain

1: $(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, y_h)$
2: $(t_h, t_\ell) \leftarrow \text{2Sum}(x_\ell, y_\ell)$
3: $(\gamma_h, \gamma_\ell) \leftarrow \text{2Sum}(s_\ell, t_h)$
4: $(v_h, v_\ell) \leftarrow \text{Fast2Sum}(s_h, \gamma_h)$
5: $(w_h, w_\ell) \leftarrow \text{Fast2Sum}(v_\ell, t_\ell)$
6: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(v_h, w_h)$

[2] Of course, we needed the DekkerProd algorithm in Section 3.5: as the goal was to emulate an FMA instruction, we could not assume that that instruction was available.

[3] A Fast2Sum can be used at Line 5 because t_ℓ can be written $K \cdot \text{ulp}\left(\min\{|x_\ell|, |y_\ell|\}\right)$ with $K \leq 2^p - 1$, and v_ℓ is by construction an integer multiple of $\text{ulp}\left(\min\{|x_\ell|, |y_\ell|\}\right)$.

Variables γ_h and w_h are the same as variables γ and w in Algorithm 10, so that the double-word (z_h, z_ℓ) still satisfies (10), but we also have (as there is no longer any inexact operation):

$$z_h + z_\ell + w_\ell + \gamma_\ell = x_h + x_\ell + y_h + y_\ell.$$

Note that (10) implies that z_h cannot be zero, unless $\hat{\Sigma} = 0$. Let us set this case aside for the moment. We have

$$|w_\ell + \gamma_\ell| \leq \frac{3u^2}{1-4u} \cdot \left|\hat{\Sigma}\right|. \tag{11}$$

As (z_h, z_ℓ) is a DW number, $|z_\ell|$ is less than or equal to

- $\frac{1}{2}\mathrm{ulp}(z_h)$ if z_h is not a power of 2 or z_ℓ has the sign of z_h;
- $\frac{1}{4}\mathrm{ulp}(z_h)$ if z_h is a power of 2 and the signs of z_h and z_ℓ differ.

Define $Z = z_h + z_\ell$. From (11) we obtain

$$\left(1 - \frac{3u^2}{1-4u}\right) \cdot \left|\hat{\Sigma}\right| \leq |Z| \leq \left(1 + \frac{3u^2}{1-4u}\right) \cdot \left|\hat{\Sigma}\right|,$$

and therefore

$$\left|\hat{\Sigma}\right| \leq \frac{|Z|}{1 - \frac{3u^2}{1-4u}} = |Z| \cdot \frac{1-4u}{1-4u-3u^2} \leq |z_h| \cdot \frac{(1-4u)(1+u)}{1-4u-3u^2}$$

So that, combining with (11),

$$|w_\ell + \gamma_\ell| \leq \frac{3u^2}{1-4u} \cdot \frac{(1-4u)(1+u)}{1-4u-3u^2} \cdot |z_h| = \frac{3u^2+3u^3}{1-4u-3u^2} \cdot |z_h|.$$

- if $|z_h|$ is not a power of 2, from $\mathrm{ulp}(z_h) \geq \frac{u}{1-u}|z_h|$, we obtain

  $$|w_\ell + \gamma_\ell| \leq \frac{(3u+3u^2)(1-u)}{1-4u-3u^2} \cdot \mathrm{ulp}(z_h) = \frac{3u-3u^3}{1-4u-3u^2} \cdot \mathrm{ulp}(z_h),$$

  which is $\leq \frac{85}{336}\mathrm{ulp}(z_h) = \left(\frac{1}{4} + \frac{1}{336}\right)\mathrm{ulp}(z_h)$ as soon as $u \leq \frac{1}{16}$;
- if $|z_h|$ is a power of 2 then $\mathrm{ulp}(z_h) = 2u|z_h|$, so that

  $$|w_\ell + \gamma_\ell| \leq \frac{3u+3u^2}{2-8u-6u^2}\mathrm{ulp}(z_h),$$

  which is $\leq \frac{17}{126}\mathrm{ulp}(z_h) = \left(\frac{1}{8} + \frac{5}{504}\right)\mathrm{ulp}(z_h)$ as soon as $u \leq \frac{1}{16}$.

Combining this with the bound on $|z_\ell|$, we therefore deduce that, as soon as $u \leq 1/16$ (or, equivalently, as soon as $p \geq 4$),

- if $|z_h|$ is not a power of 2 then

$$|z_\ell + w_\ell + \gamma_\ell| < \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{16}\right) \text{ulp}(z_h),$$

and

$$|\text{RN}\left(z_\ell + w_\ell + \gamma_\ell\right)| \leq \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{16}\right) \text{ulp}(z_h); \tag{12}$$

- if $|z_h|$ is a power of 2 and the signs of z_h and z_ℓ differ then

$$|z_\ell + w_\ell + \gamma_\ell| < \left(\frac{1}{4} + \frac{1}{8} + \frac{1}{16}\right) \text{ulp}(z_h),$$

and

$$|\text{RN}\left(z_\ell + w_\ell + \gamma_\ell\right)| \leq \left(\frac{1}{4} + \frac{1}{8} + \frac{1}{16}\right) \text{ulp}(z_h); \tag{13}$$

- if $|z_h|$ is a power of 2 and z_h and z_ℓ have the same sign then
  - bound (12) holds if $z_\ell + w_\ell + \gamma_\ell$ has the same sign as z_ℓ;
  - if the sign of $z_\ell + w_\ell + \gamma_\ell$ differs from the sign of z_ℓ then

$$|z_\ell + w_\ell + \gamma_\ell| \leq |w_\ell + \gamma_\ell| \leq \left(\frac{1}{8} + \frac{1}{16}\right) \text{ulp}(z_h),$$

and

$$|\text{RN}\left(z_\ell + w_\ell + \gamma_\ell\right)| \leq \left(\frac{1}{8} + \frac{1}{16}\right) \text{ulp}(z_h). \tag{14}$$

Hence, $\Sigma = \text{RN}(x_h + x_\ell + y_h + y_\ell) = \text{RN}(z_h + z_\ell + w_\ell + \gamma_\ell)$ and $\text{RN}(z_h + \text{RN}(z_\ell + w_\ell + \gamma_\ell))$ are either z_h^-, z_h, or z_h^+. As a consequence, the very same reasoning as the one of Section 3 can be applied. We first compute $\rho = \text{RN}(z_\ell + w_\ell + \gamma_\ell)$ and the error term $\tau = \text{RN}(z_\ell + w_\ell + \gamma_\ell - \rho)$ using Algorithm 9.

- if $|\rho|$ is *not* a power of 2 (which is checked using Algorithm 6) then

$$\Sigma = \text{RN}(z_h + \rho),$$

- if $|\rho|$ is a power of 2 then (12), (13), and (14) imply that ρ cannot be larger than the "critical" power of 2, defined as $\frac{1}{2}\text{ulp}(z_h)$ if ($|z_h|$ is not a power of 2) or ($|z_h|$ is a power of 2 and the signs of z_h and $z_\ell + w_\ell + \gamma_\ell$ are the same), and as $\frac{1}{2}\text{ulp}(z_h)$ if ($|z_h|$ is a power of 2 and the signs of z_h and $z_\ell + w_\ell + \gamma_\ell$ differ). We deduce that
  - if $\text{RN}\left(z_h + \frac{3}{2}\rho\right) = z_h$ then ρ is strictly less than the "critical" power of 2, and in that case, $\Sigma = z_h$;

– otherwise, ρ is equal to the critical power of 2, and in that case,

$$\Sigma = \begin{cases} \mathrm{RN}(z_h + \rho) & \text{if } \tau = 0, \\ z_h & \text{if the signs of } \rho \text{ and } \tau \text{ differ}, \\ \mathrm{RN}\left(z_h + \frac{3}{2}\rho\right) & \text{otherwise} \end{cases}$$

We did put aside the case $\hat{\Sigma} = 0$. Let us deal with it now. In that case, $x_h + x_\ell + y_h + y_\ell = 0$, i.e.,

$$(x_h + x_\ell) = -(y_h + y_\ell),$$

and therefore

$$\mathrm{RN}(x_h + x_\ell) = \mathrm{RN}\left(-(y_h + y_\ell)\right) = -\mathrm{RN}(y_h + y_\ell). \tag{15}$$

As (x_h, x_ℓ) and (y_h, y_ℓ) are DW numbers, $x_h = \mathrm{RN}(x_h + x_\ell)$ and $y_h = \mathrm{RN}(y_h + y_\ell)$, so that (15) implies $x_h = -y_h$ and therefore $x_\ell = -y_\ell$. It follows that

$$s_h = s_\ell = t_h = t_\ell = \gamma_h = \gamma_\ell = v_h = v_\ell = w_h = w_\ell = z_h = z_\ell = \rho = 0.$$

Remark 2.2 implies that in that case **IsPowerOf2**(ρ) returns **true**. Also, $\mathrm{RN}(z_h + \frac{3}{2}\rho) = 0 = z_h$. Note that in the "general case" considered before, when ρ is a power of 2 and $\mathrm{RN}(z_h + \frac{3}{2}\rho) = z_h$, we return z_h. In the case considered now, as $z_h = 0$, returning z_h is a correct answer. Hence there is no need for a separate handling of the case $\Sigma = 0$.

The algorithm is therefore,

Algorithm 11 FPNearestSumDW$(x_h, x_\ell, y_h, y_\ell)$. Returns RN$(x_h + x_\ell + y_h + y_\ell)$ for all pairs (x_h, x_ℓ), (y_h, y_ℓ) of double-word numbers.

$(s_h, s_\ell) \leftarrow$ 2Sum(x_h, y_h)
$(t_h, t_\ell) \leftarrow$ 2Sum(x_ℓ, y_ℓ)
$(\gamma_h, \gamma_\ell) \leftarrow$ 2Sum(s_ℓ, t_h)
$(v_h, v_\ell) \leftarrow$ Fast2Sum(s_h, γ_h)
$(w_h, w_\ell) \leftarrow$ Fast2Sum(v_ℓ, t_ℓ)
$(z_h, z_\ell) \leftarrow$ Fast2Sum(v_h, w_h)
$(\rho, \tau) \leftarrow$ **Sum3-with-error**$(z_\ell, w_\ell, \gamma_\ell)$
if IsPowerOf2(ρ) **then**
  $\rho' \leftarrow$ RN$\left(\frac{3}{2}\rho\right)$
  $\sigma_{\text{temp}} \leftarrow$ RN$(z_h + \rho')$
  if $\sigma_{\text{temp}} = z_h$ **then**
    $\Sigma \leftarrow z_h$
  else
    if $\tau = 0$ **then**
      $\Sigma \leftarrow$ RN$(z_h + \rho)$
    else
      if RN$(\rho \cdot \tau) \leq 0$ **then**
        $\Sigma \leftarrow z_h$
      else
        $\Sigma \leftarrow \sigma_{\text{temp}}$
      end if
    end if
  end if
else
  $\Sigma \leftarrow z_h$
end if
return Σ

We conclude,

Theorem 4.1. *In a binary, precision-p, floating-point arithmetic with an unbounded exponent range, if $p \geq 4$, then Algorithm 11 returns* RN$(x_h + x_\ell + y_h + y_\ell)$ *for all pairs (x_h, x_ℓ), (y_h, y_ℓ) of double-word numbers.*

We immediately deduce the following two algorithms.

Algorithm 12 Sum4(a, b, c, d). Returns RN$(a + b + c + d)$ for all 4-tuples (a, b, c, d) of floating-point numbers.

$(x_h, x_\ell) \leftarrow$ 2Sum(a, b)
$(y_h, y_\ell) \leftarrow$ 2Sum(c, d)
$s \leftarrow$ FPNearestSumDW$(x_h, x_\ell, y_h, y_\ell)$
return s

Algorithm 13 FD2(a, b, c, d). Returns RN$(ab + cd)$ for all 4-tuples (a, b, c, d) of floating-point numbers. Here, 2Mult is either DekkerProd (Algorithm 5) if no FMA instruction is available, or 2MultFMA (Algorithm 3) if we can use an FMA.

$(x_h, x_\ell) \leftarrow$ 2Mult(a, b)
$(y_h, y_\ell) \leftarrow$ 2Mult(c, d)
$\pi \leftarrow$ FPNearestSumDW$(x_h, x_\ell, y_h, y_\ell)$
return π

Conclusion

The algorithms presented in this paper provide robust methods for computing the correctly-rounded sums of three or four floating-point numbers using binary floating-point arithmetic, and for emulating the fused multiply-add (FMA) and the fused dot-product (FD2) instructions. These algorithms ensure accurate and efficient computations without relying on additional hardware support. In fact, they rely only on conditional statements and classic floating-point operations such as addition, subtraction and multiplication. These algorithms provide building blocks for accurate computation in complex floating-point arithmetic and can be used in many applications, such as Fast Fourier Transforms.

Acknowledgement

This work was partly supported by the NuSCAP (ANR-20-CE48-0014) project and the InterFLOP (ANR-20-CE46-0009) project of the French National Agency for Research (ANR).

References

[1] IEEE standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, July 2019.

[2] Sylvie Boldo, Claude-Pierre Jeannerod, Guillaume Melquiond, and Jean-Michel Muller. Floating-point arithmetic. *Acta Numerica*, 32:203–290, 2023.

[3] Sylvie Boldo and Jean-Michel Muller. Exact and approximated error of the FMA. *IEEE Transactions on Computers*, 60(2):157–164, 2011.

[4] T. J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.

[5] Brian Gladman, Vincenzo Innocente, John Mather, and Paul Zimmermann. Accuracy of mathematical functions in single, double, extended double and quadruple precision. Working paper or preprint, available at `https://hal.inria.fr/hal-03141101`, 2024.

[6] Stef Graillat and Jean-Michel Muller. Emulation of the FMA in rounded-to-nearest floating-point arithmetic. Research report hal-04575249, available at `https://hal.science/hal-04575249`, May 2024.

[7] Y. Hida, X. S. Li, and D. H. Bailey. C++/fortran-90 double-double and quad-double package, release 2.3.17, March 2012. Accessible electronically at `http://crd-legacy.lbl.gov/dhbailey/mpdist/`.

[8] Tom Hubrecht, Claude-Pierre Jeannerod, and Jean-Michel Muller. Useful applications of correctly-rounded operators of the form $ab + cd + e$. In *IEEE 31st Symposium on Computer Arithmetic (ARITH 2024)*, volume IEEE 31st Symposium on Computer Arithmetic (ARITH), Málaga, Spain, June 2024.

[9] Claude-Pierre Jeannerod, Jean-Michel Muller, and Paul Zimmermann. On various ways to split a floating-point number. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 53–60, 2018.

[10] Mioara Joldeş, Jean-Michel Muller, and Valentina Popescu. Tight and rigourous error bounds for basic building blocks of double-word arithmetic. *ACM Transactions on Mathematical Software*, 44(2), 2017.

[11] W. Kahan. Lecture notes on the status of IEEE-754. Available at `http://www.cs.berkeley.edu/wkahan/ieee754status/IEEE754.PDF`, 1997.

[12] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, MA, 3rd edition, 1998.

[13] Peter Kornerup, V. Lefèvre, N. Louvet, and J.-M. Muller. On the computation of correctly-rounded sums. *IEEE Transactions on Computers*, 61(2):289–298, March 2012.

[14] Christoph Lauter. An efficient software implementation of correctly rounded operations extending FMA: $a + b + c$ and $a \times b + c \times d$. In *ACSSC Proc.*, 2017.

[15] X. Li, J. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, A. Kapur, M. Martin, T. Tung, and D. J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, 2002.

[16] O. Møller. Quasi double-precision in floating-point addition. *BIT*, 5:37–50, 1965.

[17] Jean-Michel Muller, Nicolas Brunie, Florent de Dinechin, Claude-Pierre Jeannerod, Mioara Joldes, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, and Serge Torres. *Handbook of Floating-Point Arithmetic, 2nd edition*. Birkhäuser Boston, 2018.

[18] Jean-Michel Muller and Laurence Rideau. Formalization of double-word arithmetic, and comments on "Tight and rigorous error bounds for basic building blocks of double-word arithmetic". *ACM Transactions on Mathematical Software*, 48(2):1–24, 2022.

[19] Y. Nievergelt. Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. *ACM Transactions on Mathematical Software*, 29(1):27–48, 2003.

[20] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.

[21] P. H. Sterbenz. *Floating-Point Computation.* Prentice-Hall, Englewood Cliffs, NJ, 1974.